

UNIVERSITY OF CALGARY

A Technique for Modeling the Performance of Session-Based Systems with Bursty
Request Arrivals

by

Min Xu

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

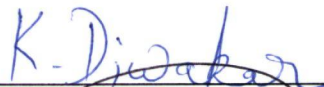
CALGARY, ALBERTA

September, 2008

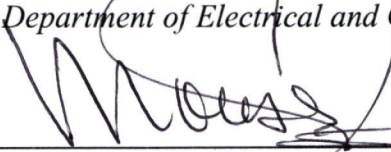
© Min Xu 2008

UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

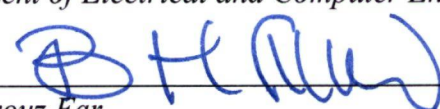
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled “A Technique for Modeling the Performance of Session-Based Systems with Bursty Request Arrivals” submitted by Min Xu in partial fulfilment of the requirements of the degree of Master of Science.




*Supervisor, Dr. Diwakar Krishnamurthy,
Department of Electrical and Computer Engineering*



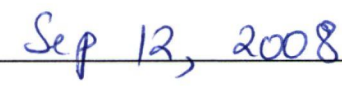
*Dr. Mahmood Moussavi,
Department of Electrical and Computer Engineering*



*Dr. Behrouz Far
Department of Electrical and Computer Engineering*



*Dr. Zongpeng Li,
Department of Computer Science*



Date

Abstract

Analytic performance models such as Queuing Network Models (QNMs) and Layered Queuing Models (LQMs) are important tools that support system sizing, capacity planning and systems management exercises. This thesis describes a new technique called the Weighted Average Method (WAM) designed to improve the accuracy of such models for systems that experience bursty arrivals of customer requests. WAM reflects the impact of burstiness by considering the customer population distribution at a system. It is a constructive technique that permits studies of how arbitrary distributions for workload parameters influence the population distribution and hence performance. Specifically, WAM uses a fast Monte Carlo simulation to estimate the population distribution for any given workload. The estimated distribution is combined with an analytic performance model (e.g., a QNM or a LQM) to predict a system's performance under that workload. The effectiveness of WAM is evaluated through case studies involving two different TPC-W installations.

Acknowledgements

There are so many people who have played very import parts in the completion of this thesis.

I would like to thank my supervisor Dr. Diwakar Krishnamurthy for being a patient supervisor and for supporting this work with ideas, suggestions as well as criticisms.

I am extremely grateful to Dr. Jerry Rolia, who was my internship mentor in HP Labs. His patience and kindness, as well as his academic experience have been invaluable to me.

My parents, Shan'an and Xianxiu, and my brother Gang, have been a constantly unconditional source of support for me, both financially and emotionally throughout my degree. This thesis would certainly not have existed without them.

My husband Zhan has been, always, my joy and my guiding light, and I thank him.

Table of Contents

Approval Page.....	ii
Abstract.....	iii
Acknowledgements.....	iv
Table of Contents.....	v
List of Tables	vii
List of Figures and Illustrations	viii
List of Symbols, Abbreviations and Nomenclature.....	x
 CHAPTER 1: INTRODUCTION	 1
1.1. Motivation.....	1
1.2. Research Objectives.....	8
1.3. Research Contributions.....	8
1.4. Publications.....	11
1.5. Thesis Organization	12
 CHAPTER 2: BACKGROUND AND RELATED WORK	 13
2.1. Performance Testing Methodologies	13
2.1.1. Trace Generation Techniques	14
2.1.2. Request Generation Techniques.....	16
2.1.3. Commercial Tools and Standard Benchmarks.....	18
2.2. Analytic Performance Models	19
2.2.1. System-Level Performance Models.....	19
2.2.2. Component-Level Performance Models	21
2.2.3. Hierarchical Performance Models	24
2.2.4. Existing Techniques for Modeling Systems with Bursty Request Arrivals	27
2.3. Summary.....	32
 CHAPTER 3: THE WEIGHTED AVERAGE METHOD (WAM).....	 34
3.1. Overview of WAM	34
3.2. WAM for Systems with Varying Number of Concurrent Sessions	36
3.3. WAM for Systems with Constant Number of Concurrent Sessions.....	43
3.4. Conclusions.....	45
 CHAPTER 4: C-TPC-W CASE STUDY	 46
4.1. Experiment Test Bed.....	47
4.1.1. Experiment Setup.....	47
4.1.2. Experiment Design.....	49
4.1.3. Experiment Methodology	54
4.2. Overview of Experimental Results	57
4.3. Predictive Performance Models for C-TPC-W	65
4.4. Results of QNMs and LQMs with WAM.....	69
4.5. Conclusions.....	83

CHAPTER 5: H-TPC-W CASE STUDY	86
5.1. Experiment Test Bed.....	87
5.2. Overview of Experimental Results	91
5.3. Modeling Approaches Evaluated for H-TPC-W	93
5.4. Predictive Performance Models for H-TPC-W	96
5.5. Evaluation of Modeling Approaches	101
5.5.1. MEAN.....	102
5.5.2. MBD.....	103
5.5.3. WAMEMP	106
5.5.4. WAMMC	107
5.6. Comparison of WAM for C-TPC-W and H-TPC-W	116
5.7. Conclusions.....	118
CHAPTER 6: SUMMARY AND CONCLUSIONS	120
REFERENCES	125

List of Tables

Table 4.1: Statistics of the session length and the think time	51
Table 4.2: Workload mixes and no-load response time of request types	53
Table 4.3: Response time and resource demand measurements from the case study.....	56
Table 4.4: Estimates of Hurst parameter for BPSLZ replications.....	59
Table 4.5: Accuracy for predicting R_{mean} for overall cases	72
Table 4.6: Accuracy for predicting R_{mean} for bursty cases	75
Table 4.7: Accuracy for BPSLZ-HiMix- workloads	76
Table 4.8: Accuracy for BPSLZ-77- workloads	77
Table 4.9: Accuracy for predicting R_{mean} for non-bursty workloads	78
Table 5.1: Test bed components	89
Table 5.2: Definition of workload mix	91
Table 5.3: Prediction errors for overall approaches.....	102
Table 5.4: Comparison for WAMMC and WAMMC-HiVar	112

List of Figures and Illustrations

Figure 1.1: Typical architecture of an enterprise application system	2
Figure 2.1: Example of a STD for a system-level model	21
Figure 2.2: Hierarchical birth-death model for a session-based system	25
Figure 3.1: Performance modeling process with WAM	35
Figure 3.2: The WAM algorithm	38
Figure 3.3 Algorithm of the <i>GetSyntheticResponse</i> method.....	40
Figure 4.1 Experiment setup for the C-TPC-W system.....	48
Figure 4.2: CDFs for BPSLZ-HiMix-77-Base and EMPSLZ-HiMix-77-Base	61
Figure 4.3: CDFs for EMPSLZ-HiMix-77-Base and EMPSLZ-MedMix-77-Base	62
Figure 4.4: CDFs for BPSLZ-HiMix-71-BigDB workload	64
Figure 4.5: Measured R_{mean} values for BPSLZ-HiMix-71-BigDB	64
Figure 4.6: Queuing network model for C-TPC-W system	65
Figure 4.7: Layered queuing model for C-TPC-W system	66
Figure 4.8: Population distributions for BPSLZ-HiMix-BigDB-71 (Replication 1)	79
Figure 4.9: Population distributions for BPSLZ-HiMix-BigDB-71 (Replication 2)	80
Figure 4.10: Population distributions for BPSLZ-HiMix-BigDB-71 (Replication 3)	80
Figure 4.11: Population distributions for BPSLZ-HiMix-BigDB-71 (Replication 4)	81
Figure 4.12: Response times for BPSLZ-HiMix-BigDB-71	81
Figure 4.13: Response times for BPSLZ-HiMix-77 and EMPSLZ-HiMix-77	83
Figure 5.1: H-TPC-W system	89
Figure 5.2: CPU Utilization of (a) front server and (b) database server.....	92
Figure 5.3 : Measured (a) throughput and (b) response times for three mixes.....	93
Figure 5.4: STD of hierarchical modeling approach for H-TPC-W system.....	95

Figure 5.5: QNM for H-TPC-W system	98
Figure 5.6: LQM for H-TPC-W system	99
Figure 5.7(a): Population distribution from MBD-LQM for Browsing-100.....	104
Figure 5.7(b): Population distribution from MBD-LQM for Browsing-200	105
Figure 5.7(c): Population distribution from MBD-LQM for Browsing-300.....	105
Figure 5.7(d): Population distribution from MBD-LQM for Browsing-400	106
Figure 5.8(a): Population distribution from WAMMC-LQM for Browsing-100.....	108
Figure 5.8(b): Population distribution from WAMMC-LQM for Browsing-200.....	108
Figure 5.8(c): Population distribution from WAMMC-LQM for Browsing-300.....	109
Figure 5.8(d): Population distribution from WAMMC-LQM for Browsing-400.....	109
Figure 5.9(a): Population distribution from WAMMC-HiVar-LQM for Browsing-100.	114
Figure 5.9(b): Population distribution from WAMMC-HiVar-LQM for Browsing-200	114
Figure 5.9(c): Population distribution from WAMMC-HiVar-LQM for Browsing-300.	115
Figure 5.9(d): Population distribution from WAMMC-HiVar-LQM for Browsing-400	115

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
QNM	Queuing Network Model
FCFS	First Come First Served
PS	Processor Sharing
MVA	Mean Value Analysis
LQM	Layered Queuing Model
TPC-W	TPC benchmark W (Web commerce)
C-TPC-W	TPC-W system in Carleton university
H-TPC-W	TPC-W system in HP-labs
WAM	Weighted Average Method
SAS	Software As a Service
SURGE	Scalable URL Reference Generator
SWAT	Session-based Web Application Tester
STD	State Transition Diagram
MOL	Method of Layers
MAP	Markovian Arrival Process
PDE	Population Distribution Estimator
BPSLZ	Bounded Pareto distribution used in Session Length and think time distributions
EMPSLZ	Empirical distribution used in Session Length and think time distributions
ABS_Error	Mean Absolute Error
Max_Error	Maximum of the Absolute Errors
Trend_Error	Difference between the largest error and the smallest error
MBD	Markov-chain Birth Death method
WAMEMP	WAM method with Empirical distribution
WAMMC	WAM method with Monte Carlo simulation
CDF	Cumulative Distribution Function
PDF	Probability Distribution Function
EB	Emulated Browser

CHAPTER 1: INTRODUCTION

1.1. Motivation

Enterprise application systems are often used to perform business functions such as online shopping, accounting, production scheduling, and customer relationship management. Most of the enterprise applications have a multi-tiered architecture [1]. As shown in Figure 1.1, a Web server receives requests from customers who can potentially be spread across the Internet. The Web server forwards a customer request to an application server. The application server typically implements the business logic of the enterprise application. It can in turn contact a database server to request data such as order details and customer account information. The application server dynamically generates a HTML page using its business logic and the data obtained from the database server. It transmits the dynamically generated page to the Web server, which in turn sends it to the customer who initiated the request.

To improve performance and scalability the Web, application, and database servers are typically deployed on separate physical machines. Furthermore, the servers support multiple threads or processes to serve many customer requests concurrently. Typically, more physical machines are added to each tier to handle expected increases in workload. For example, the application server tier may have ten physical machines each running an instance of the application server software. This technique is referred to as *horizontal scaling*.

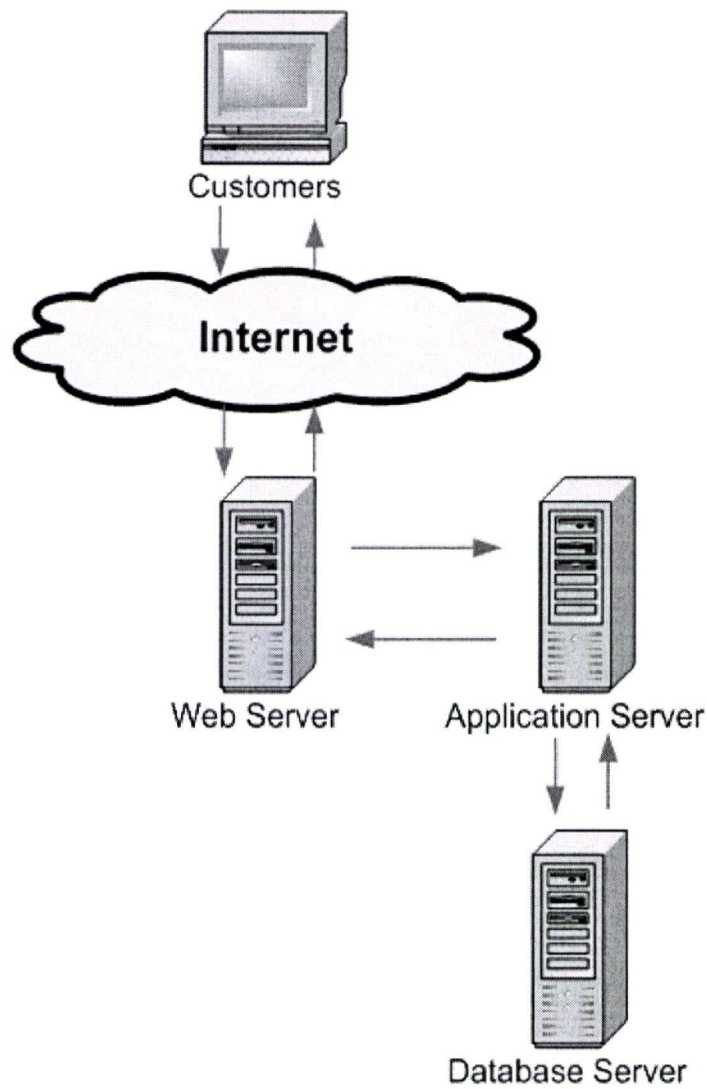


Figure 1.1: Typical architecture of an enterprise application system

Enterprise application systems are typically session-based. The workloads of these systems are often characterized in terms of sessions. A *session* is defined as a sequence of interdependent requests submitted to the system by a single entity. For example, the following is a valid session for an e-commerce system: [Home, View, Add, Purchase]. This session describes a shopper who visits the home page of this system,

views the content of the product she is interested in, adds the product into the shopping cart and finally purchases that product. Besides, the session-based workloads are usually bursty for the enterprise environments. Burstiness can be used to describe the session arrival patterns or request arrival patterns in the systems.

Since performance is crucial for enterprise application systems, it is necessary to support effective techniques for service level assessment, system sizing, and capacity planning. *Service level assessment* studies investigate whether a customer is likely to obtain adequate performance from a system. For example, an e-commerce system provider may want to design her systems such that the system is able to service 10,000 customer requests per second while at the same time ensuring that a customer's request does not experience a delay of more than 500 milliseconds. Service level assessment is important since poor performance can mean lost customers and hence lost revenue. *Sizing* is the process of determining how many software and hardware resources to provision to the system so that service level requirements can be met. For example, a system architect may want to determine the number of physical nodes to provision per tier as well as the number of server threads or process needed on each physical machine in a tier to handle the system's expected workload. *Capacity planning* deals with how to upgrade a system to deal with future increases in customer workload.

Performance testing is often used to support service level assessment, system sizing, and capacity planning. It involves submitting a synthetic workload to a system under study in controlled conditions. The *synthetic workload* emulates the real workload of the system under study (i.e., requests from real customers of the system). A synthetic workload is

constructed from a workload model. The *workload model* consists of workload attributes that are likely to impact performance the most and statistical characterizations for those attributes. Measurements such as customer response times and system resource utilizations (e.g., CPU utilization) are collected during tests to derive insights on system performance.

Performance tests must satisfy two main requirements namely, representativeness and flexibility. A synthetic workload is said to be *representative* of a real workload if both workloads yield similar performance when submitted to the system under study. Including attributes that impact performance in the workload model and specifying appropriate characterizations for the attributes influences representativeness. Very often it is difficult to choose characterizations for workload attributes since there may be little or no data available about characteristics of real workloads. As a result, it is important for a performance testing methodology to support the *flexibility* to conduct controlled sensitivity analyses on the workload attribute characterizations. This allows a tester to establish the performance behaviour of a system under a range of workloads that the system might encounter in practice.

Performance analysts often rely on analytic performance modeling in addition to performance testing. Analytic modeling supports “what-if” analyses. Typically performance testing is time consuming and hence expensive. As a result only a finite number of workloads are considered during performance tests. In order to evaluate system performance under other workloads, a model can be constructed and parameters of the models can be derived from existing performance tests. The model can be

perturbed to understand system behaviour under workloads that were not explored during performance testing. Furthermore, the model can also help understand performance behaviour under alternate hardware (e.g., faster CPU) and software configurations (e.g., more application server threads). Analytic modeling is also the only approach for cases where a system is under development and not available for testing.

Queuing Network Models (QNM)s [2][3] are widely used by practitioners as an analytic modeling methodology. A QNM consists of a system of queues where each queue is used to abstract a system resource (e.g., a CPU or disk). The parameters of the model include the number of concurrent customers in the system, the scheduling disciplines used in the queues (e.g., First Come First Served, Processor Sharing) and the average time taken to service a customer request at each resource once the request has acquired the resource. A QNM can be solved efficiently by using a technique known as Mean Value Analysis (MVA) [2]. MVA sets up and evaluates analytic expressions for the queues in a model to provide mean estimates for performance metrics such as customer response time, number of customer completions per second, and resource utilizations. MVA achieves computational efficiency by stipulating various assumptions together referred to as the product form assumptions [3]. The estimates provided by MVA are accurate if these assumptions are met in the system under study. Due to its efficiency MVA is widely used for solving QNM)s.

The straightforward application of MVA and QNM)s to study session-based systems poses the following challenges:

- Several recent studies [4][5] have indicated that the workloads of real session-based systems exhibit burstiness in the arrival of requests to the system. Since burstiness can significantly impact performance, methods must be supported to reflect its impact within models. Krishnamurthy *et al.* [6] observed that bursty workloads in session-based systems exhibited bursts in the number of concurrent customer sessions using the system. This suggests that it may be important to take into account the distribution of number of concurrent sessions during performance modeling exercises. However, traditionally a QNM only accepts a mean number of concurrent customers as an input. Furthermore, bursty request arrivals violate product form assumptions thereby limiting the straightforward application of MVA.
- Krishnamurthy *et al.* [6] showed that several workload attributes as well as the distribution of system response times influence the distribution of number of concurrent sessions. While studying systems with burstiness, performance analysts may require support for fine-grained characterization that allows them to understand how arbitrary distributions for these workload attributes impacts the distribution of number of concurrent sessions and hence performance. Currently no MVA-based technique supports the ability to construct the distribution of number of concurrent sessions for arbitrary distributions of workload attributes.
- QNMs can typically be used only for estimating the impact of contention among requests from various customers for hardware resources. They are not designed for evaluating the impact of contention for software resources such as processes and threads. Furthermore, they cannot be used to study the impact of various software request reply relationships (e.g., asynchronous processing). These factors may be of

considerable importance for enterprise application systems [7]. Consequently, QNMs with extensions to handle software related issues need to be evaluated for these systems.

Even though several techniques that do not use MVA exist for modeling burstiness, there are significant challenges in applying them in practice. Existing analytic techniques that address burstiness are more complex than MVA. Typically, exact solution methods for mean response times do not exist and reliable estimates from approximate solutions are difficult to obtain. There are also several techniques to evaluate the impact of burstiness that depend solely on discrete event simulation. The exclusive reliance on simulation makes these techniques slower when compared to MVA-based analytic techniques especially for large systems.

This thesis proposes a new performance evaluation technique to study session-based systems characterized by bursty request arrivals. The technique applies the well-known concept of hybrid modeling which typically involves combining simulation with analytic techniques to analyze complex systems. It relies on a fast Monte Carlo simulation. The simulation can take as input arbitrary distributions for a set of workload attributes that influence burstiness. It estimates the population distribution that results from these distributions. The population distribution provides a measure of the burstiness of the system. For a system where the number of concurrent sessions varies the *population distribution* is defined as the distribution of number of concurrent sessions at the system. For a system where the number of concurrent sessions is constant the *population distribution* is defined as the distribution of number of concurrent requests at the system.

The population distribution estimated through the Monte Carlo simulation is combined with MVA-based predictive models to offer mean estimates of various performance metrics of interest.

1.2. Research Objectives

The main objectives of this research are as follows:

- Evaluate the ability of straightforward applications of QNMs solved using MVA to capture the performance impact of burstiness in request arrivals.
- Compare QNMs with extended QNMs called Layered Queuing Models (LQMs). LQMs[8] can capture the impact of contention for software resources as well as model various software request-reply relationships.
- Propose a new method to model session-based systems characterized by burstiness. For efficiency and adoption by practitioners, the method should be able to exploit MVA-based predictive models. Furthermore, it should support fine-grained characterization of how various workload attribute distributions impact burstiness and hence performance.
- Use performance test results collected from session-based systems to validate the proposed technique.

1.3. Research Contributions

Data collected from performance tests on two different TPC-W [9] systems is used in this thesis. The central component of a TPC-W system is a multi-tier bookstore application. It is very widely used in academia and practice as a sample enterprise application. The

C-TPC-W system was installed in Carleton University, Ottawa, Ontario, Canada. In this system the number of concurrent sessions was allowed to change during an experiment. Hence its population distribution is characterized by the distribution of number of concurrent sessions. Furthermore this system did not serve images embedded in the HTML pages of the bookstore's pages. The H-TPC-W system was installed in Hewlett-Packard (HP) Labs, Palo Alto, California, USA. Since this system used a constant number of concurrent sessions in each experiment the population distribution is characterized by the distribution of number of concurrent requests. In contrast to C-TPC-W, H-TPC-W served the images used in the HTML pages. Based on data collected from these systems the salient findings are as follows:

- The accuracy of performance predictions from the straightforward application of a QNM solved using MVA was very poor especially for bursty workloads. For the bursty workloads considered in C-TPC-W, this approach yielded an average error of 19.34% while predicting the mean response time. The maximum error was as high as 42.56%. For the H-TPC-W system, the average error for mean response time predictions is 68.93%.
- A straightforward application of LQMs solved using MVA improved prediction accuracy but the errors were still very high. For the C-TPC-W system, the MVA-LQM approach reduced the average mean response time prediction error by 0.24% for bursty workloads when compared to the QNM. The maximum error was as high as 32.37%, about 10% improvement with respect to QNMs. For the H-TPC-W

system, the average error with the LQM was 50.54% as compared to 68.93% for the QNM.

- The use of population distributions significantly improved response time predictions for bursty workloads. Population distributions that were observed during the performance tests were used in conjunction with QNMs and LQMs developed for the systems. For the C-TPC-W system average prediction accuracy was 4.87% for bursty workloads when a LQM was used in combination with the measured population distributions. This represents an improvement of 14% over the straightforward application of the LQM. For the H-TPC-W system a LQM used with the measured population distributions yielded an average prediction accuracy of 7.22%. This represented an improvement of 43% when compared to the straightforward application of the LQM.
- The accuracy of estimating the population distribution using Monte Carlo simulations was very good for the C-TPC-W system. For the C-TPC-W system, the estimated distributions of number of concurrent sessions are very close to the corresponding distributions measured during performance tests. As a result the performance predictions made using the estimated distributions were close to those made using the measured distributions. This suggests that accurately estimating the population distribution given distributions for workload attributes that impact burstiness is feasible for this type of a system.
- The accuracy of estimating the population distribution was very poor for the H-TPC-W system where the number of concurrent sessions in each performance tests

was constant. Results show that while it is feasible to accurately estimate the distribution of number of concurrent sessions it is very difficult to estimate the distribution of number of concurrent requests. Consequently, the thesis concludes that supporting fine-grained characterizations of the impact of workload characteristics on burstiness is very difficult for systems such as H-TPC-W which forces burstiness to be modeled through the distribution of number of concurrent requests. These results also suggest that the use of a more realistic performance testing approach that lets the number of concurrent sessions vary during a test (e.g., the approach adopted in C-TPC-W) can make performance prediction easier for session-based systems characterized by burstiness.

The performance testing measurements for the two TPC-W systems were collected in previous work [6] [10]. The objectives of those studies are unrelated to the objectives of this thesis. The performance measurements collected by researchers of these two studies are reused in this thesis and to validate the proposed modeling technique. Furthermore, the thesis uses the Method of Layers (MOL) tool developed by Rolia and Sevcik [11] to solve QNM and LQM models.

1.4. Publications

Parts of this research have contributed to two papers that have been published as HP Labs technical reports. The following two journal papers that derive from these technical reports are under submission.

- 1) D. Krishnamurthy, J. Rolia, M. Xu, “WAM – The Weighted Average Method for Predicting the Performance of Systems with Bursts of Customer Sessions,” in submission.
- 2) J. Rolia, D. Krishnamurthy, M. Xu, and S. Graupner. “APE: An Automated Performance Engineering Process for Software as a Service Environment,” in submission.

Publication 1 presents the early version of the algorithm for the constructive technique to estimate the population distributions. The results presented in this case study only include the C-TPC-W system. In this thesis, the new results of the improved algorithm for estimating the population distribution are reported for both C-TPC-W and H-TPC-W. Furthermore, the measured data for the two TPC-W systems and WAM have been used in publication 2 to support automated performance evaluation of Software As a Service (SAS) environments.

1.5. Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 describes background and related work. Chapter 3 introduces the new Weighted Average Method (WAM) technique to model the impact of bursts in request arrivals. Chapter 4 and Chapter 5 present the case studies for the two TPC-W systems to validate WAM. Summary and conclusions are offered in Chapter 6.

CHAPTER 2: BACKGROUND AND RELATED WORK

This chapter reviews existing performance testing and performance modeling methodologies. In particular existing modeling techniques that address the issue of burstiness are discussed. This discussion is used to motivate the modeling technique proposed in this thesis.

Section 2.1 reviews existing performance testing methodologies and discusses the techniques used in this thesis. Section 2.2 provides background on the common analytic modeling techniques used by practitioners. Section 2.3 reviews existing work that has focused on modeling systems characterized by burstiness in request arrivals.

2.1. Performance Testing Methodologies

As mentioned in Chapter 1, performance testing involves submitting synthetic workloads to a system under study. Generating synthetic workloads for Web-based systems typically involves two steps namely, trace generation and request generation. The *trace generation* step handles the complexities of creating a synthetic trace of HTTP/HTTPS requests that adhere to a workload model. The *request generation* step submits the requests in the trace to the system under study. Pre-generating traces reduces overheads during request generation, thereby ensuring that the achieved workload characteristics stay close to the specified characteristics. The following sections describe these two aspects of performance testing.

2.1.1. Trace Generation Techniques

Many tools exist for testing Web server systems that serve static HTML pages. SURGE [12] is a synthetic workload generator for testing Web servers. It has an offline trace generation engine that is used to create a trace of HTTP requests for a set of files hosted on the Web server under test. The tool allows a performance analyst to control the distributions for file size, HTTP response size, file popularity, temporal locality, number of embedded references per Web page, and idle time between successive requests.

In contrast to systems that serve only static HTML files, the trace generation step for session-based systems must address the issue of handling inter-request dependencies. *Inter-request dependencies* arise because some requests in a session have to be submitted only after certain others requests have already been submitted. For example, in an e-commerce system a request to order an item can only be submitted after a request to add the item to the shopping cart has been submitted. The common approach to handling inter-request dependencies within sessions has been the use of a first-order Markov chain [9][13][14]. With this approach, the states of the Markov chain represent a system's request types. Typically, a request type instructs a session-based application to execute a particular script or module. For example, in an e-commerce system the "Home" request type may instruct the system to display the homepage and the "Shopping cart" request type may execute a script to add items to the shopping cart. Transitions between the states model the user behaviour of navigating from one request type to another within a session. Transition probabilities determine the number of "visits" to each state in the chain and, hence, the mix of request types. Krishnamurthy *et al.* [6] argue that such an

approach has limitations especially for systems where the first-order dependency assumption is not valid (i.e., next request to be submitted in a session depends on more than just the current request submitted).

SWAT [6] is a trace generation engine for session-based systems. It does not rely on Markov chains to enforce inter-request dependencies. Instead, it uses semantically correct request sequences for a system under test. SWAT ensures that sessions present in the synthetic workload conform to these sequences so that they are valid for the system under test. Specifically, SWAT constructs a synthetic workload with correct inter-request dependencies that has specified characteristics for a set of workload attributes that are likely to impact performance. The characterizations for these attributes can be either based on those observed in real systems or perturbations for the purpose of a sensitivity analysis.

The workload attributes modeled by SWAT are as follows. With SWAT, sessions arrive into the system from the outside world. The distribution of the times between successive session arrivals, defined as the *session inter-arrival time*, can be specified by the performance analyst. Each session behaves as a user by alternating between submitting a request and waiting for a response. The time a session spends idle before issuing its next request is defined as the *think time*. SWAT allows the think time distribution to be specified. The number of requests in a session is defined as the *session length*. SWAT also allows the distribution of session lengths to be specified. In addition to these attributes, SWAT permits control over the workload mix. *Workload mix* specifies the relative frequencies of occurrences for the different request types in the system. SWAT

outputs a trace file containing a user specifiable number of sessions such that the trace conforms to the distributions and workload mix.

2.1.2. Request Generation Techniques

There are three different types of request generation schemes that are typically supported by existing request generators namely, user-equivalents, aggregate, and mixed. These methods are briefly described as follows.

The user-equivalents method (also known as closed method) is very widely used by performance analysts. The SURGE workload generator described previously uses this method. The closed method is based on the ON/OFF approach. A performance test uses a fixed number of software processes or threads called *user-equivalents*. Each user-equivalent emulates a user's Web browsing behavior. It alternates between submitting requests specified in a trace (ON state) and lying idle for a predetermined period of time (OFF state). The load on the system under study can be increased by increasing the number of user-equivalents. The primary limitation of this approach is that it does not permit control over characteristics of the aggregate request arrival process observed at the system. For example, it is not possible to specify the exact instants at which requests have to arrive at the system under study. This is because the time instants at which a user equivalent issues requests depends on system state; long delays for requests resulting from a heavily loaded system will increase the time between successive requests from a user equivalent. Furthermore, due to the dependency with system state, when the system is heavily loaded the rate at which requests arrive at the system tends to increase less significantly as the number of user-equivalents is increased. As a result, a large number

of user-equivalents may be needed to cause significant stress (e.g., overload) on the system under study.

Request generation engines that employ aggregate workload generation (also known as the open approach) address these limitations. S-Clients [15] and httpperf [16] are request generators that support this method. With this scheme, there is no notion of a user equivalent. Instead, requests are issued at the appropriate time instants, independent of system load, so that a specified request arrival process is achieved at the system. Furthermore, since the request generator does not wait for the response of a particular request before submitting its next request, aggregate workload generation allows systems to be stressed while making use of fewer resources when compared to request generators relying on the user-equivalents approach. While the aggregate approach provides performance measures for aggregate system behavior, it does not provide corresponding measures for individual user behavior.

Mixed workload generation combines aspects of the user-equivalents and aggregate approaches. This approach is particularly appropriate for describing session-based workloads and is the approach used by SWAT. With this approach, user sessions arrive in an open manner. Sessions are initiated at the specified time intervals irrespective of the number of sessions already present in the system under study. However, each individual session behaves in a manner similar to a user-equivalent. In contrast to the user-equivalents approach, the number of concurrent sessions in the system can vary continuously during the course of a performance test. The httpperf [16] request generator

supports a mixed request generation module. SWAT uses this module to submit its synthetic trace of sessions

2.1.3. Commercial Tools and Standard Benchmarks

There are also several commercial tools [17][18] for performance testing session-based systems. However, to the best of my knowledge, these tools only address request generation issues and do not provide any automated trace generation capabilities. Typically, performance test teams have to manually code session emulation scripts that take into account inter-request dependencies for the system being studied. Such an approach can be time-consuming. Also, script development can get increasingly complex when finer control is needed over characteristics such as workload mix and session length distribution.

The industry-standard TPC-W [9] benchmark is widely used in academic and practice for investigating performance issues pertaining to multi-tier, session-based application systems. The benchmark was proposed to facilitate standardized comparisons of hardware and software platforms from different vendors that could be used for executing e-commerce applications. The benchmark suite contains specifications for developing a multi-tiered bookstore application. The suite also contains a workload generator for conducting performance tests on the bookstore application. The workload generator follows the user-equivalents approach. Several open-source implementations of the TPC-W bookstore and the workload generator have been developed [19][20]. These implementations have been widely used to emulate the behaviors of real multi-tier application systems in performance studies [10][21].

This thesis considers two different TPC-W systems to demonstrate the proposed modeling technique. The C-TPC-W system uses the TPC-W bookstore implemented using the commercial off-the-shelf software. This system was tested using SWAT instead of the default user-equivalents based TPC-W workload generator. The H-TPC-W system uses the University of Wisconsin open-source TPC-W implementation [19]. This system was tested using the default, user-equivalents-based workload generator. This thesis discusses the implications of these two different workload generation approaches on performance modeling.

2.2. Analytic Performance Models

As mentioned in Chapter 1, an analytic performance model solves a system of equations to estimate performance metrics for a system. In this section various analytic modeling techniques such as system-level performance models, component-level performance models, and hierarchical models [22] are discussed. These analytic modeling techniques are very useful for computer systems performance analysis

2.2.1. System-Level Performance Models

A system-level performance model considers the system as a “black box” [22]. The internal details of the box are not modeled explicitly. Instead, the technique relies on only the throughput function of the box. *Throughput* is defined as the number of completions per second by the system. The throughput function gives the average throughput of the system as a function of k the concurrency of the system. The meaning of k depends on the manner in which the system is modeled. For example, k can represent the average number of concurrent customer sessions in the system and $X(k)$ is

then the number of sessions completed per second when there is k concurrent sessions in the system. Alternately, k can represent the number of concurrent customer requests in the system and $X(k)$ is then the average number of customer requests completed per second when k concurrent customer requests in the system.

A system-level performance model can be solved by using the state transition approach. The definition of state depends on problem at hand. Typical examples of state are the number of concurrent sessions in the system and the number of concurrent request in the system. As shown in Figure 2.1, the state transition diagram (STD) illustrates the states that a system can be found in as well as how it transitions from state to state. An arrival transition increases system state by 1 while a departure transition decreases system state by 1. As shown in Figure 2.1, the transitions are characterized by state dependent on arrival and departure rates. A set of equations called flow equilibrium equations can be set up for the STD. These equations can be solved to estimate the state probabilities $p(k)$. $p(k)$ is defined as the fraction of time spent by the system at state k . The state probabilities can then be used to obtain performance metrics such as mean system utilization, mean response time, and mean system throughput.

It should be noted that the technique makes several assumptions. In particular, the times spent at the various states must be exponentially distributed. This is often referred to as the Markovian assumption and for this reason these models are also called as Markov models. Performance predictions from this technique can be inaccurate if these assumptions are not satisfied. Several variants of the STD shown in Figure 2.1 are also possible. For example, models that place an upper limit on the state can be used to

represent systems which impose limits on the maximum number of concurrent sessions or requests allowed. Similarly, state independent arrival and departure rates can also be used.

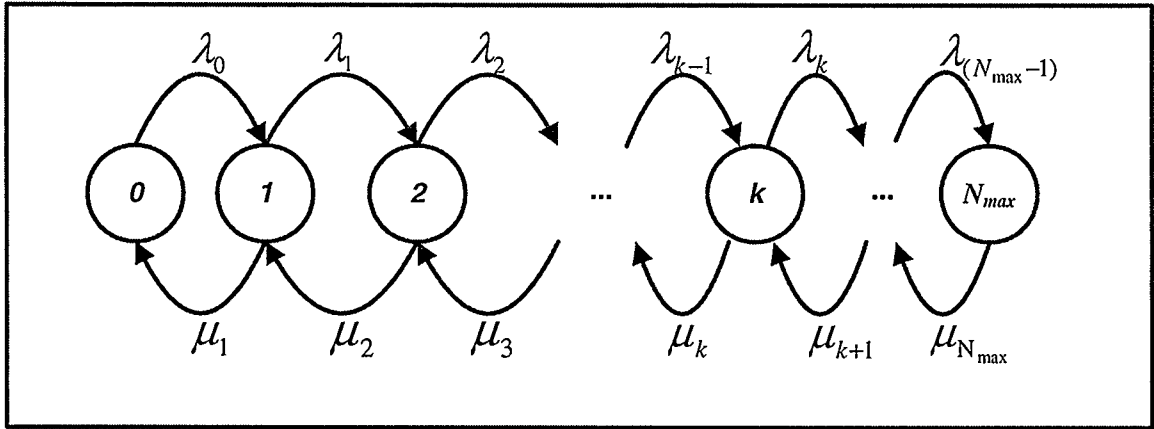


Figure 2.1: Example of a STD for a system-level model

2.2.2. Component-Level Performance Models

In contrast to system level models, component-level performance models take into account the different resources of the system and the way they are used by different requests. QNMs are widely used as component-level models. A QNM consists of a system of inter-connected queues. Each queue is used to model resources such as processors, disks, and networks. As mentioned in Chapter 1, the MVA technique can be used to efficiently solve the system of queues or obtain mean values of performance estimates such as response time, resource utilizations, and throughput.

A QNM takes as input several parameters. A resource service demand D has to be specified for each resource in the QNM. D is defined as the mean time consumed by a resource to satisfy a customer's requests. Note that D does not include the time spent by

the customer request waiting to get access to the resource. It only represents the time spent in the resource after the request obtains access to the resource. D can typically be estimated from measurements collected during performance tests [23]. In addition to the demand a queuing discipline has to be specified for each queue. Examples of queuing discipline include First Come First Served (FCFS) and Processor Sharing (PS) [22].

QNM support the concept of multiple customer classes. Multi-class models are used for systems, which have different groups of customers where each group stresses system resources in a significantly different manner than the other groups. For multi-class QNM the demand values for a resource must be specified separately for each customer class using that resource.

QNM can have open customer classes as well as closed customer classes. The number of requests present in the queuing network from an *open customer class* can be unbounded. In contrast, a *closed customer class* has a fixed number of requests in the network. A QNM that only has open customer classes is called as an *open QNM* while a QNM that only has closed customer classes is defined as a *closed QNM*. In an open QNM requests arrive into the network as per an arrival rate λ . The arrival rate is an input parameter for the model. The request consumes various resources and departs the system. For a stable system, the system throughput equals the arrival rate. In a closed QNM a fixed number of customers “circulate” in the system. The number of concurrent customers is an input parameter for the model. A customer submits a request to the system which consumes various resources. However, instead of departing the system after consuming the resources the request is replaced by another request from the same

customer. The average time between successive requests from a customer can be specified as a mean think time. The system throughput depends on the number of concurrent customers, the system response time and the think time.

While a QNM can reflect contention among requests from various customers for hardware resources, it cannot be directly used to reflect the impact of contention for software resources. Examples of software resources include software processes, threads, semaphores, and locks. Furthermore it is also difficult to model with QNMs request-reply relationships such as fork/join, asynchronous processing, and processing involving both synchronous and asynchronous phases. These interactions are commonly encountered in software systems. LQMs are extended QNMs proposed to address these limitations. A LQM consists of system of QNMs that allow a software resource to provide some service to customers while also acting as a customer for other resources (e.g., a hardware resource). Rolia and Sevcik [11] devised an algorithm called the Method of Layers (MOL) that solves the system of QNMs iteratively using MVA to obtain mean performance estimates. Balsamo *et al.* conclude that extended QNM-based approaches, such as LQMs, are the most appropriate modeling abstraction for multi-tiered software systems [7].

The MVA technique yields accurate predictions only when certain assumptions are met. As mentioned in Chapter 1, the set of assumptions is referred to as product form assumptions [3]. In particular MVA's performance estimates can be inaccurate when a system experiences burstiness in the arrival of requests. Other conditions that MVA cannot handle include priority scheduling, highly variable resource demands at FCFS

resources, and blocking relationship among resources. This thesis proposes a new technique to model systems with bursty requests arrivals. In contrast to a closed QNM which considers only the mean customer population, this technique captures the impact of burstiness by considering the customer population distribution.

2.2.3. Hierarchical Performance Models

Hierarchical performance models combine system and component-level models. The hierarchical technique has been used to study the performance of mixed systems that are characterized by both open customer arrivals and closed customer circulations [24]. They have also been used to study the load dependent behaviour of multi-threaded software servers [25].

Figure 2.2 presents the STD for a hierarchical model that can be used for a session-based system. This model is used as a baseline for comparison with WAM. In general, a hierarchical model has a higher level model that uses state dependent arrival and departure rates to calculate the state probabilities and hence performance metrics such as response time. The departure rates are obtained by solving the lower level model which is a closed QNM or LQM whose predictions are obtained through MVA.

In Figure 2.2, the state of the system represents the number of concurrent sessions in the system. The state S_k varies from 0 to N where N is the maximum number of concurrent sessions in the system. Sessions arrive at the system from the outside world. Each session arrival causes the number of concurrent sessions to increase by 1. The rates at which such transitions occur are given by the state dependent session arrival rates λ_{sk} . A

session submits L requests on an average where L is the mean session length. Z denotes the mean think time between successive requests in a session. Each session completion causes the number of concurrent sessions to decrease by 1. The rates at which such transitions occur are given by the session death rates μ_{sk} .

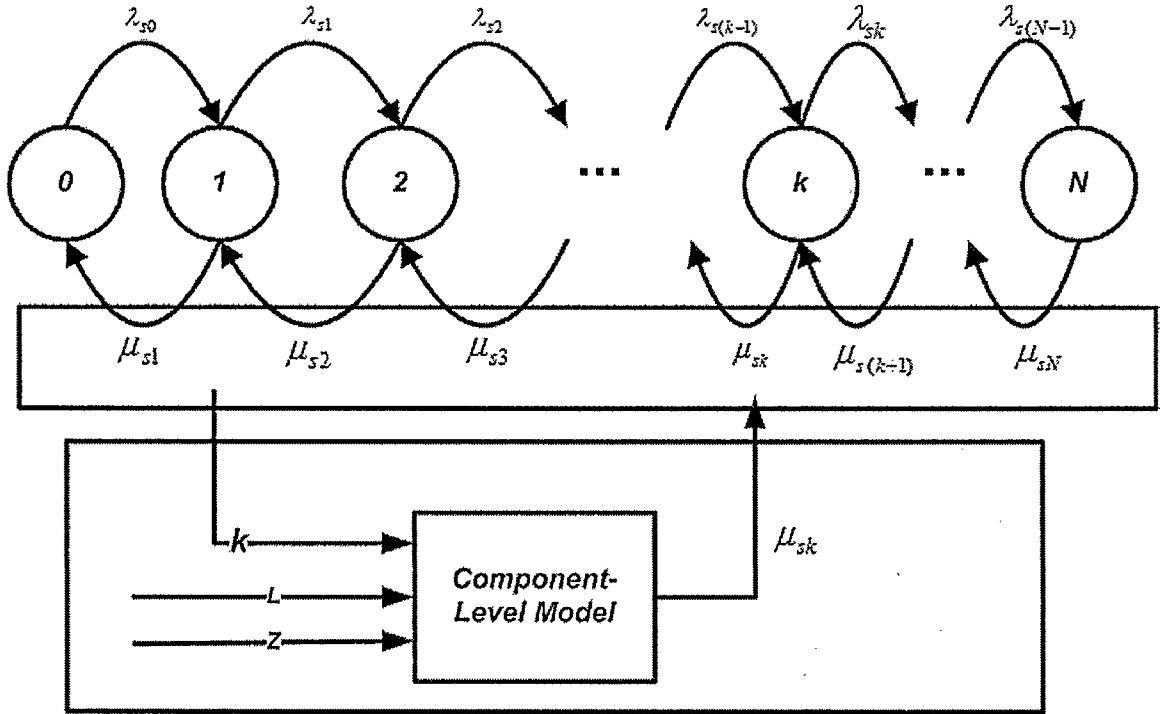


Figure 2.2: Hierarchical birth-death model for a session-based system

At a given state k , k concurrent sessions are competing for the session-based system's resources. Consequently, as shown in (2.1) the death rate μ_{sk} can be calculated as the session throughput X_{sk} obtained by solving a closed QNM with a session population of k and mean think time of Z .

$$\mu_{sk} = X_{sk} = \frac{k}{(R_k + Z)L} \quad (2.1)$$

In (2.1) R_k is the mean request response time obtained from the component-level predictive model when there are k sessions observed in the system. Balance equations involving the birth and death rates can be solved to obtain the probability P_k of residing in each state k as follows:

$$P_0 = \frac{1}{1 + \sum_{k=1}^N \prod_{i=1}^k \frac{\lambda_{s(i-1)}}{X_{si}}} \quad (2.2)$$

$$P_k = P_0 * \prod_{i=1}^k \frac{\lambda_{s(i-1)}}{X_{si}} \quad (2.3)$$

The probabilities P_k , for $k=0 \dots N$ defines the population distribution. Equations (2.2) and (2.3) can be written in terms of customer request arrival and completion rates. The request arrival rate λ_k is the session arrival rate λ_{sk} multiplied by the mean session length L . Similarly, the request throughput X_k is the session throughput X_{sk} multiplied by the mean session length L . Using these relationships (2.2) and (2.3) can be rewritten in terms of request-level rates as follows:

$$P_0 = \frac{1}{1 + \sum_{k=1}^N \prod_{i=1}^k \frac{\lambda_{(i-1)}}{X_i}} \quad (2.4)$$

$$P_k = P_0 * \prod_{i=1}^k \frac{\lambda_{(i-1)}}{X_i} \quad (2.5)$$

The mean request response time for the system is estimated using Little's law [23] as follows:

$$R_{mean} = \frac{\sum_{k=1}^N P_k X_k R_k}{\sum_{k=1}^N P_k X_k} \quad (2.6)$$

where P_k represents the population distributions, X_k represents the customer request throughputs, and R_k represents the mean request response time at population level k .

2.2.4. Existing Techniques for Modeling Systems with Bursty Request Arrivals

This thesis focuses only on burstiness in the arrival of requests to a session-based system. The term bursty is also used in literature to describe workloads characterized by high variability and correlation in resource service demands. For a given customer request arrival pattern and mean request service time, the performance of a resource with bursty service times is likely to be different than that of a resource with less bursty demands. Bondi and Whitt [26] studied the impact of high service time variability on the performance metrics of a closed network of queues and discussed the implications of the results for analytic performance modeling. Eager *et al.* [27] proposed an efficient technique for solving a closed network of queues with high service time variability. More recently, Casale *et al.* [28] proposed an analytic technique to compute upper and lower bounds of performance indices for closed systems with bursty service processes. Addressing this type of burstiness is not the direct focus of this work. However, analytic techniques that can reflect the impact of high variability in service times can be integrated with the WAM technique proposed in this thesis in a straightforward manner to study such systems.

Several studies have indicated that multi-tier session-based systems experience bursty request arrivals and that burstiness can adversely affect performance. Menasce *et al.* [5] characterized the workloads observed at an e-commerce system and an auction system. The authors found that both systems were characterized by bursty arrivals of requests over several timescales. They invoke the properties of the well-known ON-OFF process [29] to argue that the bursts observed at fine timescales, i.e., several dozen seconds, were due to the heavy-tailed [30] nature of the session length distributions observed at the systems and the presence of think times in sessions. Vallamsetty *et al.* also noticed similar burstiness in the arrival of requests at another real e-commerce system [4]. Krishnamurthy *et al.* [6] showed for a multi-tier system that distributions which cause highly variable session lengths, think times, and request resource demands result in high variability in the distribution of number of concurrent sessions and hence burstiness in request arrivals at fine timescales. This suggests that modeling customer population distribution may be helpful for modeling the impact of bursty request arrivals in session-based systems. In addition to such fine timescale burstiness, burstiness has also been observed at coarser timescales, e.g., hours, days, in real-session based systems [31].

Burstiness can have a big impact on how predictable or repeatable a system's behaviour is in response to similar workloads. Crovella and Lipsky showed that the steady-state values for performance measures from multiple statistically identical simulation runs that use heavy-tailed distributions can result in very different measures for each run [32]. Observations obtained using a heavy-tailed distribution exhibits non-negligible probabilities for very large and very small values. Krishnamurthy *et al.* [6] confirmed the

result of Crovella and Lipsky for multi-tier software systems. For a TPC-W system servicing bursty workloads, the authors found that multiple statistically identical measurements run with the same mean resource demands and same throughput resulted in significantly different mean response time measurements. These results suggest the need for modeling approaches that characterize a range of possible system behaviours under bursty workloads.

MVA of closed QNMs and LQMs, techniques widely used by practitioners, only consider the average customer population of a system. More complex techniques exist which could potentially be used for modeling the impact of burstiness. Classical queuing theory offers G/G/* queues [33] that can take into account the first and second moments of any arbitrary request inter-arrival time distribution. However, exact solution methods for mean response times do not exist for networks of such queues and reliable estimates from approximate solutions are difficult to obtain [33]. Furthermore, heavy-tail-like distributions that are a characteristic of bursty systems require more than the first two moments for a proper characterization. Recently, Psounis *et al.* [34] considered a single multi-server queue that is subjected to heavy-tail-like distributions. However, the approach has not been extended to queuing networks.

Several researchers have proposed Markovian Arrival Processes (MAPs) for modeling bursty request arrivals [28][35]. However, queuing networks having MAPs cannot be solved using efficient analytic techniques such as MVA. These techniques typically rely solely on discrete event simulations which are more time consuming than applying MVA.

Menasce and Almeida propose MVA-based techniques that consider heavy-tailed distributions and bursty request arrivals for Web server systems [22]. Specifically, they describe a QNM that reflects the impact of a heavy-tailed file size distribution at Web servers serving static HTML pages. The authors argue that a multi-class model where the classes represent requests for files belonging to different file size ranges is more suited for capturing the impact of the heavy-tailed distribution than a single class model. This technique is specific to systems that serve static files. It is not intended for transaction-oriented, session-based systems of the kind considered in this work. The authors also propose another heuristic technique that uses a QNM to reflect the impact of burstiness in request arrivals. The technique splits a given HTTP request log into equal sized time periods. It counts the number of time periods for which the average request arrival rate exceeded the request arrival rate observed over the entire log. This count is used to compute a burstiness factor which is in turn used to inflate the service demand of the bottleneck device in a QNM [22]. However, the technique was not validated with respect to measurements and was not proposed as a constructive technique that permits a performance analyst to assess the impact of distributions that contribute to burstiness on mean response time behaviour.

As mentioned previously, the hierarchical technique described in Section 2.2.3 can consider the population distribution. As shown in Section 2.2.3, the balance equations can be solved to estimate the population distribution. Equations (2.1) and (2.6) can then be used to compute performance metrics. However, as mentioned in Section 2.2.1, the population distribution estimates are accurate only for workloads that cause the times

spent in the system states to be exponentially distributed. However, this is not expected to be the case for systems affected by burstiness.

This thesis proposes and evaluates a new approach to estimate population distribution called WAM. WAM applies the well-known concept of hybrid modeling to session-based systems. Hybrid modeling typically combines simulation with analytic modeling techniques to study complex systems. The WAM technique is motivated by the hierarchical approach described in Section 2.2.3 but does not rely on a birth-death model to estimate population distribution. Instead for any given workload it exploits a fast Monte Carlo simulation to quickly estimate the population distribution, i.e., per population level probabilities $P(k)$ for $k = 0 \dots N$, rather than relying on the closed formulas given by Equations (2.2) and (2.3), or (2.4) and (2.5) for the birth-death process. The estimated distribution is combined with a performance model (e.g., a QNM or a LQM) to predict a system's performance under that workload. The primary advantage of the newly proposed method is that it is not limited by the exponential distribution assumption in the hierarchical approach. WAM is more robust with respect to the distributions that contribute to bursty behaviour. For example, it permits in a straightforward manner the study of how arbitrary distribution functions for workload parameters such as session inter-arrival time, think time, and session length impact the population distribution and hence performance.

Summarizing, the WAM technique is proposed with an objective to achieve the following advantages over other techniques reviewed in this section that address the issue of burstiness:

- **Robustness with respect to arbitrary distributions** – This allows the impact of arbitrary distributions or workload traces to be studied.
- **Support for constructive capability** – This allows fine-grained control over how distributions of various workload parameters that influence burstiness impact performance. This capability also allows a range of possible behaviors to be characterized for heavy-tailed workloads.
- **Efficient when compared to alternatives that rely solely on simulation** – As described in Chapter 3, the WAM technique can exploit efficient MVA-based predictive models. It also relies on fast, Monte Carlo simulations to estimate the population distribution. Due to its use of MVA and fast simulations, for large systems WAM is likely to be faster than techniques that solely rely on discrete event simulations (e.g., techniques that use MAPs) to evaluate the impact of bursts.

The following chapters describe the design and validation of the WAM technique.

2.3. Summary

This chapter reviewed performance testing and performance modeling methodologies. In particular, modeling techniques that address the issue of burstiness were described. Burstiness can significantly degrade system performance and hence it is important for analytic modeling techniques to capture its impact. A brief overview of the WAM approach for modeling burstiness was provided and its potential advantages were discussed.

Chapter 3 describes the WAM technique in more detail. Chapters 4 and 5 use performance test results collected from two different TPC-W systems to validate the

technique. In particular, WAM is used in conjunction with both QNMs and LQMs for those systems. The WAM-based models are compared with the following analytic modeling approaches:

- Closed QNMs of the systems solved using MVA. These models only consider mean population.
- Closed LQMs of the systems solved using MVA. These models only consider mean population.
- The hierarchical method of Section 2.2.3 applied with closed QNMs of the systems solved using MVA.
- The hierarchical method of Section 2.2.3 applied with closed LQMs of the system solved using MVA.

CHAPTER 3: THE WEIGHTED AVERAGE METHOD (WAM)

This chapter describes the WAM technique for improving the accuracy of performance predictions for systems characterized by burstiness. Section 3.1 provides a high-level overview of WAM. Section 3.2 describes WAM for mixed systems characterized by open session arrivals. Section 3.3 discusses how WAM has to be adapted to study closed systems where the number of concurrent sessions is constant. Section 3.4 concludes the chapter.

3.1. Overview of WAM

Figure 3.1 shows the process of applying WAM to predict the performance of a system under study under a given workload. WAM is a trace-based technique. The method takes as input a trace of sessions representing a particular workload. The trace can either be a historical trace or a synthetic trace generated by a tool such as SWAT. Both types of traces record session identifiers, the time at which the first request in a session arrives at the system under study, the set of think times for a session and an identifier for the last request in a session. In addition a historical trace records response times for requests in a session. *Response time* is defined as the difference between the instant at which a request was completed by a system and the instant at which the request arrived at the system. Historical traces collected at production systems can be used to predict a system's performance if it were subjected to those workloads. Synthetic traces can be used to explore how characterizations for various workload attributes that influence burstiness impact system performance. The WAM technique also relies on a predictive model for

the system under study. The model should be capable of providing accurate performance estimates for various customer population levels.

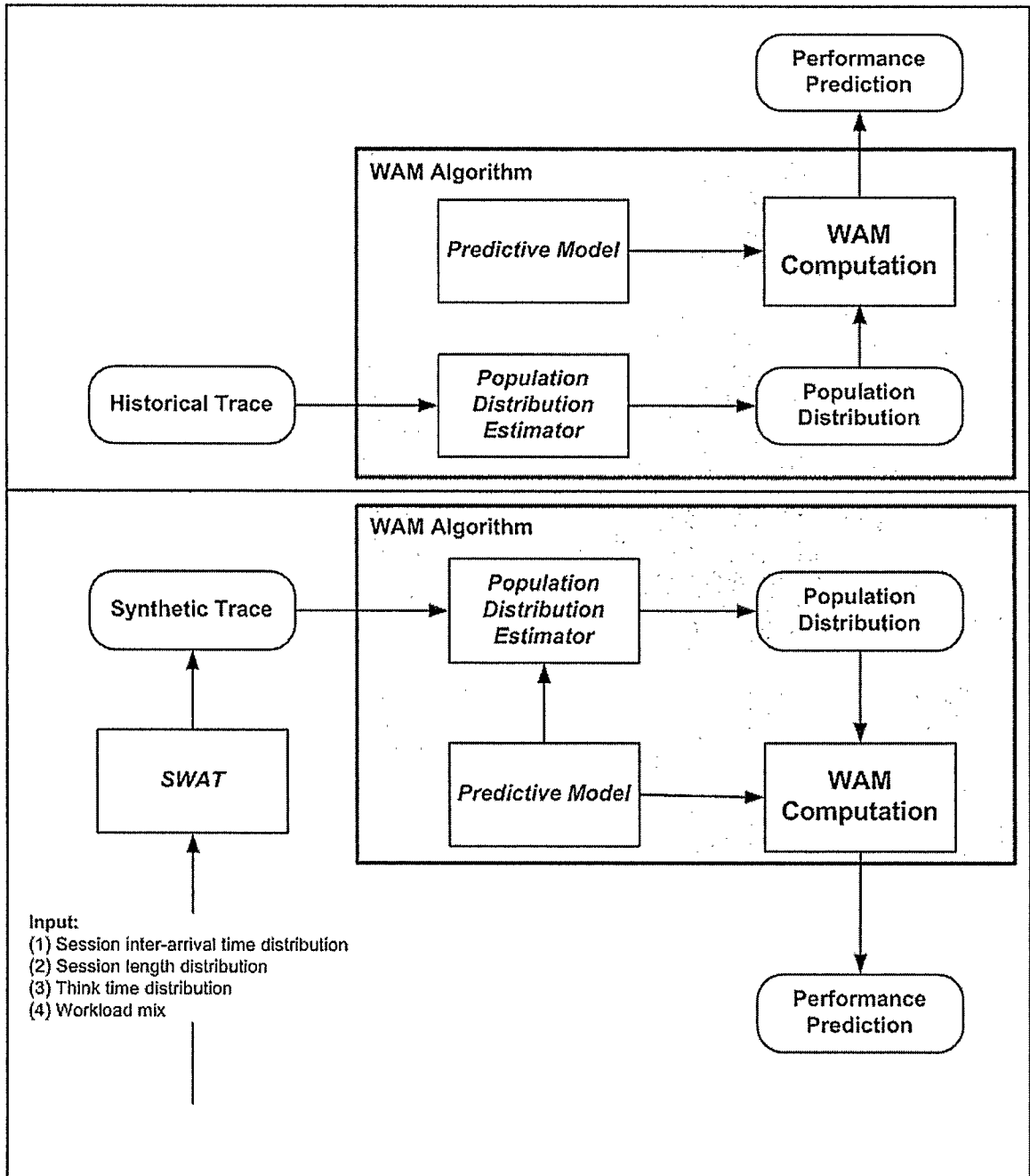


Figure 3.1: Performance modeling process with WAM

As mentioned previously, WAM estimates the population distribution and uses it to take into account the impact of bursts. The population distribution is a function of workload characteristics (e.g., distributions of session inter-arrival time, session think times, and session length) and the response times encountered by the requests in sessions. The request response times are unknown for a synthetic trace. As a result, WAM uses the trace in conjunction with mean response times from the predictive model to estimate the population distribution. Since request response times are known for a historical trace WAM does not need the predictive model for estimating the population distributions for such traces. For both types of traces, the estimated population distribution is used in combination with the predictive model to offer mean estimates of performance metrics such as request response time and throughput.

3.2. WAM for Systems with Varying Number of Concurrent Sessions

WAM is described for *mixed systems* where the number of concurrent sessions varies over time. This type of system is considered first since session-based systems typically exhibit such type of behaviour. As mentioned in Chapter 1, the population distribution for such systems is defined as the distribution of number of concurrent customer sessions. Closed QNMs or LQMs that are often used by practitioners can only consider the mean customer session population. This thesis argues that such an approach may not be appropriate for bursty systems. Instead WAM offers predictions that consider the population distribution instead of just the mean population.

WAM is motivated by the hierarchical approach described in Section 2.2.3. However it uses population distribution estimator (PDE) to quickly estimate the population

distribution, i.e., per population level probabilities $P(k)$ for $k = 0 \dots N$, rather than relying on the closed formulae for the birth-death process from (2.2) and (2.3). The remainder of the performance prediction method is similar to that shown in Figure 2.2. The population distribution estimation process is now described. The algorithm for the population distribution estimator in WAM is summarized in Figure 3.2.

The WAM approach relies upon the following:

- A trace file of sessions \mathbf{S} ;
- A sequence \mathbf{R} of mean request response time estimates $R(k)$ for $k = 0 \dots N$ for the system – one for each concurrent session population level k as obtained by solving a predictive model with a mean customer think time of Z seconds;
- A sequence \mathbf{X} of mean request throughputs $X(k)$ for $k = 0 \dots N$ for the system – one for each concurrent session population level k .

As mentioned previously, the trace file \mathbf{S} can be based on a historical session log from a real system, or it can be synthetically generated using a tool such as SWAT. Each request has a *session identifier*, a *start time* and *end time* such that (*end time* – *start time*) is the *response time* of the request, and a flag that indicates whether a request is the last request for its session. For all but the last request in a session, the request's *think time* is defined as the time between its *end time* and the *start time* of the next request in the session. The first request of a session has a *start time* that is equal to its session's *arrival time*. The sequence \mathbf{R} of response time estimates is obtained from a performance model for the system, e.g., a QNM or a LQM. The sequence \mathbf{X} of throughput estimates are obtained from the trace \mathbf{S} using the method described shortly.

```

1. Create a Future Event List (FEL). FEL stores events in chronological order.
2. Current_Population=0
3. State_Start_Time=0
4. State_End_Time=0
5. Initialize elements of Aggregate_State_Time array to 0. This array has  $N_{max}$  elements where  $N_{max}$  is the
   maximum population.
6. Initialize elements of Aggregate_State_Completions array to 0. This array has  $N_{max}$  elements.
7. Obtain Predictive_Model_Response_Time array by solving a predictive model. This array has  $N_{max}$ 
   elements.
8. Create request submission events corresponding to first requests of all sessions in trace S. Store the
   events in the FEL
9. While FEL is non-empty
   Select earliest event in FEL
   If event is submission of a request
       If request is first request in a session
           State_End_Time = start time of request
           Aggregate_State_Time[Current_Population]+=(State_End_Time-State_Start_Time)
           Completions=Request completions in the period (State_Start_Time, State_End_Time)
           Aggregate_State_Completions[Current_Population]+=Completions
           State_Start_Time=State_End_Time
           Current_Population+=1
       End If
       If S is a historical trace
           Response_Time = Get actual response time of request
       End If
       If S is a synthetic trace
           Response_Time = GenSyntheticResponse(State_Start_Time , Current_Population, FEL )
       End If
       Create a request completion event at (State_Start_Time+Response_Time)
       Update FEL with the event
       Continue
   If event is completion of a request
       If request is last request in a session
           State_End_Time = end time of request
           Aggregate_State_Time[Current_Population] +=(State_End_Time-State_Start_Time)
           Completions=Request completions in the period (State_Start_Time, State_End_Time)
           Aggregate_State_Completions[Current_Population]+=Completions
           State_Start_Time=State_End_Time
           Current_Population - =1
       Continue
   End If
   Think_Time = Get think time of request
   Create a request submission event at (State_Start_Time+Think_Time)
   Update FEL with the event
   Continue
End While
10. Compute Total_Time as sum of elements of Aggregate_State_Time
11. Compute  $P_k$  values by dividing each element of Aggregate_State_Time by Total_Time
12. Compute  $X_k$  by dividing each element of Aggregate_State_Completions with the corresponding element
    of Aggregate_State_Time
13. Use equation (2.6) to compute mean response time

```

Figure 3.2: The WAM algorithm

The population distribution estimator operates as follows. When using a historical trace file \mathbf{S} , per request response times are known so the sequence \mathbf{R} of response time estimates is not needed to compute the population distribution. The population distribution estimator computes the $P(k)$ for $k = 0 \dots N$ values by traversing the trace of sessions \mathbf{S} noting when the first request of each session starts and the last completes. In this way it is able to keep track of and report the aggregate time that the system has spent at each session population level. When normalized with respect to the total simulated time this gives the population distribution $P(k)$, $k = 0 \dots N$. Furthermore, as shown in Figure 3.2 the population distribution estimator also tracks the aggregate number of request completions observed at each session population level. Knowledge of these completions and the aggregate time spent at each session population level allows to compute estimates of $X(k)$ for $k = 0 \dots N$. The simulations are very quick, essentially requiring the time to traverse the trace file and are robust with respect to arbitrary workload parameter distributions.

When using a synthetically generated session trace file, the session arrival times, think times, and session lengths are known from the trace. However, only the first request's start time is known. The request response times and hence the request end times are not known. As the population distribution estimator traverses the session trace, each time it encounters a new request, it estimates the request response time using the *GetSyntheticResponse* method. This method, which will be discussed shortly, estimates a request's response time by considering the initial population level when the request is made and the possible changes of population level between the request's start time and its estimated end time. The estimated response time $R_{estimate}$ is used to estimate the *end time*

for the request as $start\ time + R_{estimate}$. The request's think time is recorded in the trace and could be from any desired distribution. The next request has a $start\ time$ equal to the $end\ time + think\ time$ from the previous request.

The *GetSyntheticResponse* method is used to estimate request response times and hence the request end times for a synthetic trace. As shown in Figure 3.3, this method estimates the response time of a request by considering the initial population level at the instant the request is submitted and the subsequent changes in population levels during the requests estimated duration.

```

1. currentTime = current time,  $T = currentTime$ 
2. Current population at  $currentTime = i$ 
3.  $tmpEndTime = currentTime + R[i]$  ( $R[i]$  is a value from sequence R when population =  $i$ )
4.  $endTimeAdjustmentFactor = 1$ 
5.  $nextArrivalTime = getNextArrivalTime(sessionArrivalTimeArray)$ 
6.  $nextCompletionTime = getNextCompletionTime(sessionCompletionTimeArray)$ 

While ( $nextArrivalTime < tmpEndTime \parallel nextCompletionTime < tmpEndTime$ )
  If ( $nextCompletionTime < nextArrivalTime$ )
     $T = nextCompletionTime$ 
     $endTimeAdjustmentFactor = endTimeAdjustmentFactor - (T - currentTime) / R[i]$ 
     $i = i - 1$ 
     $currentTime = T$ 
     $tmpEndTime = T + endTimeAdjustmentFactor * R[i];$ 
     $nextCompletionTime = getNextCompletionTime(sessionCompletionTimeArray)$ 
  End If
  If ( $nextCompletionTime > nextArrivalTime$ )
     $T = nextArrivalTime$ 
     $endTimeAdjustmentFactor = endTimeAdjustmentFactor - (T - currentTime) / R[i]$ 
     $i = i + 1$ 
     $currentTime = T$ 
     $tmpEndTime = T + endTimeAdjustmentFactor * R[i]$ 
     $nextArrivalTime = getNextArrivalTime(sessionArrivalTimeArray)$ 
  End If
End While

```

Figure 3.3 Algorithm of the *GetSyntheticResponse* method

Consider the first request in a session. The arrival time for the first request is known and is denoted as *currentTime*. The population at this instant is calculated as *i*. The response time is initially estimated as R_i where R_i is the mean response time prediction from the predictive model at population level *i*. The end time of the request is temporarily obtained as $tmpEndTime = currentTime + R_i$. A variable called the *endTimeAdjustmentFactor* is maintained for each request and its value is initialized to 1. This variable can take values from 0 to 1. It represents the amount of processing that remains for a request. A value of 1 indicates that the system has just started to process the request while a value of 0 indicates that the request has been completely processed.

The *tmpEndTime* estimate has to be adjusted to reflect changes to the population as the request is being processed. In the time period $[currentTime, tmpEndTime]$, the population level can change due to the arrival or departure of *other* sessions. An arrival of a session can cause increased contention among requests for system resources. Consequently, the estimated end time *tmpEndTime* has to be increased to take into account this contention. Similarly, *tmpEndTime* has to be decreased to reflect the potential decreased contention due to departing sessions. Two alternative scenarios will now be considered. The first scenario considers the arrival of a new session at time *T* where *T* is in the interval $[currentTime, tmpEndTime]$. The second scenario considers a session departure (other than that of the session being processed) at *T*.

When a session arrives at *T*, the *endTimeAdjustmentFactor* is updated as follows.

$$endTimeAdjustmentFactor = endTimeAdjustmentFactor - \frac{T - currentTime}{R_i} \quad (3.1)$$

In (3.1) i is the population observed just before a new session. After calculating the new $endTimeAdjustmentFactor$, the population is increased by 1 to $i+1$ to reflect the new session and the new end time of the current request is computed as follows:

$$tmpEndTime = T + R_{i+1} * endTimeAdjustmentFactor \quad (3.2)$$

Equation (3.2) indicates that the remainder of the request (as measured by $endTimeAdjustmentFactor$) will be processed slower by the system (i.e., at the “rate” of $1/R_{i+1}$).

For the scenario where a session departs at time T , the $endTimeAdjustmentFactor$ is updated as before by using (3.1). However, now the population is decreased by 1 to $i-1$ and the updated request end time is computed as follows:

$$tmpEndTime = T + R_{i-1} * endTimeAdjustmentFactor \quad (3.3)$$

Equation (3.3) indicates that the remainder of the request will be processed faster by the system (i.e., at the “rate” of $1/R_{i-1}$).

The $currentTime$ is updated as T and the value of T is updated to the next session arrival or session departure event. The calculations of the $endTimeAdjustmentFactor$ and the $tmpEndTime$ are repeated until there is no new session arrival or session departure (other than the session being processed) during the time interval $[currentTime, tmpEndTime]$. The difference between the final value of $tmpEndTime$ and the request start time gives an estimate of the request’s response time. As mentioned previously, the arrival time of the

next request in the session is computed by adding the think time to *tmpEndTime*. The *GetSyntheticResponse* method is again invoked for this new request to compute its end time.

As shown in Figure 3.2, the population distribution estimated in this fashion is used with (2.6) to compute the mean response time. For synthetic traces, *GetSyntheticResponse* provides a heuristic approximation of the real response time of a request. The effectiveness of this approximation is explored further in Sections 4.4 and 5.5.

Due to its use of Monte Carlo simulations, WAM can be used to explore the predictability of a system's behaviour. By using different seeds for random number generation, SWAT can be used to generate multiple synthetic session trace files that match the desired workload distributions. Each trace may provide different estimates for the population distribution and may result in a different estimate for mean system request response time. As mentioned in chapter two, this is expected for systems influenced by bursty workloads such as heavy-tailed distributions. Each execution is an example of how the system may behave. A range of estimated mean response times, from multiple simulations, provides information about how variable, i.e., unpredictable, a system's behaviour can be expected to be.

3.3. WAM for Systems with Constant Number of Concurrent Sessions

Many practitioners and academics employ user-equivalents (i.e., closed) request generation while evaluating session-based systems. For example, a number of performance studies [10][36] used the user-equivalents based default TPC-W workload

generator. A consequence of such a choice is that the number of concurrent sessions is constant during a performance test. Each user-equivalent acts as a session and all sessions start (near) simultaneously and end (near) simultaneously. Such a scenario is not likely to be valid for real session-based systems. A workload generator that is more representative of real systems would allow the number of concurrent sessions to vary [37] facilitating the method described in the previous section. However, due to the prevalent wide use of user-equivalents based request generation this thesis proposes techniques to adapt WAM for such studies.

As mentioned in Chapter 1, for a closed system the population distribution is defined as the distribution of concurrent requests at the system. In other words the state is now the number of concurrent requests contending for system resources. This number can vary from 0 to N_{max} where N_{max} is the number of concurrent sessions (i.e., number of user-equivalents). Another way of interpreting the state is that it represents only those sessions that are consuming and waiting for system resources and excludes sessions that are “thinking”. The predictive model for a closed system quantifies the contention among requests for system resources. Specifically, it provides response time estimates for various numbers of concurrent requests in the system. Due to the new choice of system state the predictive model uses a mean think time of zero second.

The simulation process remains similar to that for mixed systems except that WAM now keeps track of the number of concurrent requests in the system. Furthermore, the events of interest are now the arrival and departure of requests as opposed to the arrival and departure of sessions. For synthetic traces, whenever a new request arrives or departs the

GetSyntheticResponse method is invoked to estimate request response times. Lastly, as mentioned before, the predictive model used for this type of system provides performance estimates for various numbers of concurrent requests. The mean response time is computed as before by applying (2.6).

3.4. Conclusions

This chapter described WAM. WAM is a hybrid technique that combines simulation with analytic modeling. For a mixed system, the technique estimates the distribution of number of concurrent sessions for any given workload. For a closed system, WAM estimates the distribution of number of concurrent requests for a given workload. The estimated population distribution is used in combination with a predictive model to offer predictions for metrics such as mean request response time and mean throughput. In the ensuing chapters the effectiveness of WAM is assessed by using it to model the performance of two different TPC-W systems.

CHAPTER 4: C-TPC-W CASE STUDY

This chapter presents a case study of a multi-tier TPC-W application system. The system was installed at Carleton University in Ottawa, Canada [6]. It is referred to as C-TPC-W. Several performance test experiments were carried out for this system. C-TPC-W does not use the default user-equivalents based TPC-W workload generator for these experiments. It instead uses the SWAT workload generator that employs mixed workload generation. As a result, the number of concurrent sessions varies over each experiment.

Measurements collected from experiments are used to characterize the accuracy of WAM. Specifically, the system is subjected to several controlled synthetic workloads generated by the SWAT tool. Experiments using the synthetic workloads yield measurements that provide insights on how the system's performance is affected by distributions of session inter-arrival time, session length, and session think times and workload mix. A QNM and an LQM are developed for the system. For each experiment, the corresponding measurements are used to obtain parameters for these models. The WAM technique is then independently used in combination with the LQM and QNM to predict the mean response time for that experiment. The predicted mean response times are then compared with the measured mean response time for the experiment. The process is repeated for all experiments to assess the accuracy of WAM for various bursty as well as non-bursty workloads.

The experiment test bed for the C-TPC-W is described in section 4.1. Section 4.2 analyzes the measurements collected from experiments to derive insights on how various

workloads impact the system's performance. The predictive performance models, both a QNM and a LQM, are developed for the C-TPC-W system in section 4.3. Section 4.4 compares the straightforward application of these models that does not consider the population distribution with the hierarchical Markov chain approach described in Section 2.2.3 and WAM which considers the population distributions.

4.1. Experiment Test Bed

4.1.1. Experiment Setup

The experimental setup consists of a client node, a Web and application server node and a database node connected together by a non-blocking Fast Ethernet switch, which provides dedicated 100 Mbps connectivity to each node. The *client* node is dedicated exclusively to an httpperf Web request generator [16] that submits the synthetic workloads generated by SWAT. Logs generated by httpperf record request response times observed during an experiment. The logs also record for each session the times at which requests are issued, the times at which requests completed, and the number of requests submitted. This information can be analyzed to determine the measured population distribution during an experiment. The *Web/App server* node implements the TPC-W application's business logic software and communicates with the TPC-W database. The *DB server* node executes the database server software which manages the TPC-W database. Finally, a windows performance monitoring utility is employed that collects a user-specified sets of performance measures from both server nodes at regular specified sampling intervals. The experiment setup is shown in the Figure 4.1.

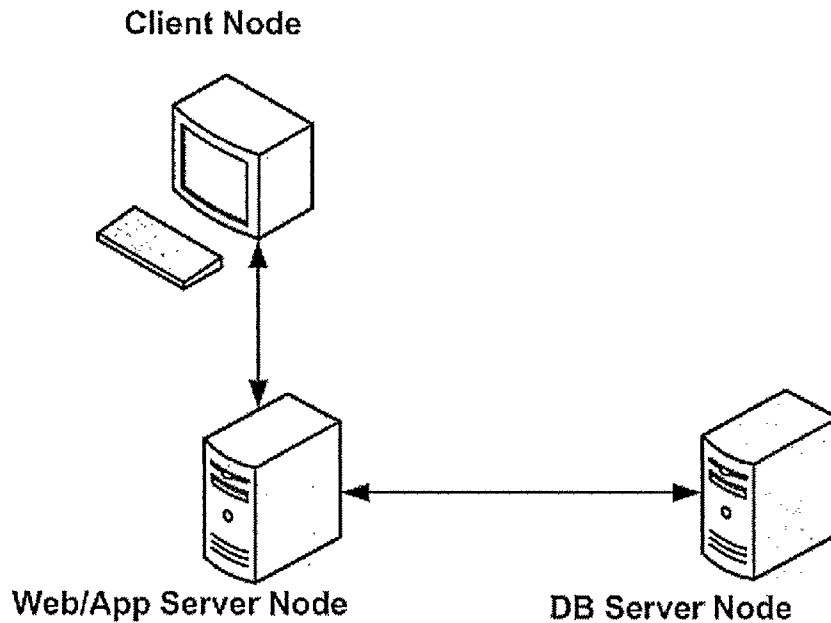


Figure 4.1 Experiment setup for the C-TPC-W system

The C-TPC-W application is deployed on Web, application, and database servers that are part of a commercial off-the-shelf software product. The name of the product has been withheld due to a non-disclosure agreement with the vendor. The system is configured to not serve images. Image request were not submitted in any of the experiments. This choice is consistent with the policy followed in several production systems [4][38]. In these systems images are hosted on separate servers or on content distribution networks. It is noted that the experiments presented in this study are not TPC-W benchmark runs. The TPC-W bookstore system merely serves as an example system for the study.

All the experiments employ HTTP 1.1 over SSL. Configuration parameters related to HTTP 1.1, e.g., persistent connection timeout, are chosen to force a single connection per session irrespective of session duration or the load on the system. This ensures that two workloads with the same number of sessions, mean session length, and mean think time

impose the same connection establishment and connection shutdown overheads on the Web server. Consequently, any difference in performance between them is solely due to the difference in the high-level workload characteristics, i.e., session length distribution, think time distribution, and workload mix.

The number of server processes and the threading levels are set as follows. The number of Web server threads is set to be 1000. This was much greater than the maximum number of concurrent connections encountered in the experiments. The number of application server processes is fixed at 16, an upper limit imposed by the application. The number of database server threads for the database server was set to the upper limit of 32.

The primary performance metric of interest for the study is the user-perceived mean response time (R_{mean}) for the requests at the TPC-W system. This metric is of interest for system sizing, capacity planning and service level management excises. *Response time* is defined as the time between initiating a TCP connection for a HTTP request and receiving the last byte of the corresponding HTTP response. The measured response time indicates the delay suffered by the request at the TPC-W system, provided the network and the client workload generator node are not saturated.

4.1.2. Experiment Design

The following factors are considered for the experiments: a) session inter-arrival time distribution; b) session length distribution; c) think time distribution; d) workload mix; and, e) application settings.

For the *session inter-arrival time* distribution, session arrivals are assumed to be uncorrelated. This is consistent with several previous studies, e.g., [39]. Consequently, an exponential distribution is used to generate session inter-arrival times for all experiments. The mean session inter-arrival time is chosen to achieve desired utilizations at the bottleneck resources. It should be noted that an exponential session inter-arrival time distribution does not imply a non-bursty, uncorrelated arrival of requests. As mentioned in Chapter 2, the burstiness of the request arrival pattern at the system depends on various attributes such as the distributions of session length and think time. Evidence of bursty request arrivals in workloads used for this study is presented in Section 4.2.

For the *session lengths* and *think times*, two different distributions are considered namely, empirical and bounded Pareto. These are used to represent the expected and worst cases for variability, respectively.

The empirical distributions are obtained from workload data collected from a large e-commerce system [38]. Since that system did not server requests for images embedded in Web pages, the request inter-arrival times within a session as measured at the system are used as an approximation of the think times within sessions.

The bounded Pareto distribution [40], a “heavy-tail-like” distribution, is used to study the impact of distributions that have a slightly heavier tail related to the empirical distribution. The probability density function of the bounded Pareto distribution is defined as followed:

$$f(X = x) = \frac{\alpha * k^\alpha * x^{-\alpha-1}}{1 - (\frac{k}{p})^\alpha} \quad (4.1)$$

Where α is defined as the Pareto index, which governs the rate at which the tail of the distribution decays; k is defined as the minimum possible value for any random variable X ; p is defined as the maximum possible value for any random variable X .

In this case study, the parameter p is set to the maximum observation for the session length or the think time obtained from the empirical distribution. Then k and α are chosen such that the mean of the empirical distribution is matched. The value of α chosen in this manner are 1.16 for the session length distribution and 1.10 for the think time distribution. Table 4.1 shows the minimum, maximum and the mean of observations obtained with the distribution for the synthetic workloads used in the study.

Table 4.1: Statistics of the session length and the think time

		Empirical	Bounded Pareto
Session Length (requests per session)	Minimum	3	3
	Maximum	120	120
	Mean	9.44	9.44
Think Time (s)	Minimum	0	12
	Maximum	900	900
	Mean	46.54	46.54

Three *workload mixes* are considered with different levels of variability in request resource demands. The workload mix includes HiMix, MedMix and LowMix workloads with high, medium, and low resource demand variation, respectively. The HiMix and the MedMix share the same average resource demands but MedMix has a little lower variation in demands so that it is comparable how the demand variations in the workloads impact the system's performance. The LowMix has lower average resource demand as well as the lower variation of demands for each request type, which used to compare to the other two mixes to show the impact on the system.

Firstly the mean “no-load” response time (R_{mean}) is measured for each of the 14 TPC-W request types. The R_{mean} values are obtained when the number of concurrent sessions is set to one. Consequently, they reflect the end-to-end resource demands across all resources for request types for the TPC-W system. The TPC-W Shopping mix [9] is used as a high demand variation mix (HiMix) in this study. A slightly different mix is defined with *slightly lower variation in demand* (MedMix). To construct such a mix, the proportion of requests belonging to the top two resource intensive request types Buy request and Buy confirm and the non-resource intensive Home request type are reduced, while the proportion of requests belonging to the Product detail request type, relative to the HiMix, is increased accordingly. Finally, the LowMix is constructed to reflect a mix that has a slightly lower mean demand and lower variation in demand than both the HiMix and the MedMix. As show in the table 4.2, this is achieved by eliminating certain resource intensive request types such as Buy request followed by a concomitant increase to the less resource intensive Home request type.

The response time $MeanR_{mean}$ for each mix is computed as followed:

$$MeanR_{mean} = \sum_{i=1}^{14} R_{mean,type=i} * p_i \quad (4.2)$$

Where $R_{mean,type=i}$ represents the no-load mean response time when type is i (total 14 request types are considered), and p_i represents the proportion of request type i .

Table 4.2: Workload mixes and no-load response time of request types

Request Type	$R_{mean}(s)$	HiMix	MedMix	LowMix
Home	0.09	16.00%	9.00%	23.46%
New Product	0.18	5.00%	5.00%	5.00%
Best sellers	0.18	5.00%	5.00%	5.00%
Product details	0.23	17.00%	27.80%	17.00%
Search request	0.07	20.00%	20.00%	20.00%
Search results	0.13	17.00%	17.00%	17.00%
Shopping cart	0.24	11.60%	11.60%	11.60%
Customer registration	0.21	3.00%	3.00%	0.00%
Buy request	0.63	2.60%	0.00%	0.00%
Buy confirm	0.25	1.20%	0.00%	0.00%
Order display	0.18	0.66%	0.66%	0.00%
Order inquiry	0.05	0.75%	0.75%	0.75%
Admin request	0.09	0.10%	0.10%	0.10%
Admin confirm	0.14	0.09%	0.09%	0.09%
<i>Mean</i>$R_{mean}(s)$		0.16	0.16	0.14
<i>COV of Request Response Time</i>		0.62	0.41	0.39

The coefficient of variation (COV) of request response time in table 4.2 is involved to evaluate the variance of the resource demand times for each mix. It is computed as followed:

$$COV = \frac{\sigma}{MeanR_{mean}} \quad (4.3)$$

$$\sigma = \sqrt{\sum_i (R_{mean,type=i} - MeanR_{mean})^2 * P_i} \quad (4.4)$$

Where σ represents the standard deviation of the data sets and $MeanR_{mean}$ represents the overall mean response time for this workload mix. Table 4.2 shows the design causes the MedMix to have a slightly lower coefficient of variation (COV) of request response time than the HiMix while a slightly higher COV of request response time than the LowMix.

In this way, the experiments are designed to indicate how the system performs under the workloads with different resource demand variations stressing the system.

To establish the robustness of this modeling technique, the experiments are conducted with three different *application settings* Base, HighDiskU and BigDB. The Base setting corresponded to a TPC-W application configured with 1,000 books in the database. For the workloads under studied with this setting, the Web server node CPUs were found to be the bottleneck. The HighDiskU setting differs from the Base setting in terms of database server configuration. Specifically, the database server's main memory cache settings were modified to cause more database node disk I/Os for a given workload when compared to the Base setting. However, in spite of the increased I/Os, the Web server node CPUs were still the bottleneck for all the workloads explored for the HighDiskU setting. Finally, the BigDB setting corresponded to a TPC-W application with 100,000 books in the database. This configuration allowed to verify the effectiveness of this approach when the bottleneck shifts from the Web server node CPUs to the database server node CPU.

4.1.3. Experiment Methodology

Due to time constraints, not a full-factorial investigation of the workload and application factors was conducted discussed in the previous section. Instead SWAT was used to create carefully controlled workloads designed to exhibit the performance impact of combinations of the factors considered. As mentioned in Section 2.1.1, SWAT can create a trace of sessions with each session in the trace containing request sequences that are valid for the TPC-W system. Table 4.3 lists the workloads that were created by SWAT.

Each workload is described by four hyphen-separated tokens. The first token describes the session length and think time distribution of the workload. For each workload, the choice of distribution type, i.e., empirical or bounded Pareto, is always chosen to be the same for session length and think time distribution. BPSLZ indicates the use of the bounded Pareto distributions of Table 4.3 while EMPSLZ indicates the use of the empirical distributions of Table 4.3. The subsequent tokens describe the workload mix, the mean utilization of each processor in the Web/App server node ($U_{Web,CPU}$) observed over the experiment duration, and the application settings, in that order.

From Table 4.3, eleven experiments are conducted for this study. Each experiment is designed to study the impact of a given workload. As shown in Table 4.3, several statistically independent replications are conducted for each experiment. To achieve this, SWAT is used with different random number generator seeds to create several session traces that are statistically identical with respect to the workload characteristics described in Section 4.1.2. In each experiment replication 10,000 sessions are submitted to the TPC-W system. The duration of a replication varied from approximately 3 hours to 5 hours depending on the mean session inter-arrival time used. Each replication yielded around 95,000 response time observations. From Table 4.3, in total thirty nine experiment replications were conducted for this study.

Table 4.3: Response time and resource demand measurements from the case study

Workload	R_{mean} (s)	Mean R_{mean} (s)	$D_{\text{Web,CPU}}$ (ms)	$D_{\text{Web,Disk}}$ (ms)	$D_{\text{DB,CPU}}$ (ms)	$D_{\text{DB,Disk}}$ (ms)
BPSLZ-HiMix-77-HighDiskU	1.1	1.11	191.64	8.44	46.54	19.04
	0.93		190.65	8.49	46.8	18.94
	1.3		194.33	8.25	46.51	17.65
BPSLZ-HiMix-71-BigDB	2.02	2.09	189.53	8.95	110.76	6.67
	2.06		190.35	8.85	110.88	6.66
	1.63		189.54	9.38	111.42	7.08
	2.65		195.86	9.04	112.31	6.76
BPSLZ-HiMix-77-Base	1.03	1.06	191.02	8.27	39.11	5.48
	0.93		190.45	8.54	38.83	5.37
	1.22		193.57	8.13	38.95	5.36
EMPSLZ-HiMix-77-Base	0.85	0.94	189.45	8.42	39.57	5.86
	0.9		191.58	8.71	39.47	5.39
	0.97		191.15	8.43	39.6	5.76
	1.02		190.45	9.07	39.49	5.44
EMPSLZ-MedMix-77-Base	0.75	0.75	188.39	9.24	34.08	5.49
	0.75		191.57	8.52	36.8	5.6
	0.76		188.79	8.43	34.18	5.47
	0.74		186.59	8.41	34.2	5.49
BPSLZ-MedMix-77-Base	0.92	0.93	189.99	8.32	32.94	5.35
	0.86		190.17	8.14	33.97	5.52
	1.02		191.45	9.96	33.82	5.57
EMPSLZ-LowMix-77-Base	0.67	0.72	176.89	6.27	26.04	4.57
	0.79		179.09	6.61	25.95	4.84
	0.69		177.4	6.23	25.98	4.56
	0.71		177.18	6.41	25.96	4.91
BPSLZ-HiMix-71-Base	0.67	0.69	184.45	8.88	38.91	5.46
	0.7		185.72	9.1	39.07	5.57
	0.6		183.67	9.14	38.97	5.46
	0.78		186.19	9.35	39.05	5.55
EMPSLZ-MedMix-71-Base	0.56	0.55	183.09	9.33	36.26	5.32
	0.55		183.38	9.88	33.18	5.52
	0.57		183.89	8.97	33.99	5.44
	0.53		183.2	9.31	34.04	5.52
EMPSLZ-LowMix-71-Base	0.49	0.52	171.55	6.7	25.98	4.79
	0.52		172.11	6.84	25.94	4.96
	0.54		174.75	7.15	25.96	4.75
	0.52		173.59	8.1	26.06	4.72
EMPSLZ-MedMix-65-Base	0.43	0.44	178.09	10.97	34.06	5.7
	0.44		178.75	10.36	34.06	5.69

The following observations were consistent across all experiments. The httpperf provided highly reproducible results. As expected, multiple repetitions of an experiment replication yielded almost the same mean response time measures. Furthermore, there was very little difference between the achieved workload characteristics, as measured from httpperf logs collected from experiment replications, and the specified workload characteristics. This verifies that the client node was not saturated in this study. The worst-case mean and peak network traffic during the experiments was only 0.40 Mbps and 0.83 Mbps,

respectively. This is because the CPU intensive nature of HTTPS and application server processing limited request throughputs. The low network traffic indicates that the response time measured by httpperf is likely to be dominated by the delay encountered at the TPC-W system. The disks at both server nodes were very lightly utilized. Virtually no memory paging activity was observed at either server node. Finally, job flow balance was achieved for all experiments with the number of request completions equalling to the number of request arrivals.

4.2. Overview of Experimental Results

Table 4.3 provides several sanity checks with regards to the experimentation. The table presents the average per-request demands in milliseconds placed on the CPUs and disk of the Web/App server node, $D_{Web,CPU}$ and $D_{Web,Disk}$, respectively, and the database server node, $D_{DB,CPU}$ and $D_{DB,Disk}$, respectively. It also provides the mean response time of requests that were submitted in an experiment replication, R_{mean} , and the mean R_{mean} over all replications in an experiment. The following observations can be made from Table 4.3.

- The demand values for an experiment's replications are always nearly identical. This confirms that statistically identical replications place similar demands on the system and that burstiness does not affect average demands.
- For a given application setting, workloads with the same mix cause similar demands on system resources. This can for example be verified by comparing the demand measurements for the BPSLZ-HiMix-77-Base, EMPSLZ-HiMix-77-Base, and BPSLZ-HiMix-71-Base workloads.

- The measurements show that the mixes chosen for the study behaved as intended. From Table 4.3, for a given application setting the MedMix workloads impose almost the same average demands on the system as the HiMix workloads (compare for example EMPSLZ-HiMix-77-Base and EMPSLZ-MedMix-77-Base). As expected, the LowMix workloads place slightly lower demands on the system than the HiMix and MedMix workloads.
- The application settings explored also exhibited the intended behaviors. For example, the BPSLZ-HiMix-77-HighDiskU workload exerts more demand on the database server's CPU and disk when compared to the BPSLZ-HiMix-77-Base workload. Similarly, the database server CPU demand for the BPSLZ-HiMix-71-BigDB workload is significantly more than that of the BPSLZ-HiMix-71-Base workload.

Furthermore the BPSLZ replications are analyzed to determine whether they caused bursty arrival of customer requests. The Hurst parameter has been widely used in literature to quantify request burstiness [41]. A Hurst parameter estimate in the range of 0.5 to 1.0 for a time series indicates the presence of bursts (i.e., prolonged patterns of consecutive large or consecutive small values) over multiple timescales. For a given experiment replication, the series of times between successive request arrivals at the TPC-W system (i.e., the request inter-arrival times) is recorded. The Hurst parameter is then estimated for this time series using various methods shown in Table 4.4¹. Table 4.4

¹ For a description of these methods the reader is referred to the paper by Taqqu *et al.* [41].

shows the Hurst parameter estimates for a BPSLZ-HiMix-77-Base replication and a BPSLZ-HiMix-71-BigDB replication. Similar results were observed for the other replications. From Table 4.4, the Hurst parameter estimates are all greater than 0.5 indicating the presence of request bursts over several timescales in these workloads. This also shows that these workloads are bursty in spite of not requesting images.

Table 4.4: Estimates of Hurst parameter for BPSLZ replications

Method	BPSLZ-HiMix-77-Base	BPSLZ-HiMix-71-BigDB
Aggregated Variance	0.81	0.82
Absolute Values of the Aggregated Series	0.70	0.71
R/S	0.61	0.62
Periodogram	0.59	0.59

Results pertaining to the Base application setting along with a detailed discussion can be found in the earlier publication [6]. Now briefly description some of the salient findings of the results from a performance modeling perspective are given. As mentioned previously, additional results pertaining to the other two new application settings are presented also.

Distributions that cause highly variable session lengths and think times can cause bursty population distributions – This observation can be reached by first comparing the BPSLZ-HiMix-77-Base and EMPSLZ-HiMix-77-Base workloads in Table 4.3. These workloads only differ with respect to their session length and think time distributions. From Table 4.3, they place almost identical demands on the TPC-W system’s resources. The CPUs and disks in the systems have similar utilizations for both workloads. However, from Table 4.3, the mean R_{mean} for the BPSLZ-HiMix-77-Base

workload is about 13% higher than that of the EMPSLZ-HiMix-77-Base workload. Similarly, from Table 4.3, the mean R_{mean} for BPSLZ-MedMix-77-Base workload is about 24% higher than that of the EMPSLZ-MedMix-77-Base workload. These results suggest that the bounded Pareto session length and think time distributions are responsible for the performance degradation.

As mentioned in Section 2.2.4, high variability in session lengths and think times impact performance since they can cause bursty request arrivals. Specifically, such distributions yield large numbers of very small and very large session length and think time values. Consequently, BPSLZ-like workloads will have larger numbers of very long duration and very short duration sessions than EMPSLZ-like workloads. As a result, for any given mean session inter-arrival time, the likelihood of observing very large and very small number of concurrent sessions is more with a BPSLZ workload than with an EMPSLZ workload. This is illustrated in Figure 4.2 which shows the cumulative distribution function (CDF) of number of concurrent sessions for BPSLZ-HiMix-77-Base and EMPSLZ-HiMix-77-Base workloads². Since the number of requests that can arrive at the system is positively correlated with the number of concurrent sessions, this phenomenon causes a more uneven or bursty arrival of requests. This increase in burstiness can sometimes, as in the experiments, be significant enough to cause periods

² The CDF for a workload was obtained by combining data from all its experiment replications.

of heightened contention for system resources during which requests incur very long response times.

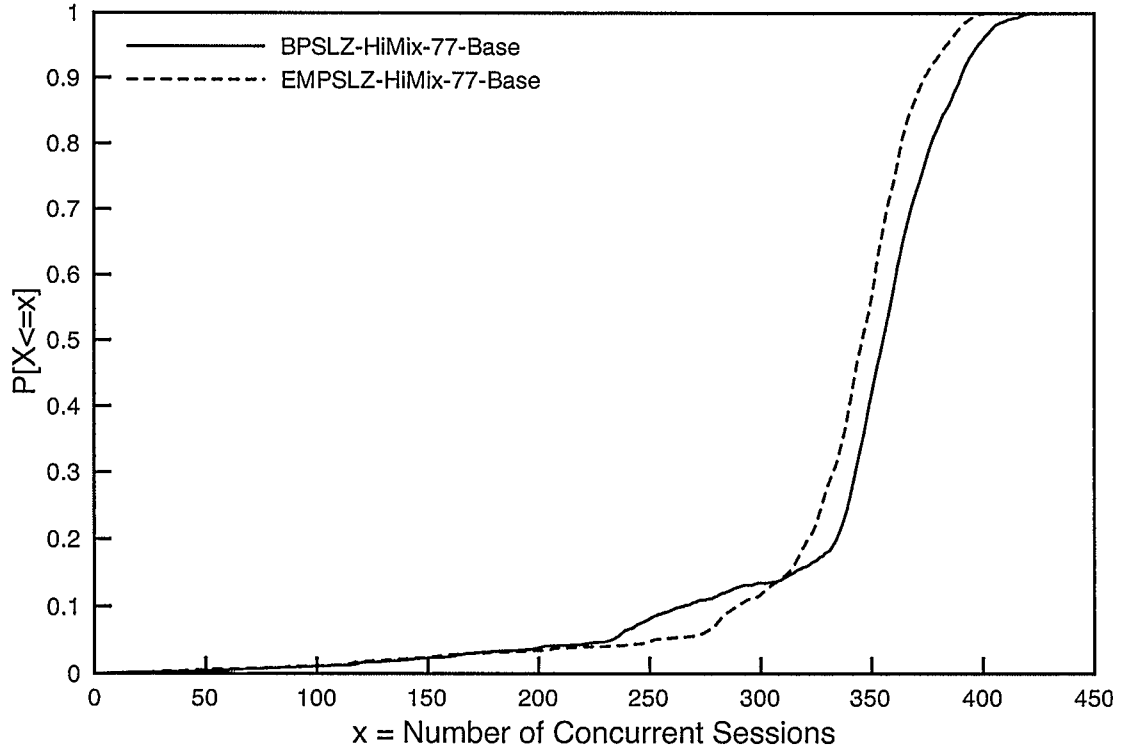


Figure 4.2: CDFs for BPSLZ-HiMix-77-Base and EMPSLZ-HiMix-77-Base

Mixes characterized by higher variability in request demands cause bursty population distributions – This conclusion can be verified from Table 4.3 by comparing the EMPSLZ-HiMix-77-Base and EMPSLZ-MedMix-77-Base workloads. Recalling from the previous sections, both these workloads are similar in all respects except their workload mix. From Table 4.2, both workloads place the same mean aggregate demands on the system’s resources. However, the HiMix workload is characterized by a slightly higher variability in request demands. Both workloads cause nearly identical utilizations

of the CPUs and disks in the system. However, the mean R_{mean} for the EMPSLZ-HiMix-77-Base workload is about 25% higher than that of the EMPSLZ-MedMix-77-Base workload. Figure 4.3 plots the CDFs of number of concurrent sessions for the workloads. From Figure 4.3, it can be seen that the HiMix workload exhibits a slightly longer tail than the MedMix workload. The reason for this behaviour is again due to the increased variability of session durations; the larger proportions of resource intensive, e.g., Buy request, and non resource intensive, e.g., Home, requests within sessions of the HiMix workload increases the likelihood of very long duration and very short duration sessions. This leads to periods of increased contention among sessions leading to a higher mean R_{mean} .

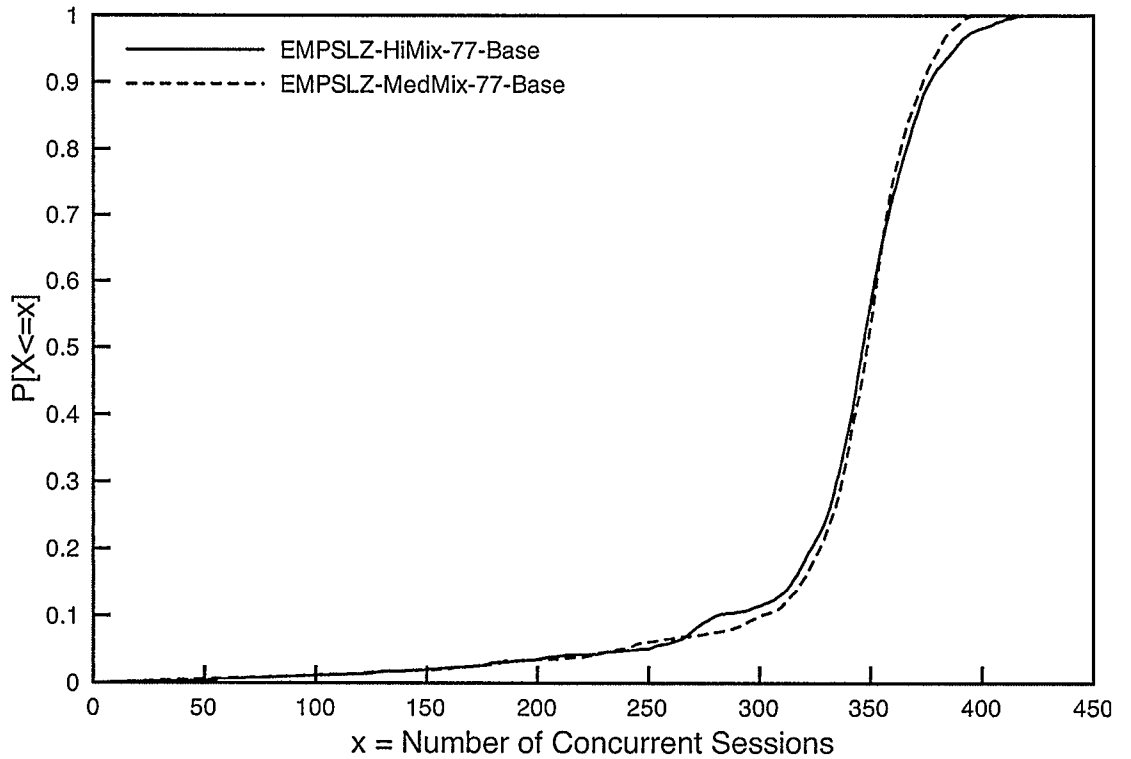


Figure 4.3: CDFs for EMPSLZ-HiMix-77-Base and EMPSLZ-MedMix-77-Base

Bursty workloads exhibit high variability in R_{mean} – As mentioned in Section 2.2.4, workloads characterized by heavy-tailed distributions lead to unpredictability in system behaviour. This phenomenon can be observed for the BPSLZ-HiMix-71-BigDB workload. Recalling from the previous section, this workload caused the database server node CPU to be the bottleneck. The mean database server node CPU utilization over the duration of each replication was 84%. From Table 4.3, the highest R_{mean} value of 2.65 seconds for this workload is 63% higher than the lowest R_{mean} value of 1.63 seconds. This is in spite of the fact that the experiment replications are statistically identical, cause near identical demands and utilizations on the system’s resources, and lasted for nearly 5 hours. Similar trends can be observed for the BPSLZ-HiMix-77-Base and BPSLZ-MedMix-77-Base workloads.

The reason for the variation in R_{mean} can again be explained in terms of the population distribution. Figure 4.4 plots the CDF of number of concurrent sessions for the four replications of the BPSLZ-HiMix-71-BigDB workload. Figure 4.5 plots the R_{mean} values for these replications. Figure 4.4 shows that the CDF is different for the different replications. In particular, replication 4’s CDF exhibits the longest tail and results in the highest R_{mean} while replication 3’s CDF has the shortest tail and causes the lowest R_{mean} . WAM can help performance analysts estimate the extent of variability in R_{mean} for bursty workloads by repeating the analysis multiple times with different session traces.

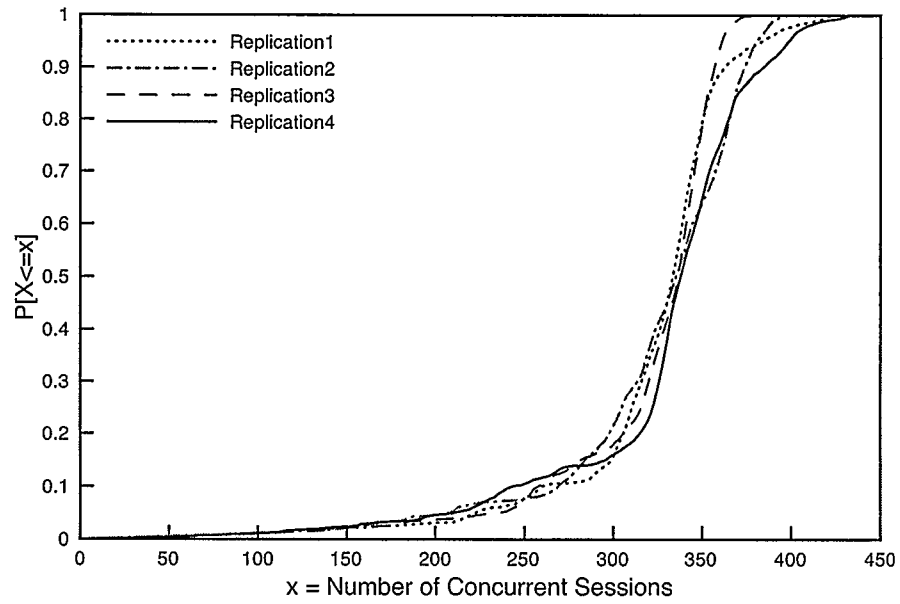


Figure 4.4: CDFs for BPSLZ-HiMix-71-BigDB workload

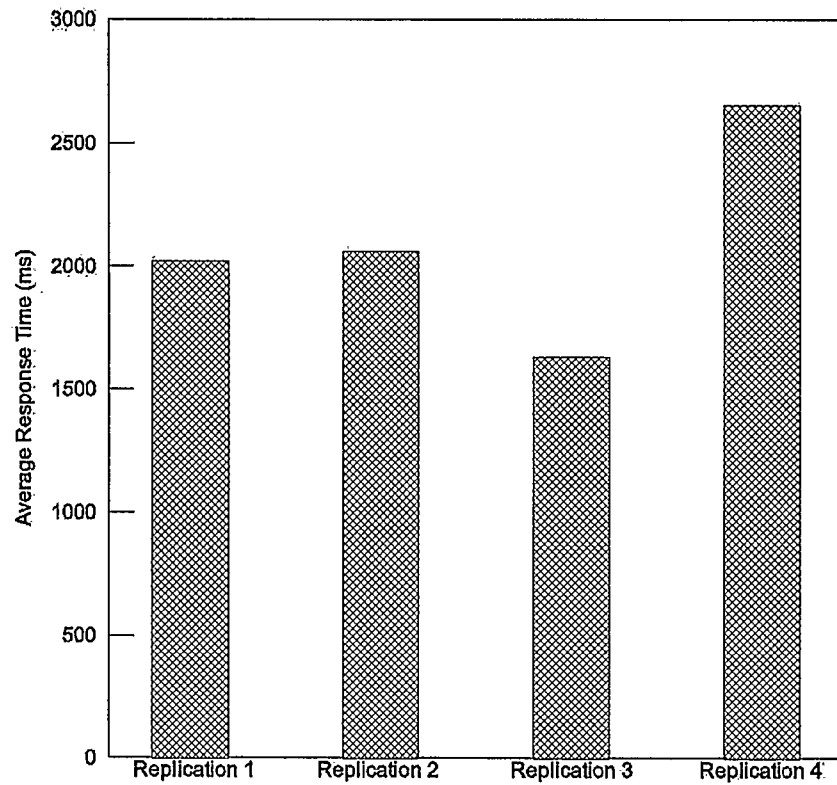


Figure 4.5: Measured R_{mean} values for BPSLZ-HiMix-71-BigDB

4.3. Predictive Performance Models for C-TPC-W

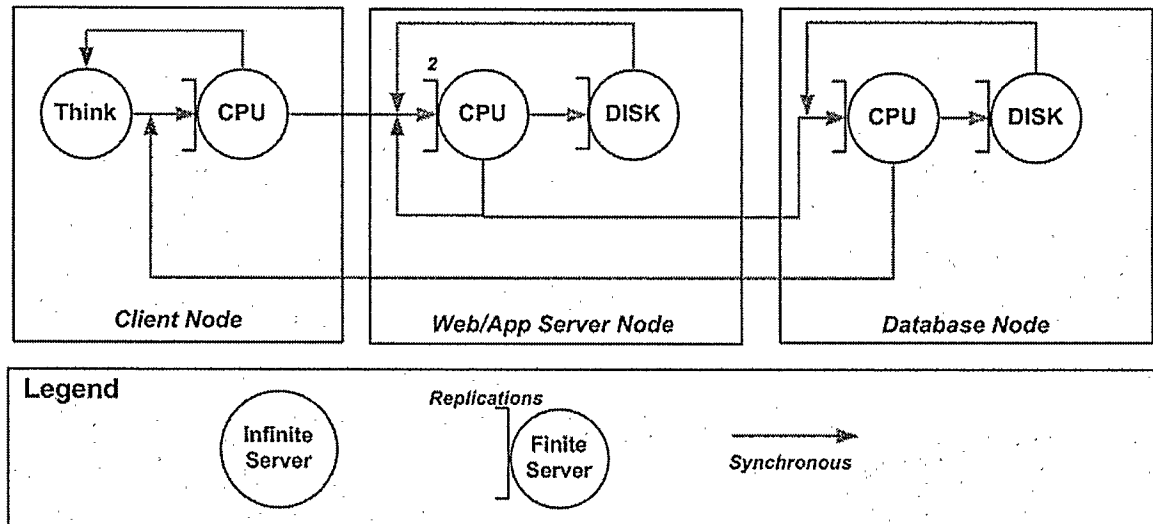


Figure 4.6: Queuing network model for C-TPC-W system

The QNM model, as shown in Figure 4.6, includes a think time delay service center and queuing service center for all the hardware resources, namely client node CPU, Web/App server node CPUs and disk, database server node CPU and database server node disk. The value 2 shown on the upper left of the Web/App node CPU indicates that the Web/App server has two CPUs. Other hardware resources are very lightly loaded so they are not included in the model. A customer that flows from the client node CPU through to the database server node CPU and back to the client node CPU completes a HTTP request. Customers flow from queue to queue. After visiting a CPU, a customer may have one or more alternative queues visit. The routing choices do not depend on the current state of the system, but are random and have probabilities such that the desired ratio of demands is incurred at the resources.

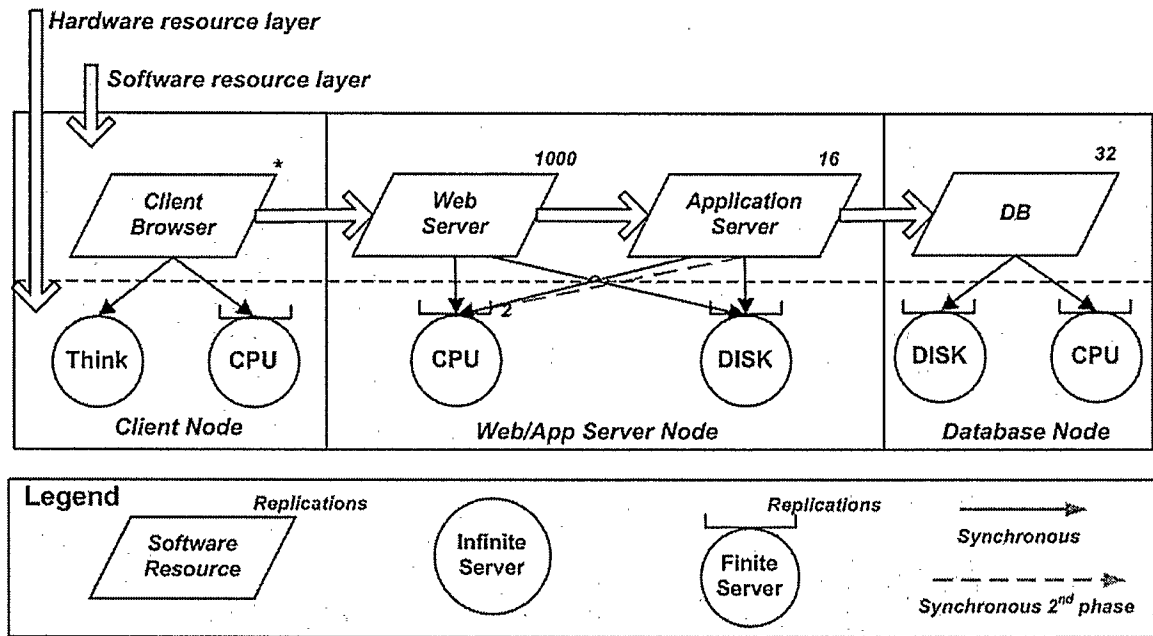


Figure 4.7: Layered queuing model for C-TPC-W system

Figure 4.7 shows the LQM for the TPC-W system. LQMs are extended QNMs that include information about logical resources such as threading levels for application servers and software request-reply relationships. The LQM for the TPC-W system includes the same think time delay centre and hardware resources. The logical resources in the model are the client browsers, Web server threads, application server threads and database server threads. Threading levels other than one are shown by placing a value near the upper right hand side of an icon. In this model, the requests are blocked between software resources and between software resources and hardware resources.

From Figure 4.7, there is one *client browser* for each concurrent session using the system. A customer using a client browser may visit its node's CPU or may think. A HTTP request causes a blocking call to the *Web server*. If a Web server thread is available then

the request is accepted. The thread uses some CPU resource from the Web/App server node CPUs and then makes a request to the *application server*. If an application server thread is available then the request is accepted. The application server thread uses some CPU resource from the Web/App server node CPUs and then makes a request to the *database server*. If a database server thread is available then the request is accepted. The thread uses some CPU and disk resource from the database server node and releases the calling thread. The released calling thread from the application server can then complete its *first phase* of work and release the calling thread from the Web server.

From Figure 4.7, after finishing its first phase and releasing the calling thread from the Web server the application server thread continues on to a *second phase* of service. The second phase of service keeps the application server thread busy so that it cannot service another calling thread. However at the same time the calling thread from the Web Server that was released after the first phase of service can complete its work and release the calling thread from the client browser. This completes an HTTP request.

During an HTTP request, if a thread is not available when a server is called, the calling thread blocks until a thread becomes available. Once a thread completes its work it is available to serve another caller. Such threading can lead to software queueing delays in addition to any contention for hardware resources that are incurred by active threads. The numbers of threads used for each tier in the model reflect the application settings as described in Section 4.1.

Both the QNM and LQM need as parameters the resource demands, the average think time and the number of customers in the system to calculate the performance metrics for the tested system using MVA.

To obtain resource demand values, for each experiment replication the CPU utilizations for the Web server threads, application server threads, and the database server threads are measured. Besides, the CPU and disk utilizations for the Web/App server node and the database server node, the elapsed time of the run, and the number of request completions, are measured as well. This enables to compute the average resource demand per request for the Web server threads, application server threads, database server threads, and for the Web/App server node and database server node as a whole. The aggregate demand values used in the models are given in Table 4.3. The formula [23] to calculate the demand is as follows:

$$D_k = U_k / X \quad (4.5)$$

Where U_k represents the utilization of the resource k , X represents the measured per-request throughput which is obtained as the total number of request completions divided by the total elapsed time of an experiment replication. It is noted that there was a very small difference between the utilization of a node and the sum of the utilizations of software processes running on that node. Thus this is modeled as background load in the LQM.

The average think time for each request can be computed from the httpperf logs collected from each experiment replication. The LQM and QNM obtained in this manner can provide mean response time estimates for different session populations.

4.4. Results of QNMs and LQMs with WAM

This section applies the QNM and LQM models of Section 4.3 with WAM to predict the mean response times for the experiments of Section 4.2. These results are further compared with the straightforward application of QNMs and LQMs and the hierarchical Markov chain birth-death approach of Section 2.2.3 for session based systems. All the modeling approaches used the Method of Layers (MOL) solver developed by Rolia and Sevcik [11].

Table 4.5 shows the four different modeling approaches that are explored for both QNMs and LQMs. The MEAN approaches ignore the distribution of number of concurrent sessions. They solve a predictive model for only one customer population, namely, the mean number of concurrent sessions observed during an experiment replication. The MBD methods use the Markov birth-death approach described in Section 2.2.3 to estimate the population distribution and R_{mean} . The birth-death model used a constant, state-independent birth rate³ that equals the mean session arrival rate observed during a measurement experiment replication. The WAMEMP methods predict R_{mean} for an

experiment replication by using WAM in conjunction with the empirical population distribution as measured during the replication. However, in practice, this measured population distribution may not be available to a performance analyst. The WAMMC method can be used to estimate the measured population distributions. For a given experiment replication, it uses the corresponding synthetic SWAT trace as input to a Monte Carlo simulation along with response times from the predictive model as per the algorithm in Section 3.2. The estimated population distribution is then used to compute R_{mean} . The accuracy of WAMMC will be evaluated by comparing its R_{mean} predictions to those of WAMEMP.

In general all the methods yielded good throughput estimates. The absolute errors for throughput were within 2% for the WAMMC methods and the MBD methods. The accuracy of the MEAN methods was slightly poorer. The throughput estimates of the MEAN-LQM and MEAN-QNM methods were within 3.5% and 4.0% of measured values, respectively. The reason for the good throughput estimates can be understood by noting that the think times are much larger than the mean response times in the experiments. From Little's law, the system throughput is largely influenced by the think time and is not affected significantly by errors in mean response time predictions.

³ Models with state-dependent birth rates were also tried but their accuracy was poorer than the state-independent approach.

In contrast to the throughput estimates, there are significant differences in prediction accuracy for R_{mean} across the different methods. Three different error metrics are used to characterize the R_{mean} prediction accuracy of the modeling approaches. The mean absolute error (ABS_Error) is defined as follows:

$$ABS_Error = 100 * \frac{\sum_i |e_i|}{\sum_i y_i} \quad (4.6)$$

Where e_i is the difference between the measured and predicted mean response time and y_i is the measured response time for the i^{th} replication in a set of replications. The maximum of the absolute e values, expressed as a percentage, calculated for a set of replications is denoted as the maximum absolute error (Max_Error). The trend error (Trend_Error) is an indicator of the range of errors obtained with a modeling approach. It is defined as the difference between the largest e value and the smallest e value, expressed as a percentage, for a set of replications. Table 4.5 shows the error measures for models pertaining to the entire set of thirty nine replications. The table gives results for the MEAN, MBD, WAMEMP, and WAMMC methods in conjunction with LQM and QNM.

Table 4.5: Accuracy for predicting R_{mean} for overall cases

Modeling Approach	ABS_Error (%)	Trend_Error (%)	Max_Error (%)
MEAN-LQM	11.77%	42.50%	32.37%
MEAN-QNM	13.10%	63.75%	42.56%
MBD-LQM	10.52%	45.09%	32.56%
MBD-QNM	12.97%	66.01%	42.68%
WAMEMP-LQM	6.57%	27.76%	15.80%
WAMEMP-QNM	10.01%	43.82%	26.84%
WAMMC-LQM	7.33%	34.64%	20.25%
WAMMC-QNM	11.64%	56.48%	29.75%

Firstly the MEAN cases are considered. These are the only cases that do not take population distribution into account. From Table 4.5, the ABS_Error is lower for the MEAN-LQM approach than the MEAN-QNM approach. The MEAN-LQM approach also does better in terms of Max_Error and Trend_Error. The improved prediction accuracy is due to the LQM taking into account the performance impacts of finite server thread pools and two phases of application server processing. However, the ABS_Error of about 12% and the Max_Error of about 32% are still large for the MEAN-LQM approach. These errors are large despite individual per session population level R_{mean} predictions from the LQM agreeing well with the corresponding measured values. This suggests there will be benefits from considering the population distributions.

As to the Markov Chain birth-death approach MBD, From Table 4.5, results show only slight improvements in ABS_Error. For example, the technique when used in conjunction with the LQM (MBD-LQM) achieves a reduction in ABS_Error of only about 1.3% when compared to MEAN-LQM.

WAM with the empirical population distribution from a historical trace with measured response times, WAMEMP, improves accuracy significantly. From Table 4.5, the `ABS_Error` drops by nearly 5% with WAMEMP-LQM when compared to MEAN-LQM. Moreover, `Max_Error` and `Trend_Error` drop by about 16% and 15%, respectively when compared to MEAN-LQM. Similar improvements are noticed when comparing WAMEMP-QNM and MEAN-QNM. This confirms the importance of considering the population distributions in modeling process.

Finally, the results of WAM with a population distribution estimated using the algorithm discussed in chapter 3 (WAMMC), from Table 4.5, show the effectiveness of the approach. The WAMMC methods performed nearly as well as their corresponding WAMEMP methods. For example, from Table 4.5 the error metrics for WAMMC-LQM are very similar to that of WAMEMP-LQM. However, WAMMC has an advantage over WAMEMP. It allows the WAM method to be applied in a constructive manner to predict the performance of systems when varying workload parameters and when a historical trace with measured response times is simply not available.

The WAMMC results validate the population distribution estimator's use of the R_{mean} prediction from a predictive model for the current population level to estimate the response time of an individual request with *GetSyntheticResponse* method as described in Section 3.2. It is suggested that the approach works well for these cases because the think times encountered in the synthetic workloads used for the study are much longer, on the order of tens of seconds, than the response times which are on the order of hundreds of milliseconds or seconds. As a result the population distribution is governed more by the

session length, think time, and session inter-arrival time distributions than the response time distribution for each population level. It is noted that an analysis of the empirical think time distribution of Table 4.1 indicates that the assumption of think times being much longer than response times is likely to be valid for real session-based workloads.

Now several subsets of the results are considered in more detail. Results are discussed for the following cases: bursty workloads; higher and lower contention for the bottleneck; higher and medium coefficients of variation for request resource demands; non-bursty workloads; and, workloads with heavy-tail-like distributions. Finally, a case is presented that demonstrates the constructive capability of WAMMC.

The WAM approach is particularly effective for bursty workloads - Table 4.6 summarizes the error measures for only those seventeen experiment replications that employed the bounded Pareto session length and think time distributions of Table 4.1. For bursty workloads using just the mean population provides very poor R_{mean} estimates. From Table 4.6, the `ABS_Error` is 19.10% for the MEAN-LQM approach. From Table 4.5 and Table 4.6, the MEAN-LQM approach applied to the bursty workloads results in about 7% greater `ABS_Error` than overall for all workloads. For these 17 workloads the WAM method results in a greater reduction in `ABS_Error` than overall for all workloads. For example, from Table 4.6, the `ABS_Error` for WAMEMP-LQM is about 14% lower than that for MEAN-LQM. This represents about 9% more reduction in error than when considering all the workloads. A similar trend can be noticed with WAMEMP-QNM. The WAMMC-LQM and WAMMC-QNM methods result in slightly increased `ABS_Error` when compared to their counterparts that use the empirically measured

population distribution. However, the errors are still significantly less than those obtained with the MEAN and MBD methods.

Table 4.6: Accuracy for predicting R_{mean} for bursty cases

Bursty Workloads (17)			
Modeling Approach	ABS_Error (%)	Trend_Error (%)	Max_Error (%)
MEAN-LQM	19.10%	28.04%	32.37%
MEAN-QNM	19.34%	45.11%	42.56%
MBD-LQM	16.98%	30.06%	32.56%
MBD-QNM	17.98%	46.79%	42.68%
WAMEMP-LQM	4.87%	25.84%	14.58%
WAMEMP-QNM	6.60%	28.43%	17.43%
WAMMC-LQM	6.41%	34.64%	20.25%
WAMMC-QNM	8.93%	41.08%	29.75%

The gains from WAM are significant when there is higher contention for the bottleneck resource - To illustrate this effect Table 4.7 compares the error metrics for the BPSLZ-HiMix-77 and BPSLZ-HiMix-71 replications. For the sake of clarity results are shown only for the MBD-LQM, WAMEMP-LQM, and WAMMC-LQM methods. From the Table 4.7, when $U_{Web,CPU}$ is 71% (BPSLZ-HiMix-71 Workload), WAMEMP-LQM results in an improvement of about 6%, 3%, and 10% in ABS_Error, Trend_Error and Max_Error, respectively, when compared to MBD-LQM. These numbers increase to 16%, 6%, and 18% when $U_{Web,CPU}$ is 77% (BPSLZ-HiMix-77 Workload). Previous studies have shown that the burstiness induced by heavy-tails becomes more pronounced at higher utilizations [42]. Consequently, the BPSLZ-HiMix-77 workload is more bursty than the BPSLZ-HiMix-71 workload. The increased gain in accuracy for the BPSLZ-HiMix-77 workload provides further evidence that WAMEMP is very effective for predicting the behaviour of bursty workloads. Similar results can be found in WAMMC-

LQM methods. It shows WAMMC-LQM is also effective to model the C-TPC-W system under bursty workloads.

Table 4.7: Accuracy for BPSLZ-HiMix- workloads

Workload	Modeling Approach	ABS_Error (%)	Trend_Error (%)	Max_Error (%)
BPSLZ-HiMix-77	MBD-LQM	20.95%	14%	28%
	WAMEMP-LQM	4.52%	8%	10%
	WAMMC-LQM	6.68%	10%	14%
BPSLZ-HiMix-71	MBD-LQM	12.32%	19%	21%
	WAMEMP-LQM	5.98%	16%	11%
	WAMMC-LQM	6.84%	10%	12%

WAM is particularly effective for mixes characterized by higher variability in request resource demands -Table 4.8 compares the MBD-LQM and WAM methods for the BPSLZ-HiMix-77 and BPSLZ-MedMix-77 workloads. Recall from Section 4.1 that the HiMix workload exhibits more variability in demands than the MedMix workload since it has a greater percentage of resource intensive and resource non-intensive requests. From Table 4.8, the MBD-LQM method results in an ABS_Error of 13.43% and a Max_Error of 17% for the MedMix workload. The method performs even poorer for the HiMix workload with the ABS_Error and Max_Error increasing to 20.95% and 28%, respectively. From Table 4.8, the WAM methods are significantly more accurate than MBD-LQM for both workloads. The gains in ABS_Error while using the WAMMC-LQM method over the MBD-LQM method are nearly 11% for the MedMix workload and 14% for the HiMix workload.

Table 4.8: Accuracy for BPSLZ-77- workloads

Workload	Modeling Approach	ABS_Error (%)	Trend_Error (%)	Max_Error (%)
BPSLZ-HiMix-77	MBD-LQM	20.95%	14%	28%
	WAMEMP-LQM	4.52%	8%	10%
	WAMMC-LQM	6.68%	10%	14%
BPSLZ-MedMix-77	MBD-LQM	13.43%	9%	17%
	WAMEMP-LQM	2.42%	4%	4%
	WAMMC-LQM	2.03%	2%	3%

WAM is effective for cases with non-bursty workloads -Table 4.9 summarizes the error measures for those experiment replications that did not use the bounded Pareto session length and think time distributions. From Table 4.9 and Table 4.5, the MEAN-LQM and MBD-LQM approaches have a much lower ABS_Error for these workloads than overall for all workloads. The ABS_Error for WAMEMP-LQM is comparable to those of MEAN-LQM and MBD-LQM. However, WAMEMP-LQM method results in a smaller range of errors when compared to the other two approaches. For example, the Max_Error and Trend_Error for WAMEMP-LQM are about 9% and 7% lower, respectively than those of MEAN-LQM. Furthermore, the WAMMC methods result in almost similar errors to those of their counterpart WAMEMP methods. These results show that the WAM technique can provide more robust performance estimates than the other approaches and is suitable for both bursty as well as non-bursty workloads.

Table 4.9: Accuracy for predicting R_{mean} for non-bursty workloads

Non-Bursty Workloads (22)			
Modeling Approach	ABS_Error (%)	Trend_Error (%)	Max_Error (%)
MEAN-LQM	6.11%	35.05%	24.92%
MEAN-QNM	8.27%	43.51%	22.32%
MBD-LQM	5.53%	35.70%	23.17%
MBD-QNM	9.11%	44.34%	21.01%
WAMEMP-LQM	7.87%	27.76%	15.89%
WAMEMP-QNM	12.65%	38.37%	26.48%
WAMMC-LQM	8.04%	30.24%	15.92%
WAMMC-QNM	13.74%	40.53%	26.73%

WAM captures the complex effects of heavy-tail-like distributions - Figures 4.8 to 4.11 show the probability distribution function (PDF) of number of concurrent sessions for the BPSLZ-HiMix-71-BigDB experiment replications estimated using WAMMC-LQM and MBD-LQM. Figure 4.12 compares the measured R_{mean} values for this workload with those predicted values using the four modeling methods in conjunction with LQM. As discussed in Section 4.2, the R_{mean} values measured for this case varied by up to a factor of 1.63 even though the measured demands and device utilizations were nearly identical for all the replications.

Figures 4.8 to 4.11 reveal that the MBD-LQM method does not capture the differences in measured PDFs among the replications. The PDFs estimated by MBD-LQM are nearly identical for all the replications. In contrast, WAMMC-LQM closely tracks the changes in PDFs. The PDFs estimated through simulation are very close to their counterpart measured PDFs. Consequently, from Figure 4.12, the R_{mean} values predicted by MBD-LQM are nearly the same for all the replications. In contrast, the R_{mean} values predicted by WAMMC-LQM closely track the measured R_{mean} values. From Figure 4.12, MEAN-LQM

also suffers from the same limitation as MBD-LQM and predicts almost the same R_{mean} for all the replications. It is noted that for non-bursty workloads there is less concentration of probability mass towards very large and very small populations. As a result the accuracy obtained with the other methods starts to approach that obtained with WAM.

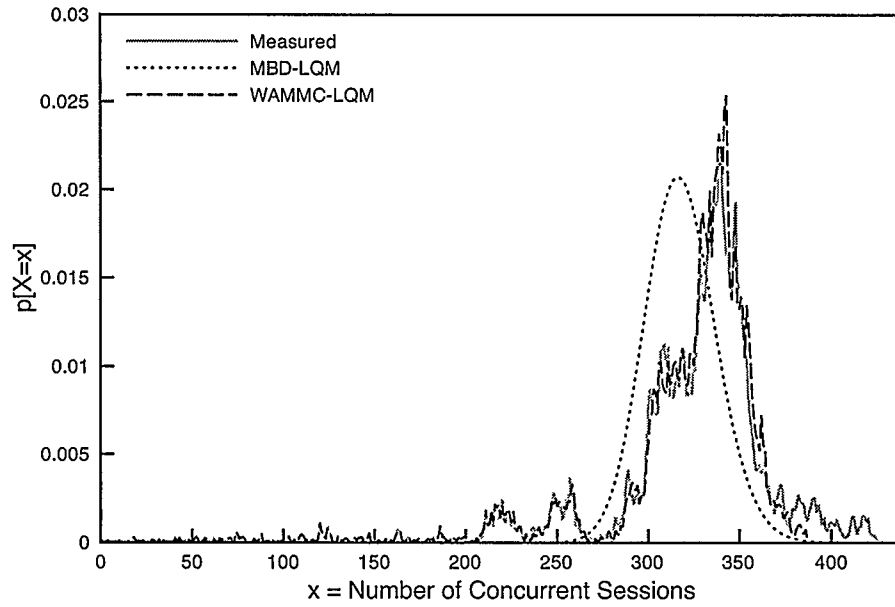


Figure 4.8: Population distributions for BPSLZ-HiMix-BigDB-71 (Replication 1)

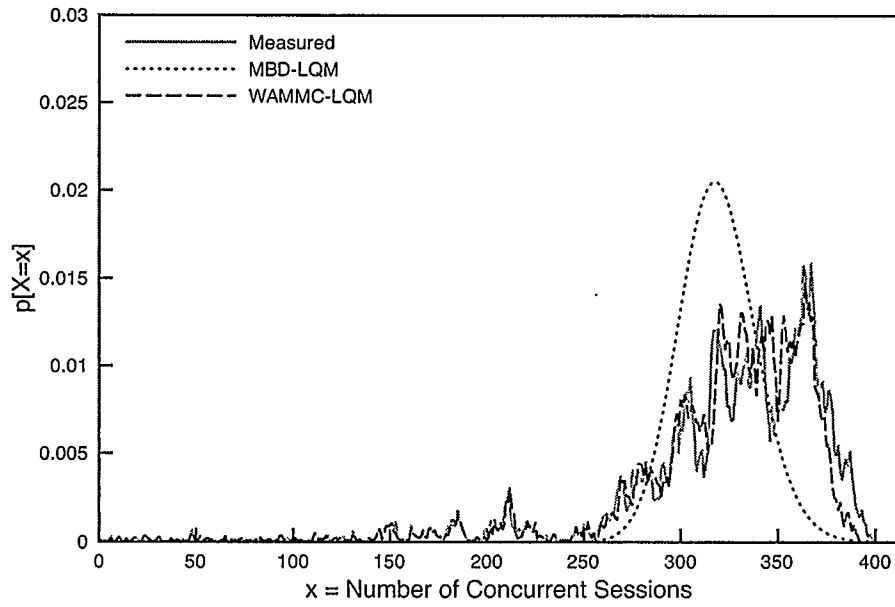


Figure 4.9: Population distributions for BPSLZ-HiMix-BigDB-71 (Replication 2)

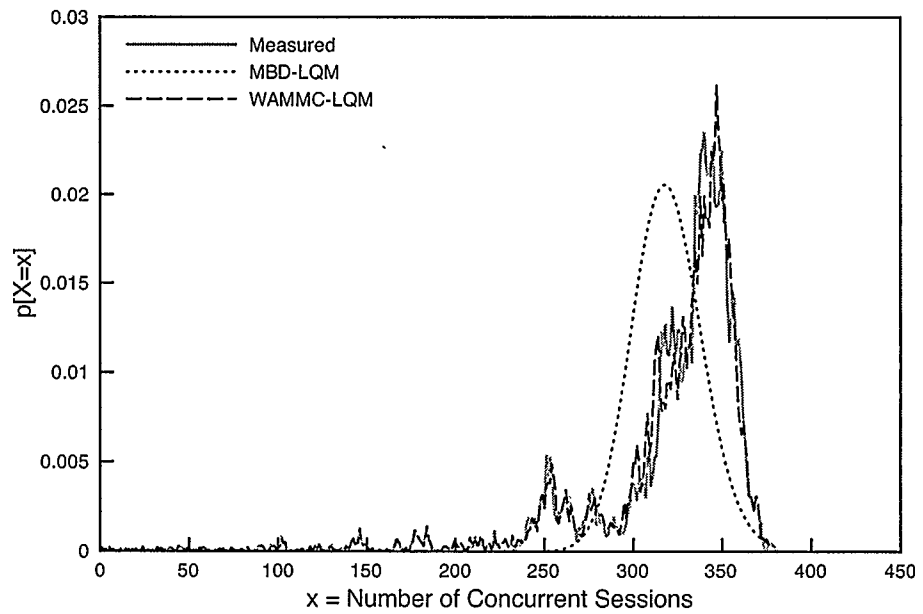


Figure 4.10: Population distributions for BPSLZ-HiMix-BigDB-71 (Replication 3)

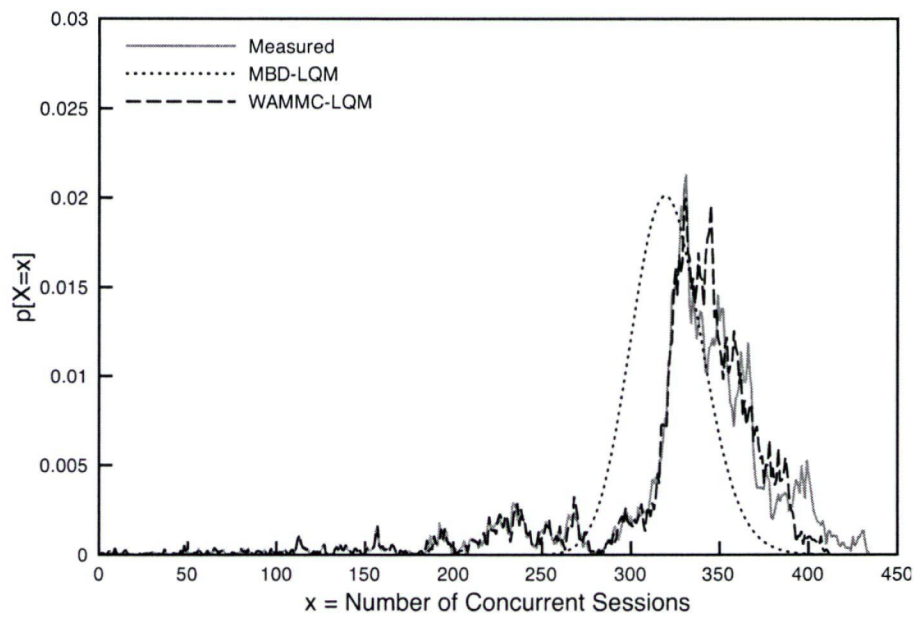


Figure 4.11: Population distributions for BPSLZ-HiMix-BigDB-71 (Replication 4)

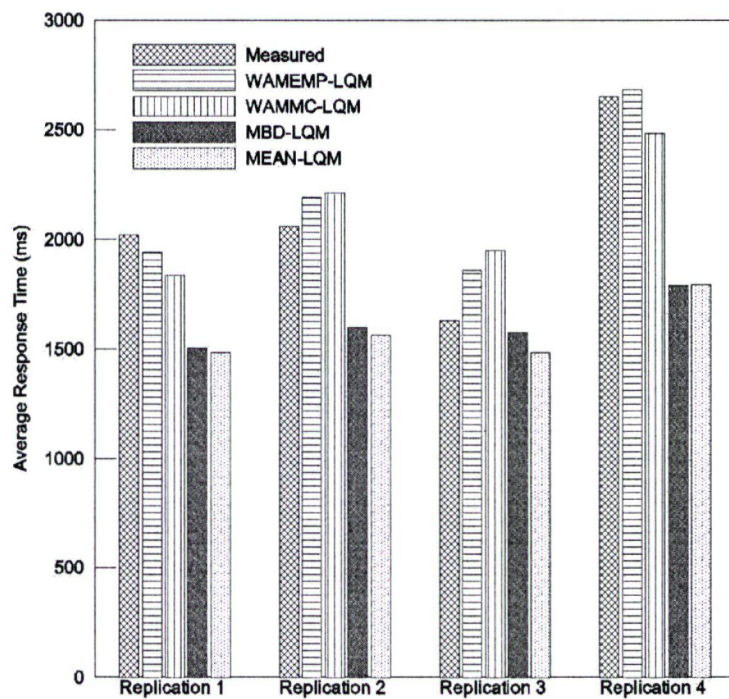


Figure 4.12: Response times for BPSLZ-HiMix-BigDB-71

Another consequence of the ability to accurately estimate the population distribution is that WAM can assess the predictability of performance. As shown in Figure 4.12, WAM is the only method able to capture the variation in measured R_{mean} values for the statistically identical replications of the BPSLZ-HiMix-71-BigDB workload. The results are similar for the other bursty cases, though less pronounced.

Finally, it is shown that WAM is better suited for predicting the impact on system performance of changes in workload characteristics than the other methods. Figure 4.13 plots the mean of measured R_{mean} values over all replications for both the BPSLZ-HiMix-77 and EMPSLZ-HiMix-77 workloads. It also shows the mean of the predicted R_{mean} values over all replications for both workloads while employing the MEAN-LQM, MBD-LQM, WAMEMP-LQM, and WAMMC-LQM methods. From the Figure 4.13, WAM captures the increase in the measured mean R_{mean} that is caused by increased heavy-tail behaviour for session lengths and think times in the BPSLZ-HiMix-77 workload. While the measured increase is approximately 125 ms, the increase predicted by WAMEMP-LQM and WAMMC-LQM is about 101 ms and 89 ms, respectively. In contrast, the MEAN-LQM and MBD-LQM methods do not reflect the impact of the changes and offer almost identical results for both workloads.

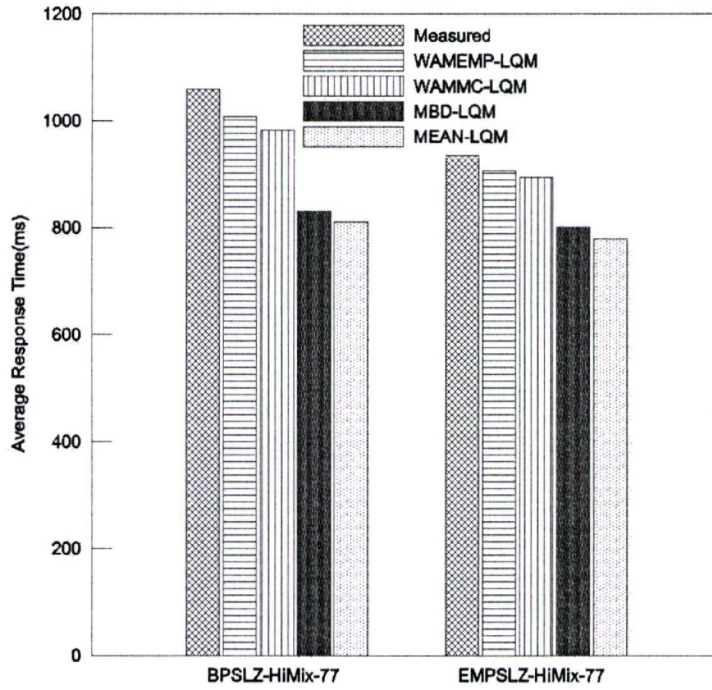


Figure 4.13: Response times for BPSLZ-HiMix-77 and EMPSLZ-HiMix-77

4.5. Conclusions

This chapter used the C-TPC-W system to study the effectiveness of WAM. The results indicate that modeling approaches that only consider the mean number of concurrent customers produce very poor estimates of mean response time for systems with bursty workloads. The average prediction error for bursty workloads is nearly 19.34% and 19.10% for the QNM, and the LQM, respectively. Furthermore, the maximum prediction errors are nearly 43% and 32% for the QNM, and the LQM, respectively. For bursty workloads, using the QNM and LQM models in combination with a Markov birth-death model does not improve the average and the maximum prediction accuracy significantly. In contrast, the WAM approach significantly improves the accuracy of mean response time predictions. For bursty workloads, average prediction accuracy by WAM in

conjunction with the measured population distribution improved by 12% and 11% for LQMs and QNMs, respectively, as compared to the Markov birth-death approach. Moreover, the LQM-based WAM approach had much lower average error and range of errors than the QNM-based WAM approach. Furthermore, WAM also enabled the prediction of very different mean response times reported by multiple statistically identical runs for cases that include heavy-tail-like distributions. In effect, WAM can be used to assess whether a system has unpredictable behaviour by reporting a range of possible behaviours.

The accuracy of WAM's predictions for the system studied is due both to WAM's approach for estimating customer session population distribution and the benefits obtained from using LQMs rather than QNMs. The results presented likely benefit from the relatively large think times between requests. The think times were on the order of 46 seconds with response times typically less than a second. The session population distribution involves estimating the time spent by the system in various states. This in turn depends on the time spent by the various sessions in a workload at the system. It follows that accurate estimates of the population distribution can be obtained if the time spent by a session at the system is estimated accurately. In C-TPC-W since think times are much larger than response times the time spent by a session is dominated by the think times encountered in the session. As a result, accurate population distribution estimates can be obtained in spite of approximating the real request response time using the mean response times obtained from the predictive model. It should be noted that the think

times chosen were realistic since they were based on empirical measurements from a large e-commerce site [38].

The next chapter applies WAM on the H-TPCW system. That differs from C-TPC-W in several important ways. First, it uses the default user-equivalents based workload generator that is part of the TPC-W suite. As a result, the number of concurrent sessions remains constant during an experiment. Secondly, H-TPCW workloads contain requests to the images embedded in the HTML pages returned by the system. Thirdly, H-TPCW is implemented on a different hardware and software platform.

CHAPTER 5: H-TPC-W CASE STUDY

This chapter presents a case study for a multi-tier TPC-W application system which was deployed at HP Labs in Palo Alto, California, USA [10]. As mentioned previously, the H-TPC-W test bed differs from the C-TPC-W test bed in that user-equivalents based workload generation was used. Consequently, the population distribution is characterized in terms of the distribution of number of concurrent requests in the system. Furthermore, the source of burstiness in H-TPC-W's workloads is due to the presence of image requests. Whenever a client receives a HTML response from the system it immediately issues requests for a number of requests for images embedded in that HTML page. Since the arrival of an HTML request is likely to be followed immediately by the arrival of its embedded images the H-TPC-W workloads exhibit burstiness. This case study characterizes such burstiness through the population distribution.

Section 5.1 describes the H-TPC-W test bed. Section 5.2 briefly discusses the experimental results. Section 5.3 discusses the various modeling methods used for H-TPC-W. A QNM and a LQM are developed for H-TPC-W in Section 5.4. Section 5.5 uses these models to assess the effectiveness of WAM. Section 5.6 compares the accuracies of WAM for C-TPC-W and H-TPCW and argues for the use of realistic workload generation techniques in performance tests. Specifically, it describes how employing user-equivalents based workload generation for session-based systems limits the effectiveness of WAM.

5.1. Experiment Test Bed

H-TPC-W was deployed on different software and hardware platforms than the C-TPC-W system. Figure 5.1 shows the experimental setup. The details of the software/hardware used are given in Table 5.1. Figure 5.1 shows that the H-TPC-W system includes three types of nodes, client node, front server node and DB server node, which are connected by a non-blocking Fast Ethernet switch. There are two client nodes. Each *client* node executes a fixed number of user-equivalents during a performance test. A user-equivalent is called as an Emulated Browser (EB) and each EB⁴ mimics a session. A session generates a series of requests with successive requests being separated by a think time. The *front server* node executes the Web/Application server. The Web/Application server processes both the image and non-image requests issued by the EBs. The *DB server* node executes the database server that manages the TPC-W database. All of the three nodes ran on the Linux platform.

As mentioned previously, the system is configured to serve image files. A session first submits a TPC-W interaction (e.g., Home) to the system by establishing a TCP connection. The system dynamically generates a HTML response for this interaction. The session then parses the HTML response to get a list of associated image URLs. In H-TPC-W, the maximum number of images downloaded concurrently is set to four. As a result, the session establishes up to four concurrent TCP connections to download these

images from the front server. The number of images for different types of HTML responses and the sizes of these images are described in the TPC-W specifications. For this case study a *request* is defined to consist of an HTML response and all of its associated images. The request *response time* is defined as the time between receiving the last byte of the last image and issuing the first byte of the corresponding TPC-W interaction.

The number of server processes and the treading levels was not recorded during the measurement experiments. The Web/Application server has a flexible number of server processes that varies with load. However, the actual number of server processes was not monitored during the measurement experiments. The database server has a fixed number of processes that is large compared to the number of EBs. In this case study, a database with 10,000 items and 1,440,000 customers is used.

⁴ The terms EB and session are used interchangeably in this thesis.

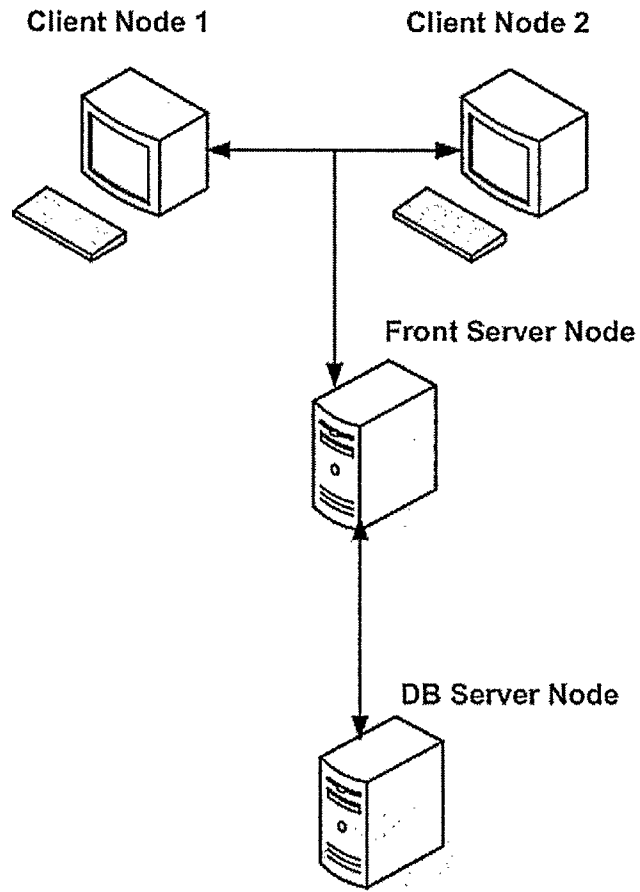


Figure 5.1: H-TPC-W system

Table 5.1: Test bed components

Node	Software	Processor	RAM
Client Node	Emulated Browsers	Pentium III /1 GHz	2 GB
Front Server Node	Apache2.0/Tomcat4.0	Pentium III /1 GHz	3 GB
DB Server Node	MySQL4.0	Pentium III /1 GHz	3 GB

Measurement runs used the standard TPC-W workload generation method with parameters as defined by the TPC-W benchmark. During each run measurements were collected from both the EBs and the server nodes. EBs provided information such as the session identifier, submission time of request, and request response time. However,

finer-grained information on the requests was not recorded. For example, even though the EB logs record the submission and completion time of an entire request they do not contain information on the submission and completion times of individual images making up that request. As explained in Section 5.4, lack of such information influenced the modeling decisions while developing a QNM and LQM for the system. In addition to the log files from the EBs, the SAR [43] Linux utility was used to collect CPU and disk utilizations for each of the nodes at a sampling interval of one second.

As the H-TPC-W uses user-equivalents based workload generation, the number of concurrent sessions in the system always equals the number of EBs. Totally 14 different request types have been defined in the TPC-W benchmark specification. As specified by TPC-W, the think time between successive requests in a session follows an exponential distribution with a mean of 7 seconds.

The TPC-W benchmark defines three types of workload mixes namely, Browsing, Shopping and Ordering. For each mix, the proportions of the total 14 types of requests (i.e., TPC-W interactions) specified by TPC-W [9] can be seen in Table 5.2. In Table 5.2, the average number of images per request for each mix is computed from the collected measurements. The sizes of the images associated with each request type are obtained from the TPC-W specifications and shown in Table 5.2. Finally, the average size of images for each mix is calculated from the measurements collected.

For each workload mix, a set of experiments were run with the number of EBs set to 30, 100, 200, 300, 400, 500 and 600. Each experiment ran for 5 hours. The first 20 minutes

and the last 20 minutes are considered as the warm-up and cool-down periods, and thus omitted in the analysis. The experiments showed that the system was CPU bound. I/O (either at the disk or network) was not found to be significant [10].

Table 5.2: Definition of workload mix

Request Type	Browsing (%)	Shopping (%)	Ordering (%)	Number of Images
Home	29.00	16.00	9.12	11
New Products	11.00	5.00	0.46	9
Best Sellers	11.00	5.00	0.46	9
Product Details	21.00	17.00	12.35	6
Search Request	12.00	20.00	14.53	9
Search Results	11.00	17.00	13.08	9
Shopping Cart	2.00	11.60	13.53	9
Customer Registration	0.82	3.00	12.86	4
Buy Request	0.75	2.60	12.73	3
Buy Confirm	0.69	1.20	10.18	2
Order Inquiry	0.30	0.75	0.25	3
Order Display	0.25	0.66	0.22	2
Admin Request	0.10	0.10	0.12	6
Admin Confirm	0.09	0.09	0.11	5
Average Number of Images (Per Request)	8.5	8.1	6.3	
Average Image Size (Per Image)	4KB	4KB	4KB	

5.2. Overview of Experimental Results

An overview of the measurements for H-TPC-W is discussed first to show derived insights on how the workloads used for the study impact system performance. For a more detailed discussion of the results, the reader is referred to the paper [10] by Zhang *et al.* The following are some key observations from the measurements.

The front server is a bottleneck when the system is processing Shopping and Ordering mixes - Figure 5.2(a) shows that the CPU utilization of the front tier reaches 90-98% when the number of EBs is greater than 500, while Figure 5.2(b) shows the CPU utilization of the database tier is under 40-60% for Shopping and Ordering mixes. As for the Browsing mix, it is not obvious which resource is the bottleneck. When the

number of EBs reaches 400, 500, 600, the utilizations of the front server ranges from 70%-80% while the utilization of the database server also ranges from 70%-80%.

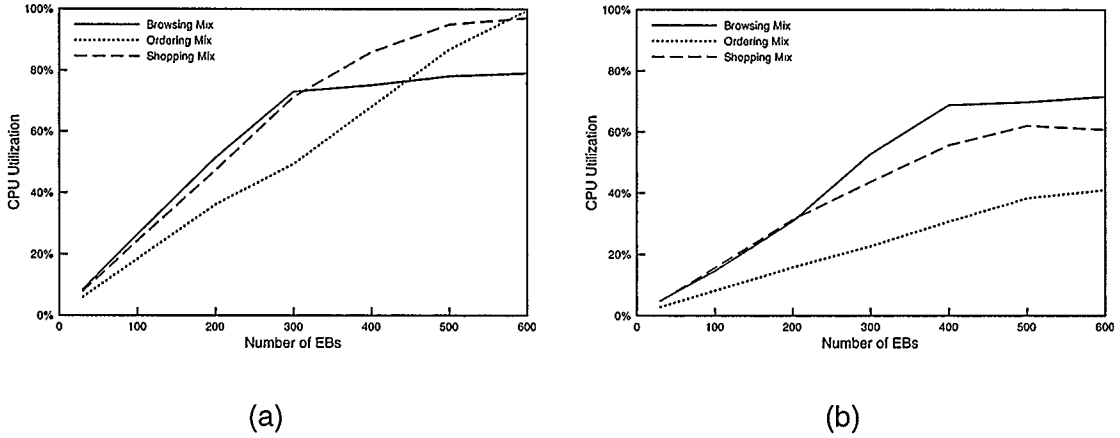


Figure 5.2: CPU Utilization of (a) front server and (b) database server

The system becomes overloaded with 400 EBs, 400 EBs and 500 EBs under the **Browsing mix**, **Shopping mix** and **Ordering mix**, respectively. The collected measured data in Figure 5.3(a) shows that the system throughput flattens at higher number of EBs because of device saturation. For Browsing mix, as shown in Figure 5.3(a), the throughput reaches 40 requests per second and does not increase any more after 400 EBs. For Shopping mix, the plot of the throughput in Figure 5.3(a) shows the maximum can reach 53 requests per second and it flattens out after 400 EBs. For the Ordering mix, the throughput in Figure 5.3(a) seems to increase with the number of EBs. However, the figure also shows that the rate of increase becomes less marked after 500 EBs.

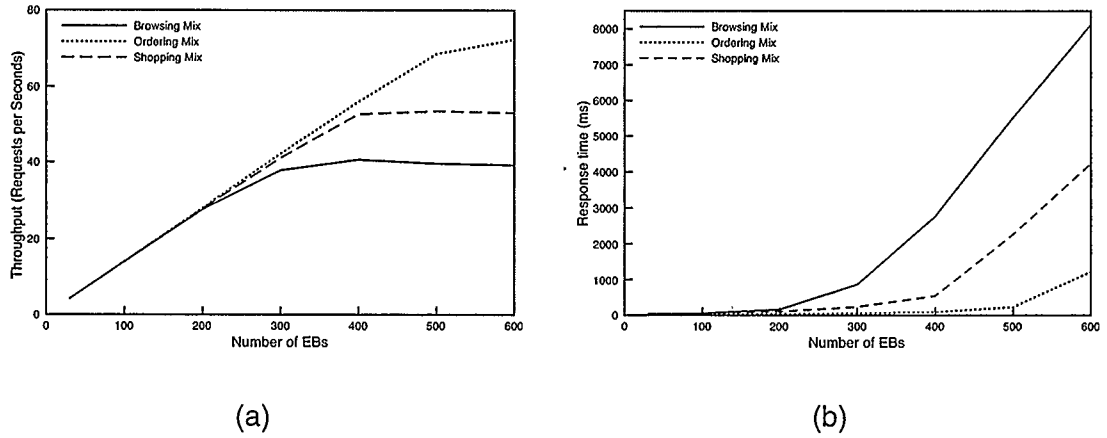


Figure 5.3 : Measured (a) throughput and (b) response times for three mixes

The **Browsing mix** causes poorer performance when compared to the Shopping mix and Ordering mix. The mean response time of the three mixes for various numbers of EBs plotted in Figure 5.3(b). The response time figures show significant nonlinear behaviour. For example, the mean response time values increase from 163 ms for Browsing-200 to 8100 ms for Browsing-600. The tripling of the number of emulated browsers caused mean response time to increase by a factor of 50! Ordering-600 shows similar behaviour with respect to Ordering-500.

5.3. Modeling Approaches Evaluated for H-TPC-W

Since the H-TPC-W system differs from the C-TPC-W system, there are slight differences in the modeling approaches evaluated for this study. Similar to the previous chapter, these methods are used independently with a QNM and a LQM for the H-TPC-W system. The QNM and the LQM for H-TPC-W will be described in Section 5.4. The modeling methods using these models are described in detail as follows:

MEAN - The MEAN method considers only the number of EBs in an experiment. This is identical to the MEAN approach followed in the previous chapter. Since this method considers all sessions including those that are thinking, a think time of 7 seconds is used while solving the QNM and the LQM.

MBD – The main difference between the MBD method used in this chapter to the one used in the previous chapter is the use of the request population distribution instead of the session population distribution. Figure 5.4 shows the STD for the MBD method for H-TPC-W system. As mentioned in Chapter 2, the state of the system is the number of concurrent requests contending for system resources. Since the number of EBs is constant, state dependent birth rates are used in this method. The birth rate λ_k is obtained using the following formula:

$$\lambda_k = (N - k) / Z \quad (5.1)$$

In (5.1) k represents the number of concurrent requests contending for or using system resources, N represents the number of EBs and Z represents the average think time between requests within a EB (in this case 7 seconds). The reason behind the arrival rates can be described as follows. When there are k requests in the system, there are $(N - k)$ EBs are in the thinking state. The $(k + 1)$ state can be reached if any one of these thinking sessions submits a request. Each EB can issue a request every Z seconds and hence the request arrival rate from all these thinking EBs is $(N - k) / Z$. As shown in Figure 5.4, this represents the arrival rate for advancing to the $(k + 1)$ state.

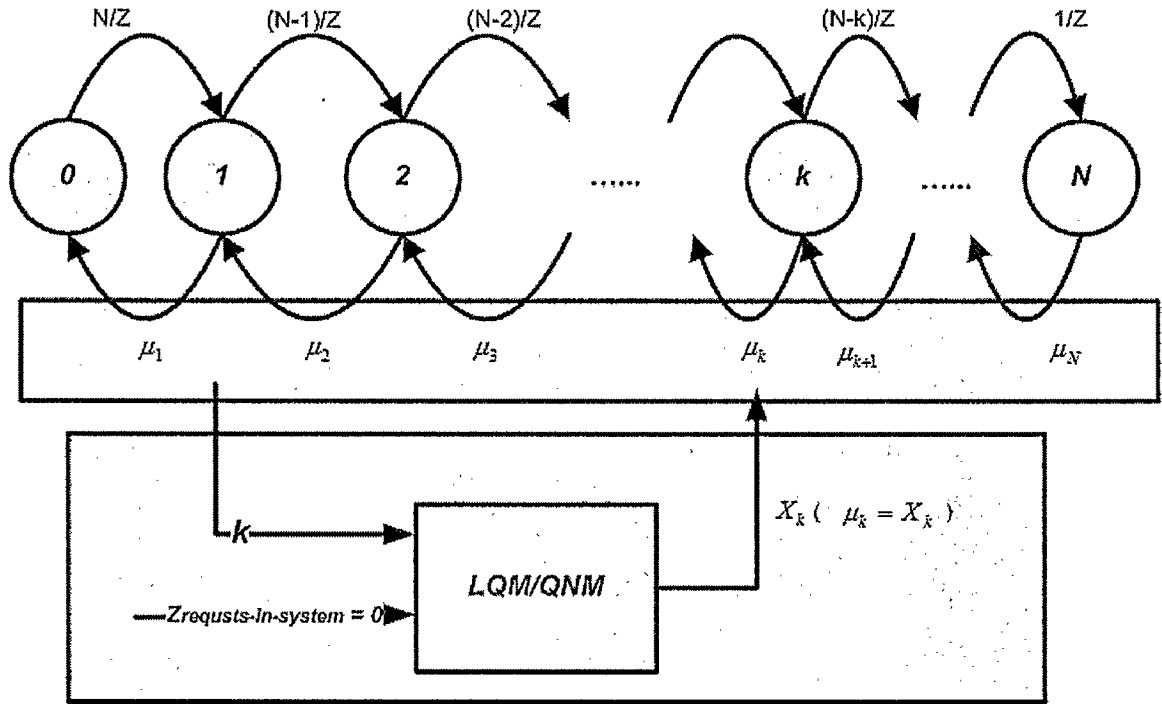


Figure 5.4: STD of hierarchical modeling approach for H-TPC-W system

The death rates μ_k are state dependent and obtained by solving a LQM or a QNM. Specifically, the death rate μ_k is obtained by solving the predictive model for a population of k and a think time of zero. The reason for the zero think time can be explained as follows. As mentioned in Chapter 3, the average number of concurrent requests can be visualized as only those sessions that are either contending for or using the system's resources. It does not include sessions that are in the thinking state. As a result, for this method the QNM and the LQM are solved with a think time of 0 seconds. Thus the fraction of time spent at each population level can be computed as follows,

$$P_0 = \frac{1}{1 + \sum_{k=1}^N \prod_{i=1}^k \frac{\lambda_{(i-1)}}{X_i}} \quad (5.2)$$

$$P_k = P_0 * \prod_{i=1}^k \frac{\lambda_{(i-1)}}{X_i} \quad (5.3)$$

The mean request response time for the system is estimated using Little's law [23] as follows:

$$R_{mean} = \frac{\sum_{i=1}^N P_i X_i R_i}{\sum_{i=1}^N P_i X_i} \quad (5.4)$$

WAMEMP – This method uses the distribution of concurrent requests as measured during an experiment instead of using the closed formulae shown in (5.2) and (5.3).

WAMMC – This method estimates the distribution of number of concurrent requests using Monte Carlo simulations as described in Chapter 3. The effectiveness of the simulations can be characterized by comparing how close the response time predictions obtained with WAMMC are to the corresponding ones obtained with WAMEMP.

5.4. Predictive Performance Models for H-TPC-W

Figure 5.5 and 5.6 show the QNM and LQM, respectively, for H-TPC-W. Similar to the previous case study, the QNM can only capture the impact of contention for hardware resources. The LQM can capture the impact of contention for software resources such as Web, application, and database server threads as well as hardware contention. The QNM and LQM are similar to each other in all other aspects. The mean think time is indicated

as Z in both the models. As discussed in the previous section, the mean think time used in the model differs for different modeling approaches.

Both the QNM and LQM employ two classes of customers to model the impact of image downloads. The HTML class represents the non-image portion of a request. A customer belonging to this class contends with other HTML class customers (i.e., non-image portion of requests emanating from other EBs). She also contends with the image downloads initiated by other EBs in the system. This image portion of a request is represented in the models as a separate IMAGE class. As mentioned previously, both models take as input the number of concurrent requests in the system. The number of HTML class customers is set to be the same as the number of concurrent requests. As mentioned previously, the TPC-W workload generator used in this study opens four concurrent connections to download images. Consequently, the number of IMAGE class customers is set to be four times that of the number of HTML class customers.

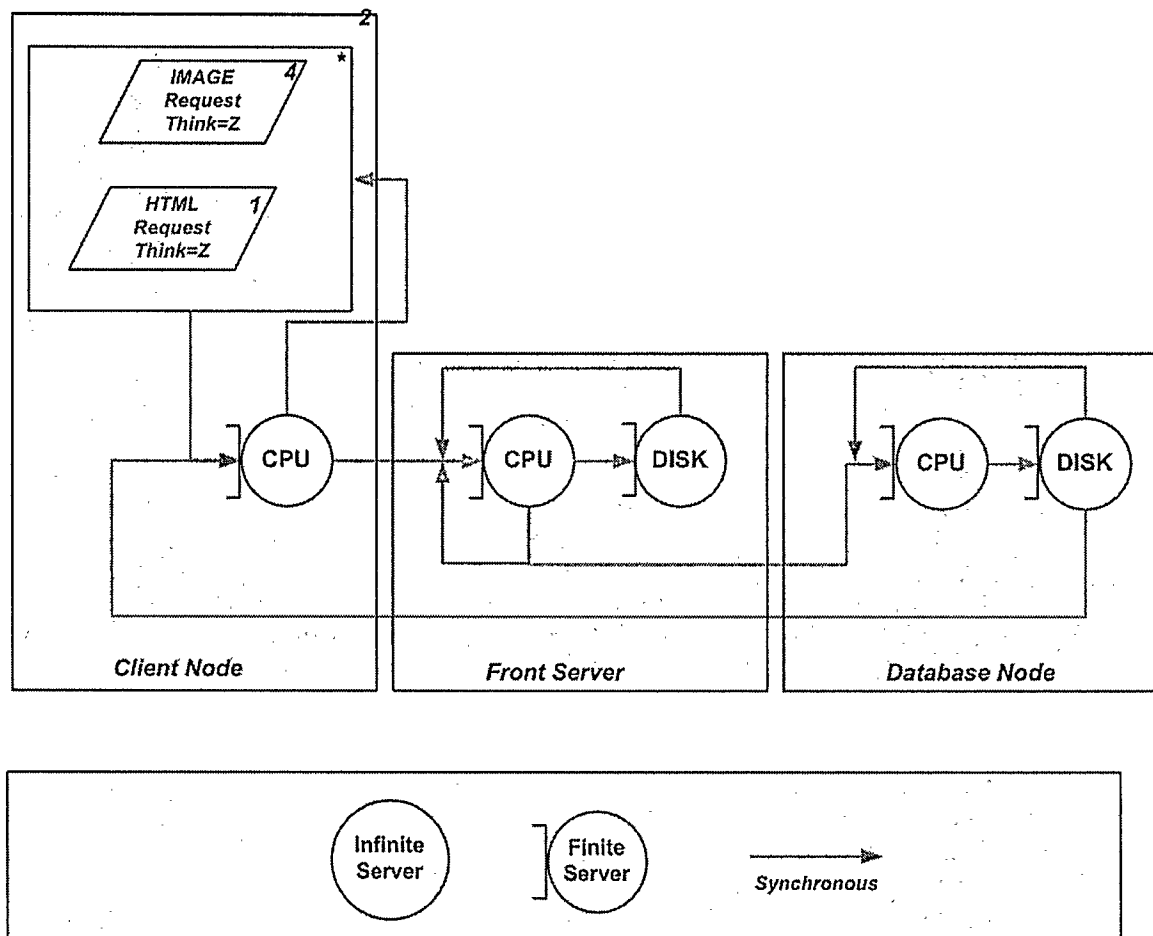


Figure 5.5: QNM for H-TPC-W system

The mean response time for a request is estimated as the mean response time for a HTML class customer. This approach assumes that a request's response time is dominated by the contention (i.e., queuing at resources) it faces with the HTML portions of other requests and the IMAGE portions of other requests. It is not dominated by the request service times at resources. In other words, this approach assumes that time a request spends waiting to get access to a resource is much longer than the time it spends getting service from that resource. The validity of this assumption is discussed shortly.

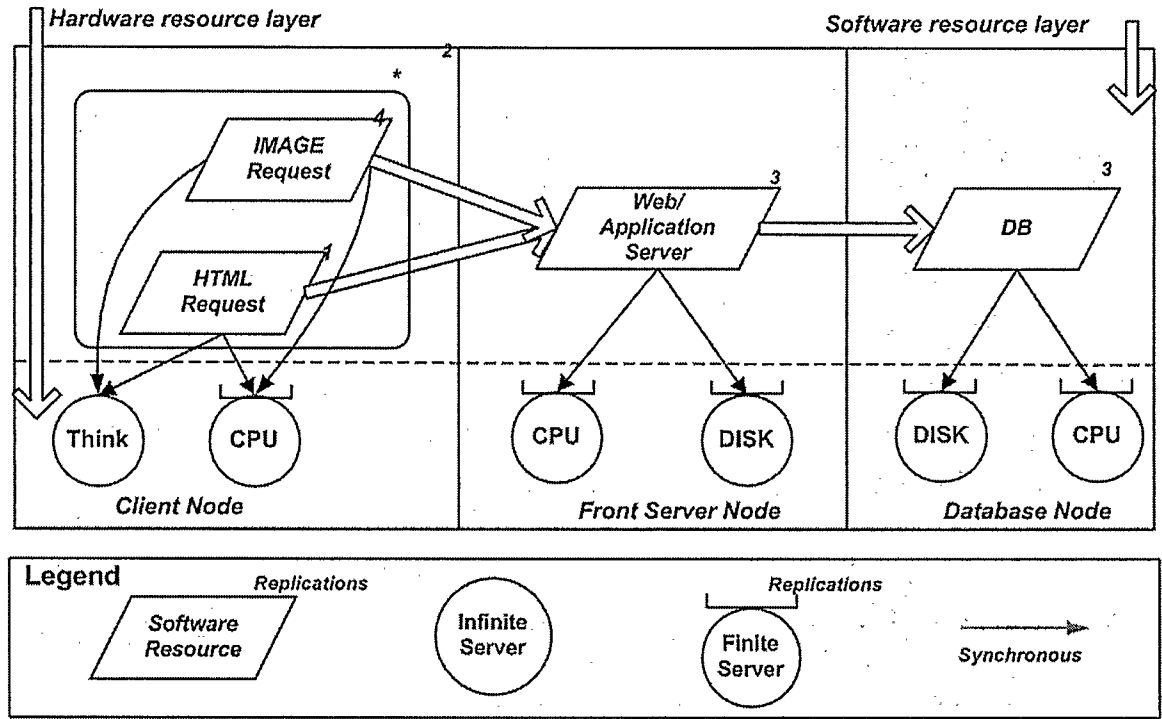


Figure 5.6: LQM for H-TPC-W system

The resource demands for the IMAGE class and HTML class have to be estimated since only the overall utilization of the front server node and database server node by both classes is measured. Since IMAGES exist only on the front server they do not place any demands on the database server's resources. For the front server node the total demand at a resource k denoted by $D_{Web,k}$ is obtained by dividing the node's utilization by the request throughput. This represents the demand for servicing one HTML request and the N_{img} IMAGE requests associated with that request. This study assumes that an IMAGE class request and a HTML class request incur the same demand. In the model, the population of IMAGE class requests is 4 times that of the HTML class. This implies that

each IMAGE class request in this model is actually dealing with $N_{img} / 4$ IMAGE requests.

Consequently, the demands for both classes can be estimated as follows:

$$D_{HTML-Web,k} = \frac{D_{Web,k}}{1 + N_{img}} \quad (5.5)$$

$$D_{IMAGE-Web,k} = \frac{N_{img} * D_{Web,k}}{4 * (1 + N_{img})} \quad (5.6)$$

In (5.5) and (5.6) $D_{HTML-Web,k}$ and $D_{IMAGE-Web,k}$ represent the resource demand for HTML class and IMAGE class, respectively. N_{img} is different for different mixes and can be obtained from Table 5.2. The demand placed by the HTML class on the database server node is obtained by dividing that node's utilization by the request throughput.

As mentioned in Section 5.1, the number of processes for the front server node was not recorded. This parameter was used to calibrate the LQM. A value of 3 for both the Web/Application server and the database server is found to be appropriate. Specifically, for each experiment mean response time predictions are extracted from the LQM for different numbers of concurrent requests. These predictions are compared with the corresponding experimentally observed mean response times for various numbers of concurrent requests. A threading level of 3 for the front server and the database server resulted in a consistently good match between predicted and measured mean response times across different experiments. The LQM validation process suggested that unlike C-TPC-W two phase processing did not seem occurring in H-TPC-W.

The lack of finer-grained measurement data limited the ability to pursue more detailed models for H-TPC-W. Specifically, the EB logs did not record the arrival and completion times of image requests. As a result more advanced features of the LQM capable of modeling the parallel download of images in more detail were not explored. Furthermore, as mentioned previously, it was not possible to measure the resource demands separately for IMAGE requests and HTML requests necessitating the approximations used for estimating the demands of the two classes used in the models. However, the simplifying assumptions did not limit the accuracy of the LQM. As mentioned previously, the LQM provided mean response time estimates that were consistent across the different experiments.

5.5. Evaluation of Modeling Approaches

The prediction of system throughput is based on the prediction results of the average request response time. In this thesis, the average system throughput is calculated as follows using Little's law:

$$X = N_{EB} / (Z + R) \quad (5.7)$$

In (5.7) N_{EB} represents the number of EBs in the system, Z is the average think time and R is the predicted average request response time. In most of the experiments, think time Z is usually much greater than the average response time R . Thus the system throughput is dominated by the known value N_{EB} and Z . However, for those cases where the request response time R is compatible to the think time, the accuracy of the prediction of the R directly impacts the accuracy of the system throughput. Therefore, the prediction

of the average response time can better reflect the effectiveness of the modeling approaches investigated for H-TPC-W system. In this case study, MEAN-QNM had the worst throughput prediction error with an average prediction error of 34%.

Table 5.3 shows the summary of mean response time prediction errors for all the approaches evaluated for the H-TPC-W system. The error metrics are calculated as described in Section 4.4.

Table 5.3: Prediction errors for overall approaches

Approach	ABS_Error (%)	Trend_Error (%)	Max_Error (%)
MEAN-LQM	50.54%	133%	83%
MEAN-QNM	68.93%	61%	95%
MBD-LQM	22.18%	106%	54%
MBD-QNM	53.31%	122%	92%
WAMEMP-LQM	7.22%	21%	18%
WAMEMP-QNM	31.17%	108%	78%
WAMMC-LQM	56.00%	111%	107%
WAMMC-LQM-HiVar	28.57%	82%	84%

5.5.1. MEAN

The results of the MEAN approach are discussed first. As shown in Table 5.3, the MEAN approaches' prediction accuracy is very poor. For all the workloads considered, the MEAN-QNM results in an ABS_Error of 68.93% and a Max_Error of 95%. MEAN-LQM yields better accuracy than MEAN-QNM. The ABS_Error and Max_Error are 50.54% and 83%, respectively. However, these errors are still unacceptably large. The errors are large in spite of using a validated LQM. It can be recalled from the previous section that the LQM yielded good estimates of mean response time for various concurrent request populations. The reason for the large errors is probably because the MEAN approach does not consider the burstiness at the system caused due to the

embedded image requests. Attention is focused next on techniques that consider the distribution of concurrent requests at the system to capture the impact of burstiness.

5.5.2. MBD

MBD, the method that uses the STD of Figure 5.4 to estimate the population distribution is considered next. From Table 5.3, it can be observed that the MBD technique improves `ABS_Error` significantly when compared to MEAN for both LQM and QNM. The `ABS_Error` of MBD-LQM drops to 22%, about 28% improvement compared to MEAN-LQM, and that of MBD-QNM results in nearly 16% improvement in prediction accuracy when compared to MEAN-QNM. However, as shown in Table 5.3, the `Trend_Error` and `Max_Error` are still significantly large for the MBD method.

The population distributions estimated by MBD-LQM significantly diverge from the corresponding empirically measured population distributions. For example, Figure 5.7 plots the population distributions from MBD-LQM for the Browsing mix when the number of EBs is 100, 200, 300 and 400. Figure 5.7 shows that distributions estimated from MBD do not match the empirically measured distributions. At low load, for example 100 EBs, the population distribution output by MBD-LQM seems to match the corresponding empirical distribution (Figure 5.7 (a)). However, when the load increases to 200 EBs, the distributions estimated by MBD is less bursty than the corresponding empirically measured distribution (Figures 5.7 (b)). Specifically, MBD estimates slightly lower probabilities for higher population levels and slightly higher probabilities for lower population levels than what the measurements indicate. When the load increases to 300

and 400 EBs, the system becomes extremely overloaded as discussed in Section 5.2. In this case, as shown in Figure 5.7(c) and 5.7(d), there is a significant discrepancy between the distribution predicted by MBD and the empirically measured population distribution. These results suggest that the MBD method's assumption that the distributions of time spent at various states are exponential may not be valid for H-TPC-W.

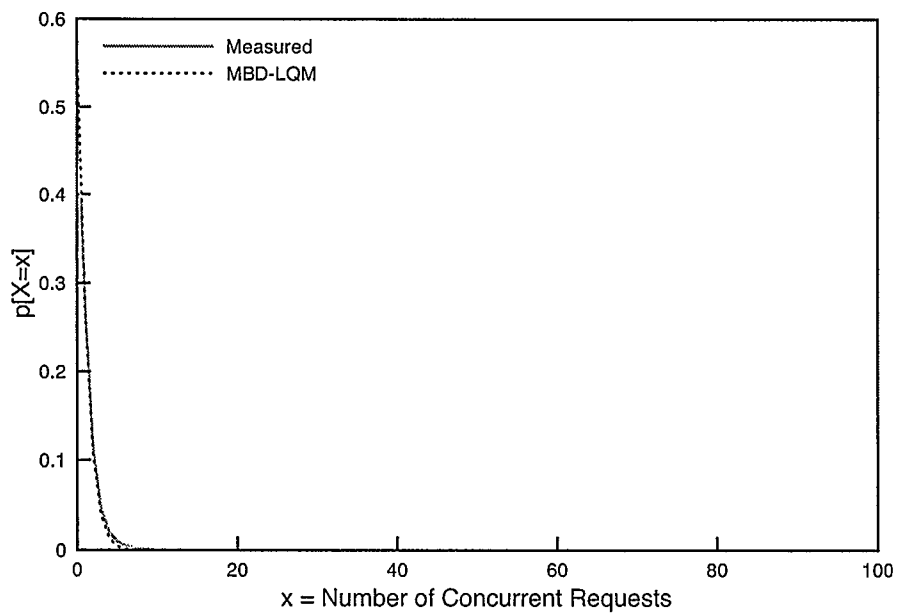


Figure 5.7(a): Population distribution from MBD-LQM for Browsing-100

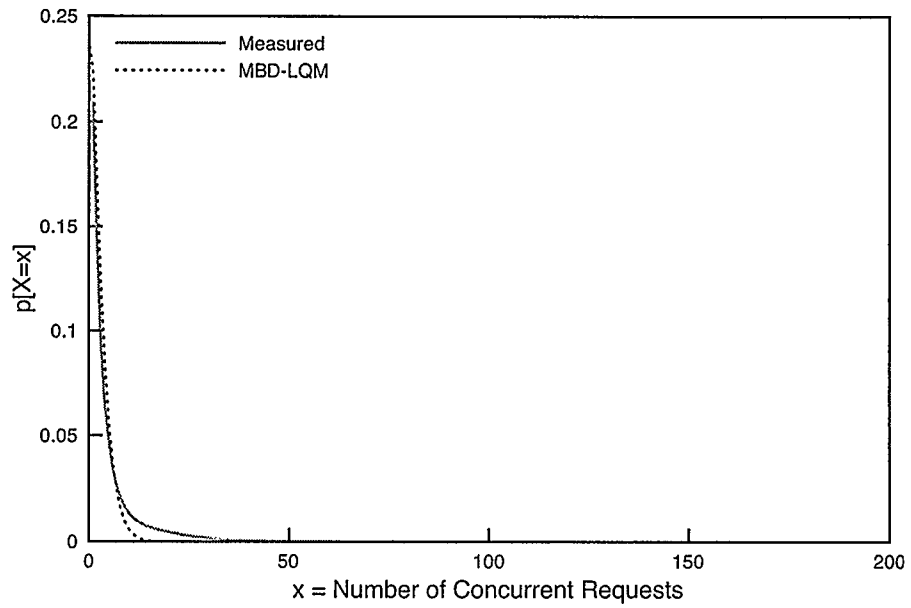


Figure 5.7(b): Population distribution from MBD-LQM for Browsing-200

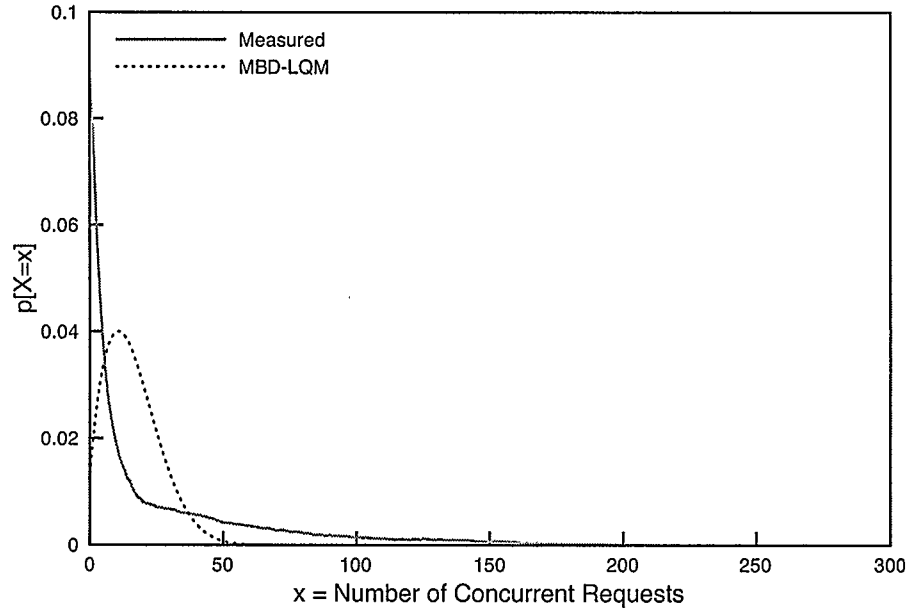


Figure 5.7(c): Population distribution from MBD-LQM for Browsing-300

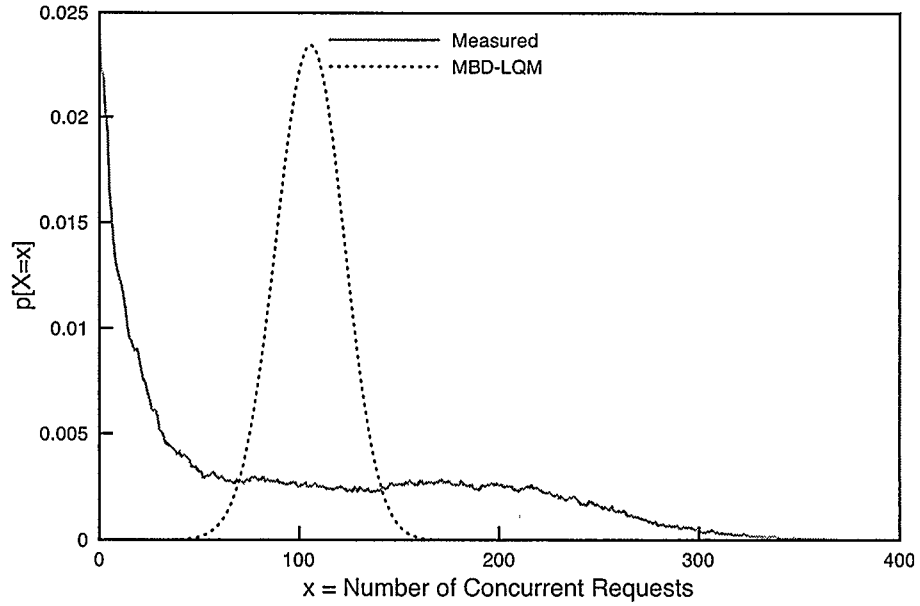


Figure 5.7(d): Population distribution from MBD-LQM for Browsing-400

5.5.3. WAMEMP

WAMEMP, the method which uses the measured population distribution from an experiment, is considered next. From Table 5.3, it can be seen that WAMEMP significantly improves the prediction accuracies of the LQM. Table 5.3 shows the ABS_Error for WAMEMP-LQM is 7.22%, nearly 43% less than that of MEAN-LQM and 15% less than that of MBD-LQM. The gains in Trend_Error and Max_Error are even more significant for WAMEMP-LQM. The Trend_Error drops by nearly 112% and 85% from MEAN-LQM, and MBD-LQM, respectively. The Max_Error drops by nearly 65% and 36% from MEAN-LQM, and MBD-LQM, respectively. These results show that taking into account the population distribution can capture the impact of burstiness and hence improve the accuracy of performance predictions.

WAMEMP also improves the accuracy for QNM when compared to the MEAN and MBD methods. The `ABS_Error` decreases by nearly 38% and 22% when compared to MEAN-QNM and MBD-QNM, respectively. However, since the QNM does not capture software contention, the errors for WAMEMP-QNM are still very large. Due to very inaccurate predictions from QNM, QNM results are ignored for the rest of the discussions.

5.5.4. WAMMC

In this method, Monte Carlo simulation is applied to estimate the distribution of number of concurrent requests as described in Chapter 3. From Table 5.3, the `ABS_Error` is nearly 56%, the `Trend_Error` is nearly 111%, and the `Max_Error` is nearly 107% for this method. This indicates that WAMMC failed to adequately model the H-TPC-W system. The reasons for WAMMC's poor performance are discussed as follows.

Figure 5.8 plots the population distributions estimated by WAMMC for Browsing mix in various numbers of EBs and compares them with the corresponding empirical population distributions. It shows that in a WAMMC estimated distribution the probabilities for higher populations are lower when compared to the corresponding measured distribution. This implies that WAMMC predicts the time spent by the system at higher populations to be much shorter than what is observed in the experiments. The burstiness of the system is not captured properly by WAMMC.

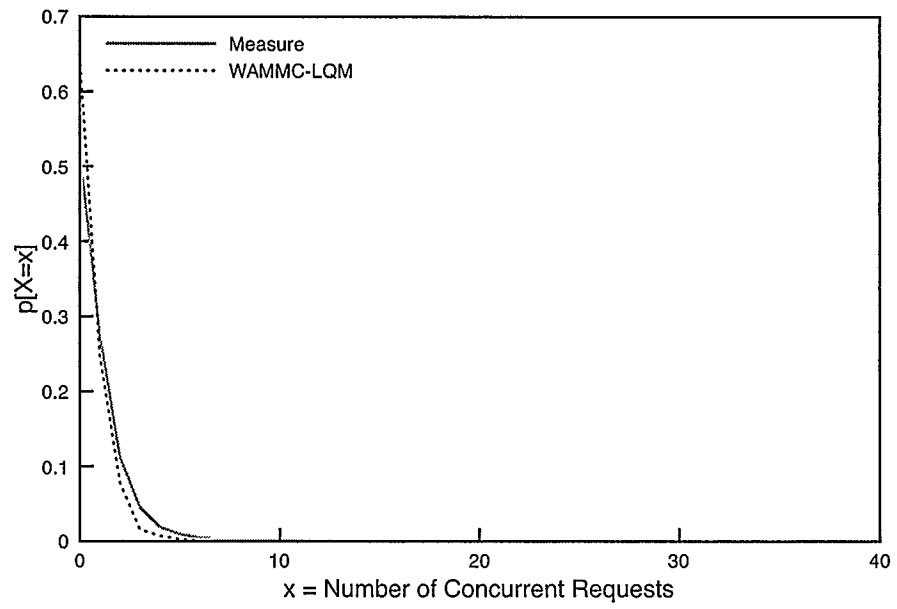


Figure 5.8(a): Population distribution from WAMMC-LQM for Browsing-100

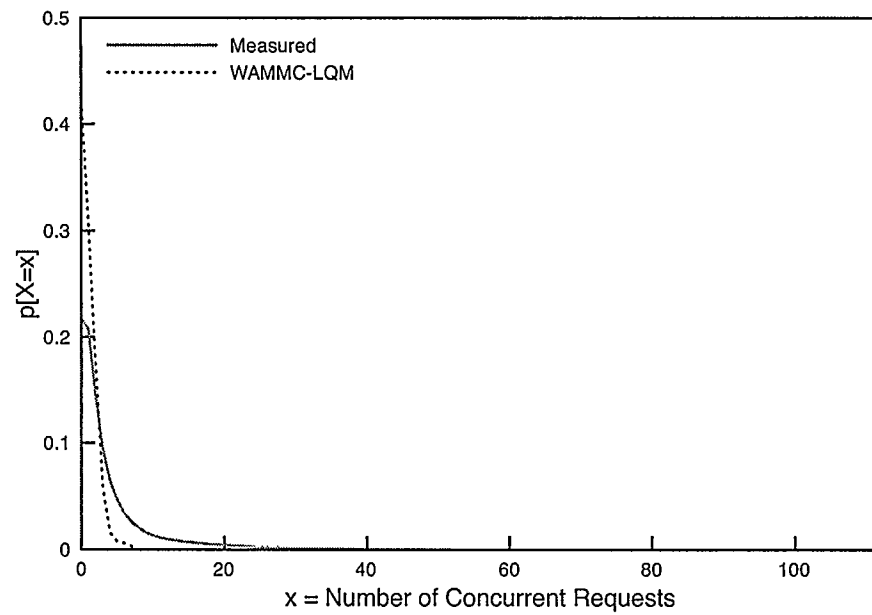


Figure 5.8(b): Population distribution from WAMMC-LQM for Browsing-200

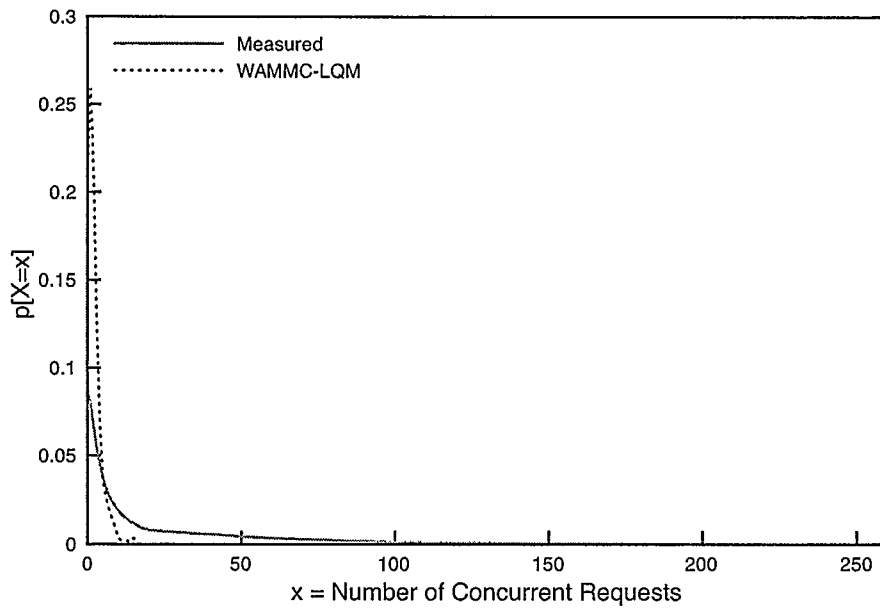


Figure 5.8(c): Population distribution from WAMMC-LQM for Browsing-300

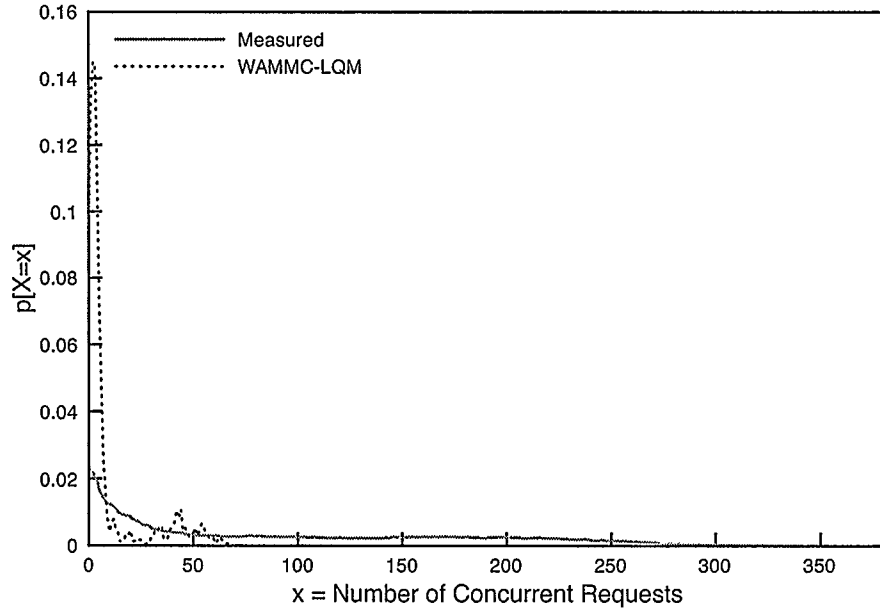


Figure 5.8(d): Population distribution from WAMMC-LQM for Browsing-400

To further investigate the reasons for the poor results with WAMMC, for each experiment the log file generated by the Monte Carlo simulation is compared with the EB log file collected from the measurement. It can be recalled from Chapter 3 that a simulation log file used the mean request response time estimates from the LQM to approximate the real request response times after considering the population changes during the simulation. Analysis of the EB log files showed that the real request response times exhibited considerable variability at each population level. The response times observed for any given population level spanned a very wide range. The simulation logs did not exhibit such a high variability in request response times. This is due to the use of the *mean request response time* prediction to approximate the real request response times. The LQM can only give the mean response time estimate for each population. It cannot capture the variability in response time.

The issue of response time variability is further clarified through an example. Consider the workload Browsing-300. The EB logs show that 35,585 requests occurred when the population level at the system was 5. The measured data shows the response times of these requests range from 2ms to 5675ms. The average response time of these requests is 112ms and the standard deviation is 179ms. The WAMMC method estimates the response time for all these requests as 121ms or around which is the mean response time output by the LQM for Browsing-300 when the population level is 5. Even though the mean response time prediction is pretty close to the actual mean response time at this population, WAMMC ignores the variation in response time at the specified population

level. This is probably one of the main reasons for the simulated population distribution not being as bursty as the empirical distribution.

Due to the less bursty nature of the simulated population distributions, WAMMC's predictions of the mean response times are much lower than the measured data. For example, the predictions from WAMMC-LQM are 84ms, 61ms and 31ms for Browsing-300, Shopping-300 and Ordering-300, respectively. The corresponding actual mean response times measured are 865ms, 235ms and 57ms.

To verify whether capturing the response time distribution will help improve predictions from WAMMC, a new approach called WAMMC-HiVar is devised. This approach uses information about the empirical response time distributions observed for each population level to inject some variability into the response times used during simulation. Specifically, this method maintains arrays of response time values for all population levels observed in a measurement. An array contains all the response time values observed for its corresponding population during an experiment run. Similar to the WAMMC method, WAMMC-HiVar initially obtains a mean response time prediction from the LQM for a request based on the population calculated for the instant at which the request is submitted. However, in contrast to WAMMC, this initial estimate is multiplied by a scale factor to inject some variability. The scale factor is computed as follows. Firstly, as with WAMMC, the population level for the instant at which a request is submitted is determined. Secondly, the response time array corresponding to this population is chosen. Finally, the scale factor is computed by dividing a randomly

chosen response time value from this array by the median of the response time values in the array. The results of this method are discussed as follows.

Table 5.3 shows the prediction results of WAMMC-LQM-HiVar for all cases. For each experiment, simulations are performed with 10 different seeds. The result for each case is the average value from the 10 replications. From Table 5.3, the ABS_Error drops to 28.57% for WAMMC-LQM-HiVar. This represents about 27% improvement when compared to the WAMMC-LQM method. The Max_Error and the Trend_Error for WAMMC-LQM-HiVar also achieve about 23% and 29% improvements when compared to the results for WAMMC-LQM. For example, for Browsing mix, Table 5.4 shows the detailed prediction results when the number of EBs is 100, 200, 300, and 400. The table further confirms that WAMMC-LQM-HiVar's accuracy is significantly better than that of WAMMC-LQM. For example, the predictions of WAMMC-LQM-HiVar for Browsing-100, Browsing-200, Browsing-300 and Browsing-400 are 51ms, 171ms, and 695ms, and 1090ms, respectively. These numbers are higher than their corresponding numbers for WAMMC-LQM of 44ms, 58ms, 84ms and 322ms, respectively. These results confirm that accurate modeling of the response time distribution of the system is necessary for achieving good performance predictions.

Table 5.4: Comparison for WAMMC and WAMMC-HiVar

Comparison of Response Times (ms)			
Cases	Measured	WAMMC-LQM	WAMMC-HiVar-LQM
Browsing-100	62	44	51
Browsing-200	164	58	171
Browsing-300	865	84	695
Browsing-400	2772	322	1090

Figure 5.9 shows further information of the plots of the population distributions for the Browsing mix when the number of EBs is 100, 200, 300 and 400 for WAMMC-LQM-HiVar method. It shows that the WAMMC-LQM-HiVar can achieve more bursty distributions when compared to the WAMMC-LQM method. This leads an improvement in mean response time predictions for Browsing-100, Browsing-200, and Browsing-300 as shown in Table 5.4.

Table 5.4 and Figure 5.9(d) show that WAMMC-LQM-HiVar method has limitations when applied to a very heavily loaded system. Recalling from Section 5.2, for the Browsing mix with 400 EBs the front server node and the database server node CPU utilizations are 75% and 69%, respectively and the system throughput flattens after 400 EBs for Browsing mix. From Table 5.4, with 400 EBs WAMMC-LQM-HiVar predicts a mean response time of 1090 ms when compared to the measured mean response time of 2772 ms. Similarly, as shown in Figure 5.9(d) the population distribution estimated by WAMMC-LQM-HiVar diverges significantly from the measured population distribution. This suggests that mere knowledge of response time distribution may not be adequate especially for heavily loaded systems. It is likely that knowledge about the correlations in the response times sequence (i.e., the response time process) is needed to accurately predict the behaviour of such systems.

It should be noted that WAMMC-LQM-HiVar cannot be used as a constructive method. This is because it requires detailed knowledge of the response time distributions for each population level and possibly even the correlations in the response time sequence. Such information is not likely to be available in practice to a performance analyst.

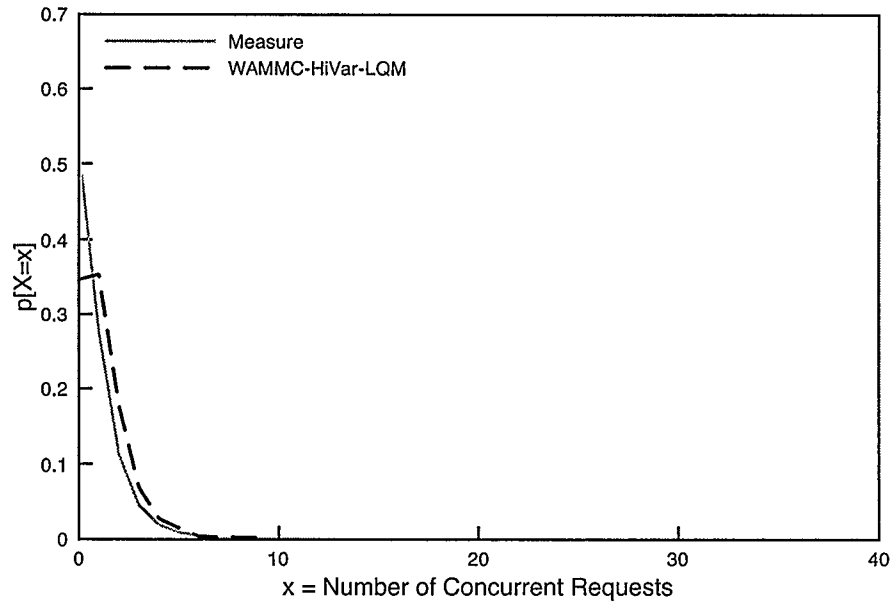


Figure 5.9(a): Population distribution from WAMMC-HiVar-LQM for Browsing-100

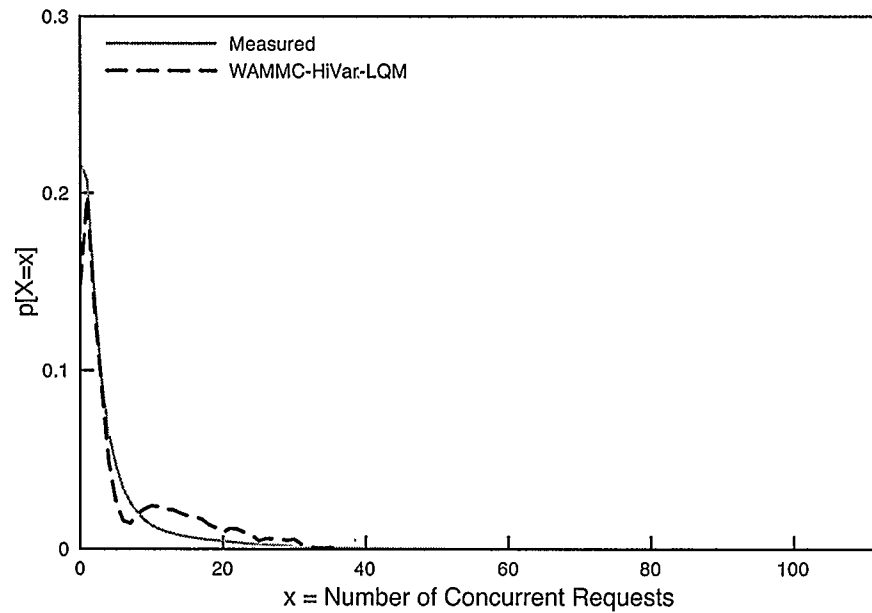


Figure 5.9(b): Population distribution from WAMMC-HiVar-LQM for Browsing-200

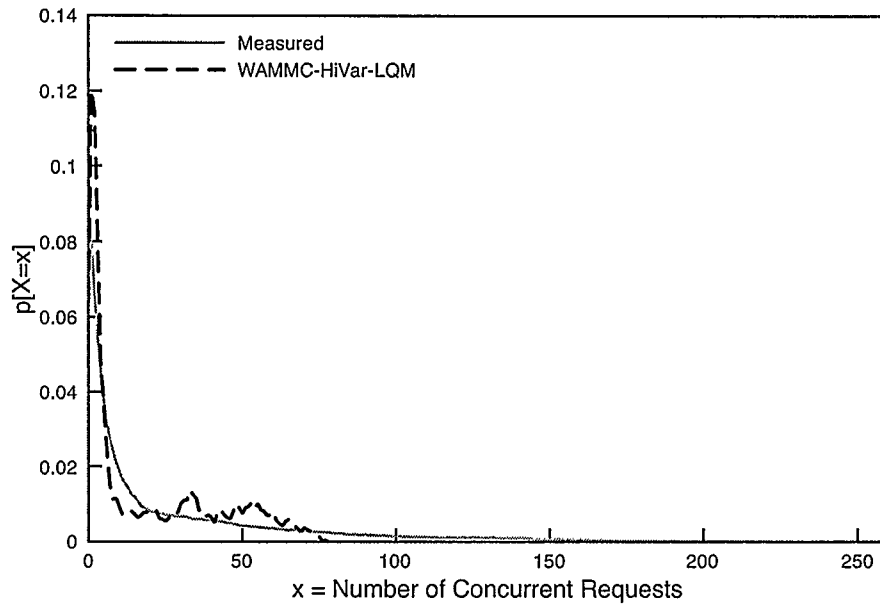


Figure 5.9(c): Population distribution from WAMMC-HiVar-LQM for Browsing-300

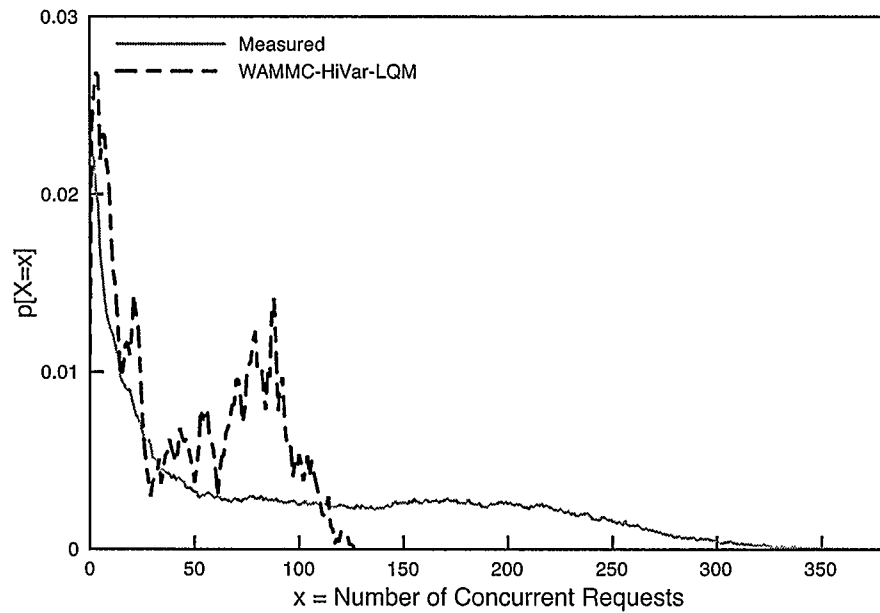


Figure 5.9(d): Population distribution from WAMMC-HiVar-LQM for Browsing-400

It can be recalled that the WAMMC method provided accurate response time predictions for the C-TPC-W system. This was *in spite* of using just the mean response time estimates from the LQM for estimating request response times and not considering the detailed per population response time distributions. The reasons for the differences in behaviour of WAMMC for C-TPC-W and H-TPC-W are discussed in the next section.

5.6. Comparison of WAM for C-TPC-W and H-TPC-W

The reason for the success of WAMMC for C-TPC-W is discussed first. For C-TPC-W, WAMMC was used to predict the distribution of number of concurrent sessions. Accurately predicting this distribution involves accurately predicting the time spent at various states in the system. To predict the time spent at various states in the system well, one needs to have good predictions for the *times spent by sessions* in the system and *the session arrival process*. The session arrival process (i.e., sequence of session inter-arrival times) is in general specified as an input in performance studies and is hence, as was the case with C-TPC-W, known. The time spent by a session depends on the session length, the session think times, and the response times for the session's requests. For C-TPC-W, a session's duration is largely determined by its think times. As discussed in Chapter 4, this is because the think times are much larger than the response times. As a result, session durations and hence the population distribution can be accurately determined if inputs used in the performance evaluation process such as the sequences of session inter-arrival times, think times, and session lengths are known. There is very little dependency with the response time distribution or the sequence of request response times which are

both unknown. The ability to accurately predict the population distribution ensures the accuracy of mean response time predictions.

Due to the different definition of a state, to predict the population distribution for H-TPC-W one must know *the request arrival process* and the *times spent by requests* in the system. Both these quantities are typically not known to a performance analyst due to the following two properties. Firstly, the sequence of times spent by requests (i.e., sequence of request response times) in a session is unknown. Analytic models such as LQMs can only estimate the mean response time for a system under steady state conditions. As mentioned in the previous section, they cannot estimate the *response time sequence* for a workload. Secondly, the time at which a request from a session arrives at a system depends on the response time incurred by the previous request in the session. Since the response time for the previous request is unknown, the request arrival process is also unknown. In essence, these dependencies with system response times make it very difficult to predict the population distribution in a constructive way as was possible for C-TPC-W.

These results motivate the need for a workload generation method for session-based systems that is more realistic than the widely used user-equivalents approach. The inability to predict the distribution of number of concurrent requests adversely impacts the ability to predict mean response time for bursty workloads. This is in spite of having a LQM that provided good per request population estimates of mean response times. This thesis argues that a mixed workload generation approach such as the one employed for C-TPC-W is more realistic for session-based systems and enables accurate mean

response time predictions for bursty workloads. It should also be noted that many measurement studies of session-based systems use the mean think time of 7 seconds specified by TPC-W. This value is much lower than the mean think times observed in real session-based systems. Based on insights derived from this research such unrealistically low think times can not only impact the representativeness of performance tests but also adversely impact the ability to predict mean response time for bursty workloads.

5.7. Conclusions

This chapter assessed the effectiveness of WAM for H-TPC-W. As with C-TPC-W, the results indicate that methods that incorporate the population distribution can improve the accuracy of predictive models. However, the study showed that it was not possible to constructively estimate the population distribution for H-TPC-W. This stems from the need to estimate the population distribution at the granularity of a request due to the use of user-equivalent's based workload generation. The distribution of number of concurrent requests is significantly dependent on system response times. Since detailed information about system response times (e.g., the response time distribution or the response process) are typically unknown, the request response time distribution cannot be simulated accurately. In contrast, with mixed workload generation the population distribution has to be characterized at the granularity of sessions. For realistic systems where session think times are much larger than system response times, the population distribution has very little dependency with system response times. It can be accurately constructed based solely on inputs to the performance evaluation process such as the

sequences of session inter-arrival times, session think times, and session lengths. In light of these results, this thesis suggests that the widely used practice of user-equivalents based workload generation complicates the ability to accurately predict the performance of systems characterized by burstiness.

CHAPTER 6: SUMMARY AND CONCLUSIONS

In this thesis, a new technique called the Weighted Average Method (WAM) is introduced for improving the accuracy of predictive models for systems with bursty request arrivals. The technique is appropriate for session-based systems such as e-commerce systems and enterprise application systems. Others have shown that real session based systems exhibit such bursty behaviours so sizing, capacity planning, and on-going management exercises should benefit from WAM.

The technique was motivated by a well-known hierarchical method that combines a Markov birth-death process and QNMs. The general approach is applied but the closed expression for estimating population distribution is replaced with a fast Monte Carlo simulation technique that arbitrary distributions that affect burstiness for request arrivals can be taken into account. Furthermore, both QNMs and LQMs are considered in this thesis. Measurements from two TPC-W systems allow to compare the effectiveness of all these methods at predicting the mean request response time.

The results indicate that modeling approaches that only consider the mean number of concurrent customers produce very poor estimates of mean response time for systems with bursty workloads. For the C-TPC-W system, the average mean response time prediction error for bursty workloads is nearly 19.34% and 19.10% for the QNM, and the LQM, respectively. Furthermore, the maximum mean response time prediction errors for bursty workloads are nearly 43% and 32% for the QNM, and the LQM, respectively. For the H-TPC-W system, the average prediction errors reach 69% and 51% for QNM and LQM, respectively. The results also indicate that LQMs are better than QNMs since they

take into account contention for software resources and software request-reply relationships.

The use of population distributions significantly improved response time predictions especially for bursty workloads. Population distributions that were observed during the performance tests were used in conjunction with QNMs and LQMs developed for the systems. For the C-TPC-W system, the average mean response time prediction accuracy was 5% for bursty workloads when the system's LQM was used in combination with the measured population distributions. This represents an improvement of 14% over the straightforward application of LQM. For the H-TPC-W system the use of the empirically measured population distributions in combination with the LQM for the system reduced the average prediction error from 50% to 7%. These results motivate the need for a technique that can estimate the population distribution given arbitrary characterizations for workload parameters that impact the distribution.

WAM is designed to support constructive estimation of the population distribution for any given workload. This will permit analysts to explore how arbitrary distributions for workload parameters that influence burstiness impact performance. The effectiveness of the constructive capability of WAM was investigated for C-TPC-W and H-TPC-W.

For C-TPC-W, WAM was able to accurately estimate the population distribution for any given workload. As a result the performance predictions made using the estimated distributions were accurate. This result suggests that accurately estimating the population distribution given distributions for workload attributes that impact burstiness is feasible

for this type of a system. This was exploited to explore the impact of changes to workload parameters on the customer population distributions and hence on system behaviour. Furthermore, the constructive capability enabled the prediction of very different mean response times reported by multiple statistically identical runs for cases that include heavy-tail-like distributions. In effect, WAM can be used to assess whether a system has unpredictable behaviour by reporting a range of possible behaviours.

In contrast to C-TPC-W, WAM's population distribution estimates were very poor for H-TPC-W. Consequently, performance predictions using these estimated distributions were not accurate. Since H-TPC-W was used as a closed system, burstiness had to be modeled through the distribution of number of concurrent requests. In contrast to the distribution of number of concurrent sessions, the distribution of number of concurrent requests has a significant dependency with system response times. As a result, since system response times are typically unknown, it is very difficult to estimate the distribution of number of concurrent requests. The thesis concludes that constructive characterization of how various workload characteristics impact burstiness and performance is difficult for closed systems.

The results from the case studies suggest that the use of a realistic performance testing approach such as the one used in C-TPC-W that lets the number of concurrent sessions vary during an experiment can make performance prediction easier for session-based systems characterized by burstiness. This is due to the fact that for such systems burstiness can be characterized at the granularity of a session. With realistically large think times, the distribution of number of concurrent sessions has very little dependency

with system response times. It can be determined accurately based only on the inputs of the performance evaluation process such as the distributions of session inter-arrival time, session think times, and session length. The ability to accurately estimate the distribution of number of concurrent sessions *enables* modeling of the impact of bursty request arrivals in session-based systems. In contrast, the widely used user-equivalents based testing approach complicates modeling by forcing burstiness to be characterized at the granularity of a request.

A significant number of experiment hours were spent exploring two different systems, multiple application settings (for C-TPC-W) and workloads to better establish the generality of the proposed approach. For example, the various C-TPC-W application settings allowed to realize very different system behaviours in terms of the relative demands placed on the system resources. Based on the experiences from this work, three conditions have been identified that are essential for WAM to yield accurate predictions for a system. Firstly, the system being modeled by WAM must exhibit variation in the number of concurrent sessions. Secondly, WAM requires a good predictive model for the system under study. The model should provide good per population performance estimates for the system. Finally, as mentioned previously, the think times in the system must be larger than the response times.

Future work includes extending the technique to consider multi-class models and load dependent service rates for session-based systems. Specifically, techniques will be developed to ensure the efficiency of WAM for multi-class models. Support for load dependent service rates is important for modeling Software as a Service environments

where the amount of resources allocated to applications can fluctuate dynamically based on load. Future work will also apply and validate WAM for other multi-tier software systems, including enterprise application systems.

Two recent studies [28] [44] have proposed MAP-based analytic techniques that can yield approximate mean response time estimates for systems characterized by burstiness. These techniques were publicized after the conclusion of the research presented in this thesis. Comparing these techniques with WAM would be another topic that deserves future investigations.

REFERENCES

- [1]. W. Eckerson, "Three tier client/server architecture: achieving scalability, performance, and efficiency in client server applications," *Open Information Systems*, vol. 10, no. 1, pp. 1-12, Jan. 1995.
- [2]. J. P. Buzen, "Computation algorithms for closed queuing networks with exponential servers," *Communications of the ACM*, vol. 16, no. 9, pp. 527-531, Sept. 1973.
- [3]. E. D. Lazowska, G. Scott Graham, K. Sevcik, J. Zahorjan, *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*. Prentice Hall, 1984.
- [4]. U. Vallamsetty, K. Kant, and P. Mohapatra, "Characterization of e-commerce traffic," *Electronic Commerce Research*, vol. 3, no. 1-2, pp. 167-192, Jan. 2003.
- [5]. D. Menasce, V. Almeida, R. Reidi, F. Pelegri-nelli, R. Fonesca, and W. Meira Jr., "In search of invariants in e-business workloads," in *ACM Conference on Electronic Commerce*, pp. 56-65, Oct. 2000.
- [6]. D. Krishnamurthy, J. Rolia, and S. Majumdar, "A synthetic workload generation technique for stress testing session based systems," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 868-882, Dec. 2006.
- [7]. S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 295-310, May 2004.
- [8]. M. Woodside, J. E. Nielsen, D. C. Petriu, and S. Majumdar, "The stochastic rendezvous network model for performance of synchronous client-server-like distributed software," *IEEE Transactions on Computers*, vol. 44, no.1, pp. 20-34, Jan. 1995.
- [9]. Transaction Processing Performance Council, "TPC benchmark W (Web commerce) specification," Feb. 2002, http://www.tpc.org/tpcw/spec/tpcw_V1.8.pdf.
- [10]. Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *4th International Conference on Autonomic Computing*, 2007, pp. 27-27.
- [11]. J. Rolia, K. Secvik, "The Method of Layers", *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 689-700, Aug.1995.
- [12]. P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proceedings of ACM SIGMETRICS Conference*, 1998, pp. 151-160.
- [13]. K. Kant, V. Tewary, and R. Iyer, "Geist: A generator for e-Commerce and internet server traffic," in *Proceedings of 2001 IEEE Int'l Symp: Performance Analysis of Systems and Software (ISPASS 01)*, 2001, pp. 49-56.
- [14]. D. Menasce, V. Almeida, R. Fonesca, and M. Mendes, "A methodology for workload characterization of e-commerce sites," in *Proceedings of ACM Conference on Electronic Commerce*, 1999, pp. 119-128.

- [15]. G. Banga and P. Druschel, "Measuring the capacity of a web server under realistic loads," *World Wide Web*, vol. 2, no. 1, pp. 69-83, May 1999.
- [16]. D. Mosberger and T. Jin, "httpperf: A tool for measuring web server performance," in *Proceedings of the Workshop on Internet Server Performance*, 1998, pp. 59-67.
- [17]. W. Mar, "LoadRunner architecture," <http://www.wilsonmar.com/loadrun.htm>.
- [18]. W. Mar, "WinRunner.info", <http://www.wilsonmar.com/winrun.htm>.
- [19]. T. Bezenek *et al.*, "Java TPC-W Implementation Distribution," June 2003, <http://www.ece.wisc.edu/~pharm/tpcw.shtml>.
- [20]. Rice University, "TPC-W," <http://www.cs.rice.edu/CS/Systems/DynaServer/TPC-W/>.
- [21]. C. Stewart, T. Kelly, and A. Zhang, "Exploiting nonstationarity for performance prediction," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 31-44, June 2007.
- [22]. D. Menasce and V. Almeida, *Capacity Planning for Web Services: Metrics, Models and Methods*, Prentice Hall Inc., 2001.
- [23]. R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley & Sons Inc., 1991.
- [24]. D. Menasce and V. Almeida, *Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems*, Prentice Hall, 1994.
- [25]. D. Menasce and M. Bennani, "Analytic performance models for single class and multiple class multithreaded software servers," in *Proceedings of the International Computer Measurement Group (CMG) Conference*, 2006, pp. 475-482.
- [26]. A. B. Bondi and W. Whitt. "The influence of service-time variability in a closed network of queues," *Performance Evaluation*, vol. 6, no. 3, pp. 219-234, Sep. 1986.
- [27]. D. Eager, D. Sorin, and M. Vernon, "AMVA Techniques for high service time variability," in *Proceedings of ACM SIGMETRICS Conference*, 2000, pp. 217-228.
- [28]. G. Casale, N. Mi, and E. Smirni. "Bound analysis of closed queuing networks with workload burstiness," in *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2008, pp. 13-24.
- [29]. W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1-15, Feb. 1994.
- [30]. N. Markovich, *Nonparametric Analysis of Univariate Heavy-Tailed Data: Research and Practice (Wiley Series in Probability and Statistics)*, Wiley-Interscience, 2007.
- [31]. V. Almeida, M. Arlitt, and J. Rolia, "Analyzing a web-based system's performance measures at multiple time scales," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 2, pp. 3-9, Sept. 2002.
- [32]. M. E. Crovella and L. Lipsky, "Long-lasting transient conditions in simulations with heavy-tailed workloads," in *Proceedings of the Winter Simulation Conference*, 1997, pp. 1005 - 1012.
- [33]. L. Kleinrock, *Queueing System Volume 1: Theory*, John Wiley & Sons Inc., 1975.

- [34]. K. Psounis, P. Molinero-Fernández, B. Prabhakar, and F. Papadopoulos, "Systems with multiple servers under heavy-tailed workloads," *Performance Evaluation*, vol. 62, no. 1-4, pp. 456-474, Oct. 2005.
- [35]. M. Andersson, J. Cao, M. Kihl, and C. Nyberg, "Performance modeling of an apache Web server with bursty arrival traffic," in *Proceedings of the International Conference on Internet Computing*, 2003, pp. 508-514.
- [36]. N. Mi, Q. Zhang, A. Riska, E. Smirni, and E. Riedel, "Performance impacts of autocorrelated flows in multi-tiered systems", *Performance Evaluation*, vol. 64, no. 9-12, pp. 1082-1101, Oct. 2007.
- [37]. B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: A cautionary tale", in *Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation*, 2006, pp. 239-252.
- [38]. M. Arlitt, D. Krishnamurthy, and J. Rolia, "Characterizing the scalability of a large web-based shopping system," *ACM Transactions on Internet Technology*, vol. 1, no. 1, pp. 44-69, Aug. 2001.
- [39]. V. Paxson and S. Floyd, "Wide area traffic: The failure of Poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226-244, June 1995.
- [40]. M. Harchol-Balter, M. Crovella, and C. Murta. "On choosing a task assignment policy for a distributed server system," *Parallel and Distributed Computing*, vol. 59, no. 2, pp. 204-228, Nov. 1999.
- [41]. M. Taqqu, V. Teverovsky, and W. Willinger, "Estimators for long-range dependence: an empirical study," *Fractals*, vol. 3, no. 4, pp. 785-798, 1995.
- [42]. K. Park, G.T. Kim, and M. Crovella, "On the relationship between file sizes, transport protocols, and self-similar network traffic," *Proceedings of the International Conference of Network Protocols*, pp. 171-180, Oct. 1996.
- [43]. B. Calkins, "Solaris 9 System Monitoring and Tuning", Dec. 2002, <http://www.informit.com/articles/article.aspx?p=30362&seqNum=6>.
- [44]. A. Horváth, G. Horváth, and M. Telek, "A joint moments based analysis of networks of map/map/1 queues," In *Proceedings of 5th International Conference on the Quantitative Evaluation of Systems (QEST)*, Sept. 2008.