NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

Synthetic Workload Generation for Stress Testing Session- Based Systems

by

Diwakar Krishnamurthy, MEng.

**A thesis submitted to the Faculty of Graduate Studies and Research in partial
fulfillment of the requirements of the degree of**

Doctor of Philosophy

**Ottawa Carleton Institute for Electrical and Computer Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario
Canada, K1S 5B6**

January 23, 2004

© Copyright 2004, Diwakar Krishnamurthy

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-94203-1

Our file Notre référence

ISBN: 0-612-94203-1

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

PAGINATION ERROR.

ERREUR DE PAGINATION.

TEXT COMPLETE.

LE TEXTE EST COMPLET.

Abstract

A large portion of the world's computing infrastructure is dedicated to the support of enterprise applications such as e-commerce and business process management. Such applications are often business critical and require effective stress testing techniques to understand how they will perform under various workload conditions. The workloads of these systems are characterized by sessions of interdependent requests that often cause and exploit dynamically generated responses. Inter-request dependencies must be reflected in synthetic workloads for these systems to stress application functions in a representative manner. In addition, to support tasks such as capacity planning and service level management, it should be possible to have flexible control over many different workload characteristics that are likely to affect system performance. Simultaneously satisfying these requirements can make manual construction of synthetic workloads difficult and time consuming.

This thesis presents a synthetic workload generation approach that allows session-based systems to be stress tested in a representative, flexible, and automated manner. The proposed approach uses a workload model which includes attributes that can influence the performance of session-based systems. Statistical characterizations can be specified for each of these attributes. The approach employs a trace generation algorithm that automates the tasks of ensuring correct inter-request dependencies and simultaneously incorporating the desired characteristics for many different attributes into a synthetic workload. The chief advantage of the approach is the fine control it offers over workload characteristics. For example, it permits the characterizations of one or more attributes to be changed at a time while keeping those of others unchanged so that a system's sensitivity to those characteristics alone can be established.

The approach is validated in a number of ways through a stress testing case study of an e-commerce system. The study shows that the approach can produce synthetic workloads with characteristics that match those observed in real systems. The study also demonstrates that a synthetic workload created with the approach causes a nearly identical performance-impact on the system as the original workload it modeled. Finally, by using the approach to generate various workloads that differ from one another in carefully controlled ways, the study reveals that the system's performance is sensitive to the characteristics of attributes included in the workload model.

Acknowledgements

I am grateful to Professor Jerry Rolia for his invaluable guidance and support. His patience, encouragement, and keen insight have had a significant part in the successful completion of this work. I also thank him for his vision which ensured that I had a rather unique research experience replete with generous financial support and industry exposure.

I thank Professor Shikharesh Majumdar for co-supervising my research. His ideas, attention to detail, and critiques have immensely benefited my thesis. I am also thankful to him for arranging financial support at critical junctures of my graduate studies.

My sincere thanks go to Martin Arlitt of HP Labs for his collaboration, mentoring, and friendship. I also thank Professor Dorina Petriu for serving on all my thesis committees and for her constant encouragement. Special thanks go to my lab mate, Mohamed El-Darieby, for helping me in so many ways over the years.

I record my appreciation for the department staff for making my stay a pleasant one. I would like to especially thank Darlene Hebert for sorting out many a pay related issue. John Knox's timely system administration help and his friendship are also deeply appreciated.

I gratefully acknowledge the financial support of Carleton University, IBM Canada, HP Labs, Government of Ontario, Computer Measurement Group (CMG), and Communications and Information Technology Ontario (CITO).

Finally, I am grateful for the love and support of my family. I thank my brother,

Sudhakar, for being a constant source of inspiration. I would like to dedicate this work to my dear parents. This thesis is as much a result of their selflessness and unconditional love as it is an outcome of my efforts. Amma and Anna, I am truly blessed to be your son.

Table of Contents

Chapter 1	1
Introduction	1
1.1 Background and motivation	1
1.2 Research overview	6
1.3 Research contributions	10
1.4 Research scope	12
1.5 Thesis organization	13
Chapter 2	14
Related work	14
2.1 Characteristics of Web workloads	14
2.1.1 Characteristics of traditional Web workloads.....	14
2.1.2 Characteristics of session-based workloads.....	20
2.2 Generating synthetic Web workloads	23
2.2.1 Tools for generating traditional Web workloads	23
2.2.2 Tools for generating session-based workloads	27
2.2.3 Limitations of existing tools	31
2.2.3.1 Limitations due to the Markov chain based approach.....	31
2.2.3.2 Other limitations.....	34
2.2.3.3 Proposed solution	35
2.3 Summary and conclusions	36
Chapter 3	38
Workload characterization of a large Web-based shopping system	38
3.1 The system under study	38
3.1.1 System description.....	38
3.1.2 Measurement Data	42
3.2 Summary of Workload Data	44
3.3 Analysis of request arrivals	45
3.4 Characterization of sessions	47
3.4.1 Inter-request time.....	48
3.4.2 Session length	50

3.5 Characterization of requests	53
3.5.1 Request classes	53
3.5.2 Distribution of requests to unique objects	58
3.5.3 Temporal locality	60
3.6 Summary and conclusions	65
Chapter 4.....	68
The Session-based Web Application Tester.....	68
4.1 Overview	69
4.2 Workload Model.....	70
4.3 Trace generation.....	75
4.3.1 Creating valid sessions.....	75
4.3.2 Matching desired workload characteristics.....	76
4.3.2.1 Workload mix and session length distribution	77
4.3.2.2 Name-value lists and think time distributions.....	85
4.3.3 Summary.....	86
4.4 Request generation.....	87
4.5 Summary and conclusions.....	89
Chapter 5.....	93
Stress testing study of a TPC-W system.....	93
5.1 System under study	93
5.1.1 TPC-W	94
5.1.2 Experimental setup	95
5.2 Experimental methodology.....	97
5.3 Validation.....	107
5.4 Sensitivity analysis	113
5.4.1 Impact of session length and think time distributions	115
5.4.1.1 Comparison of exponential and empirical distributions.....	115
5.4.1.2 Effect of heavy-tailed distributions	117
5.4.1.3 Summary	124
5.4.2 Impact of workload mix.....	125
5.4.3 Impact of workload conditions on system capacity.....	129
5.5 Summary and conclusions.....	132

Chapter 6.....	134
Summary and conclusions.....	134
6.1 Summary.....	134
6.2 Contributions.....	139
6.3 Future work.....	141
References.....	143
Appendix A.....	152
Formulation of the linear programming problem.....	152
Appendix B.....	155
Time-complexity of the trace generation algorithm.....	155
Appendix C.....	157
Sample synthetic workload trace file.....	157

List of Figures

Figure 1.1: Conceptual overview of the stress testing approach.....	7
Figure 2.1: Example of a Markov chain for an e-commerce system	29
Figure 2.2: Example of an incorrect Markov chain	33
Figure 3.1: Logical diagram of the system under study.....	39
Figure 3.2 (a): Traffic volume: March 2000	46
Figure 3.2 (b): Traffic volume: July 2000.....	46
Figure 3.3 (a): Cumulative distribution of session inter-request times: March 2000	49
Figure 3.3 (b): Cumulative distribution of session inter-request times: July 2000.....	49
Figure 3.4 (a): Cumulative distribution of the session length: March 2000	51
Figure 3.4 (b): Cumulative distribution of the session length: July 2000	51
Figure 3.5 (a): Distribution of requests to objects: March 2000	59
Figure 3.5 (b): Distribution of requests to objects: July 2000.....	59
Figure 3.6: Cache miss ratios with LRU, LFU, and LFU-Aging policies (July data)	63
Figure 4.1: Components of SWAT	69
Figure 4.2: Schematic diagram of trace generation process.....	76
Figure 4.3: Comparison of achieved and desired workload mixes	83
Figure 4.4: Comparison of achieved and desired session length distributions	84
Figure 5.1: Experiment setup	96
Figure 5.2 (a): LLCD plots of session lengths	103
Figure 5.2 (b): LLCD plots of think times	103
Figure 5.3 (a): Cumulative distribution of session inter-arrival time for Original and Synthetic workloads.....	108
Figure 5.3 (b): Cumulative distribution of session length for Original and Synthetic workloads	109
Figure 5.3 (c): Cumulative distribution of think time for Original and Synthetic workloads	109
Figure 5.3 (d): Workload mixes of Original and Synthetic workloads	110
Figure 5.4 (a): Cumulative distribution of response size for Original and Synthetic workloads	112
Figure 5.4 (b): Cumulative distribution of response time for Original and Synthetic workloads	112
Figure 5.4 (c): Cumulative distribution of number of concurrent sessions for Original and Synthetic workloads.....	113
Figure 5.5 (a): Cumulative distribution of response time for Empirical and Exponential workloads	116
Figure 5.5 (b): Cumulative distribution of number of concurrent sessions for Empirical and Exponential workloads.....	116
Figure 5.6 (a): Cumulative distribution of response time for Empirical and Heavy-tailed session length workloads.....	119
Figure 5.6 (b): Cumulative distribution of number of concurrent sessions for Empirical and Heavy-tailed session length workloads	119
Figure 5.7 (a): Cumulative distribution of response time for Empirical and Heavy-tailed think time workloads	120

Figure 5.7 (b): Cumulative distribution of number of concurrent sessions for Empirical and Heavy-tailed think time workloads.....	120
Figure 5.8 (a): Cumulative distribution of response time for Empirical and Combined-heavy-tailed session length workloads.....	122
Figure 5.8 (b): Cumulative distribution of number of concurrent sessions for Empirical and Combined-heavy-tailed session length workloads.....	122
Figure 5.9: LLCD plots of response time for the Combined-heavy-tailed and Heavy-tailed session length workloads.....	123
Figure 5.10 (a): Cumulative distribution of response time for HV-Mix and LV-Mix (empirical case).....	126
Figure 5.10 (b): Cumulative distribution of number of concurrent sessions for HV-Mix and LV-Mix (empirical case).....	126
Figure 5.11 (a): Cumulative distribution of response time for HV-Mix and LV-Mix (heavy-tailed case).....	128
Figure 5.11 (b): Cumulative distribution of number of concurrent sessions for HV-Mix and LV-Mix (heavy-tailed case)	128

List of Tables

Table 3.1: Breakdown of requests by object types.....	45
Table 3.2: Relative mean application server CPU demands for request classes.....	55
Table 3.3: Request class mix for March and July data.....	55
Table 5.1: Configuration settings for the TPC-W system.....	99
Table 5.2: Workload characteristics.....	100
Table 5.3: Session length and think time statistics.....	102
Table 5.4: Mean no-load response times of request types and workload mixes.....	104
Table 5.5: Comparison of Original and Synthetic workloads.....	110
Table 5.6: R_{mean} and R_{95} values for the Empirical and Exponential workloads.....	116
Table 5.7: Effect of the heavy-tailed session length and think time distributions	117
Table 5.8: R_{mean} and R_{95} values for LV-Mix and the HV-Mix (empirical case).....	127
Table 5.9: R_{mean} and R_{95} values for the LV-Mix and the HV-Mix (heavy-tailed case)	127
Table 5.10: Effect of workload conditions on R_{95}	131
Table B-1: Elapsed time measures for the trace generation algorithm	155

Chapter 1

Introduction

1.1 Background and motivation

During the early years of the Web, Web-based server systems predominantly served only *static* content, constituting files (e.g., HTML, image, and multimedia files) stored in their file systems. The processing of user requests is fairly straightforward in such a system. Users submit requests for files and in response the Web server retrieves and transmits the desired file. The server's workload can thus be visualized as a series of requests for files in its file system. Such systems are referred to as *traditional* Web systems in this thesis.

In recent years, however, Web-based systems have evolved into more complex distributed systems that support applications such as e-commerce, online banking and online trading. These systems are made up of tiers of Web servers, application servers, database servers, and payment servers connected by networking infrastructure. Responses to user requests are typically constructed *dynamically* by invoking the services of one or more of the server tiers. Consider an e-commerce system example. Handling a request from a user to display the contents of a shopping cart may involve the following steps: 1) a Web server receives the request and passes it on to an application server; 2) the application server queries and obtains the shopping cart information of the user from a database server; 3) the application server formats the database query results in HTML and passes the HTML response to the Web server; and 4) the Web server transfers the response received from the application server to the user. The manner in which a request is processed depends on the application logic of the system. This is complex when compared with traditional Web systems which do not have such application logic.

The workload of such systems must be characterized in terms of sessions. A *session* is defined as a sequence of inter-dependent requests submitted to the system by a single entity. Dependencies arise because some requests depend on the responses of earlier requests in a session. For example, an order cannot be submitted to an e-commerce system unless the previous requests have resulted in an item being ordered. This phenomenon is defined as *inter-request dependency* and this class of systems is referred to as *session-based* systems.

Enterprises are increasingly relying on session-based systems to support critical business functions. As a result, the performance of these systems becomes a crucial factor that influences their profitability. *Stress testing* [41], a technique of measuring performance by submitting a synthetic workload to a system under study within a controlled environment, is often used to determine whether a system can meet its performance objectives. The *synthetic workload* used in the stress test serves to mimic the request patterns of real system users. Measurements such as user response times and server utilizations are collected during the tests and used to support several performance management exercises. Some examples of such exercises are as follows:

- 1) **Service level assessment** – *Service levels* are often used to specify performance guarantees to a system's users. Stress tests can be used to assess whether a particular system can deliver a target service level to its users. For example, stress test results can be used to find out if it is possible to respond to user requests within a target time (e.g., 1 second) while operating the system at a target utilization.
- 2) **Capacity planning** – *Capacity planning* is the process of estimating the computing resources that will be required to handle anticipated changes to a system's workload [29].

Stress tests can be used to support capacity planning exercises. A system's performance can be measured under various synthetic workloads, each emulating a particular workload scenario of interest for capacity planning.

Different workloads may stress a system in significantly different ways and can hence yield very different performance results. Consequently, the choice of synthetic workloads forms an important aspect of stress testing. Stress testing methodologies must address the following requirements with respect to synthetic workloads:

1) **Representativeness** – To reach reliable conclusions (e.g., about service levels and capacity requirements) based on the results of a stress test, the synthetic workload used must be representative of real observed workloads. A synthetic workload is said to be *representative* of a real workload if both workloads result in similar performance when submitted to a system [23]. Several performance indices may be used to compare the workloads; a detailed list can be found in [15].

One way of ensuring representativeness is to replay pre-recorded traces of real workloads during stress tests. However, such an approach may not be feasible in practice and it is also difficult to cause selective changes to the workload for studying how changes in specific workload characteristics impact performance [21]. Instead, important features (from a performance standpoint) of real workloads can be captured by means of a *workload model* [29, 21, 10], which contains a set of workload *attributes* along with statistical characterizations for the attributes. A synthetic workload is then constructed such that it adheres to the workload model. File size, response size, number of requests to individual files, and time between successive requests are examples of attributes used

to characterize Web-based systems.

The representativeness of a synthetic workload is significantly influenced by the attributes in the workload model and the characterizations used for them. A discussion on the perils of using incorrect characterizations can be found in [22, 50]. Ideally the attribute characterizations should be deduced from measurements (e.g., Web server access logs).

2) **Flexibility** – In practice, it is very difficult to know precisely what a real workload's characteristics will be. The system under study may not have enough (or any) real workload data to conclusively establish the characterizations to be used. Also, the characterizations deduced by analyzing workloads of other systems may not necessarily be appropriate for the system under study. The stress testing methodology must thus provide the flexibility to achieve arbitrarily desired characterizations for attributes in the workload model so that the system's performance can be observed under many different characterizations of interest. This feature is also needed for emulating anticipated changes to a system's workload for capacity planning purposes.

3) **Compactness** - The workload model must contain as few attributes as possible. Attributes that do not have a significant impact on performance should be identified and excluded from the workload model.

4) **Automation** – The process of obtaining a synthetic workload that adheres to a specified workload model must be automated to alleviate difficulties in conducting stress tests. There are a number of challenges in achieving automation. For instance, it is difficult to simultaneously incorporate specified characterizations for individual attributes

of the workload model into the synthetic workload. Also, for session-based systems, automation must handle inter-request dependencies. A synthetic workload that does not satisfy the correct inter-request dependencies may not stress the system's application functions as intended since many requests will be treated as erroneous requests by the system.

Although significant effort has been focused on devising stress testing methodologies satisfying these requirements for traditional Web systems [10, 55], there exists very little corresponding work for session-based systems. To the best of this author's knowledge, the only published studies on session-based workloads are due to Menasce *et al.* and Kant *et al.* [44, 33]. However, these studies have several limitations related to the requirements discussed previously. Firstly, both the methodologies proposed can only be used with systems having first-order inter-request dependencies, i.e., systems where the next request to be submitted in a session can be decided based solely on the current request. In general, session-based workloads have higher-order dependencies. The result is that some sessions generated using these methodologies may stress a system's application functions in a non-representative manner. Secondly, the methodologies do not provide flexibility over the characterizations of several key attributes that previous studies had suggested might impact system performance. Finally, workload emulation issues arising out of the dynamic nature of the system's responses are not addressed. A more detailed description of these issues is provided in Section 2.2.3.

An advantage of a flexible stress testing methodology is that it can be used to conduct a sensitivity analysis to identify important workload characteristics affecting system performance [21]. For example, such a methodology would allow characterizations of

one or more attributes to be changed at a time while keeping those of others unchanged so that the system's sensitivity to those characteristics alone can be established. Such studies were instrumental in establishing how the characteristics of file sizes, file popularity, and user idle times influence the performance of traditional Web systems [10, 14]. However, perhaps due to the lack of similar tools, sensitivity analysis studies have not yet been repeated for session-based systems.

This thesis describes a stress testing approach for session-based systems that addresses the limitations discussed above. The approach employs a workload model that contains several attributes likely to be of interest while evaluating the performance of session-based systems. The characterizations for these attributes can either be based on those observed in the Web server access logs of real systems or perturbations for the purpose of a sensitivity analysis. The thesis presents a technique that automatically produces a synthetic workload simultaneously satisfying desired characterizations for all attributes in the workload model while maintaining the correct inter-request dependencies for the system under study. The flexibility of the approach is exploited to experimentally evaluate how characteristics of individual workload attributes affect the performance and capacity of a session-based system. An overview of the approach is provided in the next section.

1.2 Research overview

Figure 1.1 provides a conceptual overview of the stress testing approach. The approach uses a workload model that includes attributes that influence performance and hence are of interest for the performance evaluation of session-based systems. It relies upon an input trace of valid sessions for the system under study to enforce correct inter-request

dependencies in the synthetic workload. The “offline” *trace generation engine* produces a synthetic trace of sessions exhibiting the correct inter-request dependencies and the characteristics specified for attributes in the workload model. Finally, the *request generation engine* submits the sessions in the synthetic trace to the system under study and collects various performance metrics of interest from the test. Next, each component is described in further detail.

Figure 1.1: Conceptual overview of the stress testing approach

The selection of attributes for the workload model is based on a study of a large e-commerce system. The observed characteristics of several workload attributes are presented and their impact on system performance is discussed. In particular, the study finds evidence of significant variability in the number of requests per session, the idle times of sessions, and the service demands for the different types of requests in the system. Each can adversely impact a system’s performance. Other studies [43, 61] have also reported similar behaviour with respect to these attributes, suggesting that the high variability might be a ubiquitous phenomenon in session-based workloads. Consequently, the thesis argues that these attributes must be considered and modeled

with appropriate characterizations to cause representative loads during stress tests.

The workload model permits the flexibility of employing two different workload generation modes depending on the desired type of arrival characterization. A *session mode* allows control over the arrival instants of new sessions at the system under study whereas a *request mode* offers control over the arrival instants of requests. Depending on the mode chosen, the model also allows users to specify characterizations for the number of requests per session, the idle times in a session, and the proportions of the different types of requests (e.g., View_Product, Add_Product_To_Shopping_Cart) in the system.

The trace generation engine uses a linear programming technique to create a synthetic trace of sessions that simultaneously matches the desired characteristics for the number of requests per session, session idle times, and the proportions for the different types of requests. Inter-request dependencies are satisfied by ensuring that the synthetic sessions only follow those request sequences observed in an input trace of valid sessions for the system under study. The trace generation engine also identifies and tags requests that have to be dynamically constructed based on system responses received during the request generation phase.

The request generation engine submits the synthetic workload trace to a system under study according to the desired (session or request) arrival characteristics. The thesis research extends *httpperf* [46], a highly scalable and open source request generator, to implement the request generation approach. The extensions support the request mode, dynamic construction of requests, and detailed logging of performance metrics. In

addition to these features, *httperf* also has built-in support for features such as HTTP 1.1 [27], SSL [57], HTTP cookies [26], and emulation of request abandonments and retries. These are all compatible with the extensions.

The approach developed addresses the stress testing requirements described in the previous section as follows:

- a) Representativeness – The approach promotes representativeness in a number of ways. Firstly, existing studies have provided widespread evidence of burstiness in the arrival of requests at session-based systems [43, 61]. Such burstiness can degrade a system's performance. The approach provides capabilities for incorporating such behaviour within synthetic workloads thereby ensuring their representativeness with respect to real workloads. Secondly, the characterizations for all the attributes used in the workload model can be estimated from commonly available workload data such as Web server access logs. This facilitates synthetic workloads whose characteristics match those observed in real systems.
- b) Flexibility - The trace generation engine uses a linear programming technique that allows stress test teams to specify arbitrarily desired characteristics for the attributes in the workload model. This in turn facilitates sensitivity analysis studies. Also, unlike existing approaches, the request generation engine offers the flexibility of causing requests to arrive at pre-determined time instants to the system under study while simultaneously ensuring the valid behaviour within a session of submitting a request only after the response to the previous request has been completely received.
- c) Automation – Given a trace of valid sessions, the approach automates the tasks of

satisfying inter-request dependencies and simultaneously achieving the specified characteristics of many different attributes. Also, the approach permits synthetic workloads to be automatically created from real workload data (e.g., as recorded in Web server access logs) for the system under study. This is because the trace of valid sessions required by the approach can be extracted in a straightforward manner from such data.

Controlled workloads created with the approach are used to study the individual and joint impact of three characteristics on the performance of a session-based system: the distribution of the number of requests per session, the distribution of session idle times, and the distribution of request service times. The results establish that higher variability in these distributions can cause increased contention among sessions for a system's resources thereby degrading the system's responsiveness. Consequently, high variability can force a system to operate at reduced utilization in order to maintain a given service (responsiveness) level to its end users.

1.3 Research contributions

This research makes the following contributions:

- 1) The thesis identifies the limitations of existing stress testing methodologies when applied to session-based systems.
- 2) The research analyzes workload measured at a large e-commerce system. It discusses how the observed characteristics of several workload attributes impact the system's performance. The study helps to identify attributes that need to be considered when stress testing session-based systems.

3) A synthetic workload generation approach for session-based systems is developed that addresses the limitations of existing methodologies. The approach allows systems to be stress tested in a representative, flexible, and automated fashion.

4) The ability of the proposed approach to produce synthetic workloads differing only in the characteristics of one or more chosen attributes is exploited to conduct a sensitivity analysis study for a session-based system. The results illustrate how variability in the characteristics of several attributes affects the performance and capacity of the system.

The following publications have resulted from this research.

1) D. Krishnamurthy, J. Rolia, and S. Majumdar, "SWAT: A Tool for Stress Testing Session-based Web Applications," *Proceedings of the 2003 Computer Measurement Group (CMG) conference*, December 7 –12, 2003, Dallas, Texas, USA.

2) D. Krishnamurthy, J. Rolia, and S. Majumdar, "SWAT: A Tool for Stress Testing Session-based Web Applications," (Extended version), Technical report, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, May 2003.

3) D. Krishnamurthy, J. Rolia, M. Arlitt, and S. Majumdar, "A Systematic Method for Creating Performance Tests for E-Commerce Systems," *Proceedings of the 11th GI/ITG Conference on Measuring, Modeling and Evaluation of Computer and Communications Systems*, September 11-14, 2001, Aachen, Germany.

4) 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, August 2001.

1.4 Research scope

This section outlines the scope of the research.

The latency perceived by an end user of a session-based system is the sum of the latencies suffered at the session-based system, at the Internet (i.e., at Internet links and routers between the system and the user), and at the user's host machine. The approach developed in this thesis is concerned only with the component due to the session-based system. Characterizing Internet and user host machine latencies are outside the scope of this research.

An end user of a session-based system may also submit requests for static files in addition to the requests eliciting dynamic responses. For example, the system may generate a dynamic HTML response for the user containing embedded references to static image files. The static files are often hosted on separate servers to improve the performance of the system [5, 61]. In such a case, the system generating the dynamic response does not have to wait for these static files to be transferred and is free to handle other requests. This research focuses only on the performance of the servers generating the dynamic responses. Consequently, requests for static files are not considered. The performance of servers hosting the static files can be evaluated by applying existing stress testing methodologies for traditional Web systems.

Identifying statistical characterizations for the workload model attributes that are representative for all session-based systems is complex and is beyond the scope of this research. Instead, the approach developed in this thesis permits a sensitivity analysis to show how the characterizations of the attributes impact a system's performance.

1.5 Thesis organization

The rest of the thesis is organized as follows.

Chapter 2 reviews related work on Web workload characterization and stress testing. It concludes by identifying limitations in existing stress testing approaches when applied to session-based systems.

A detailed workload characterization study of a large e-commerce system is presented in Chapter 3. Characteristics of several workload attributes are presented and their impact on the system's performance is discussed.

Chapter 4 presents the proposed stress testing approach. It describes in detail the workload model, trace generation engine, and the request generation engine.

Chapter 5 describes a stress testing case study using the proposed approach. The study employs controlled workloads to investigate the sensitivity of a session-based system's performance to the characterizations used for several attributes in the workload model.

Chapter 6 summarizes the thesis and provides directions for future research.

Chapter 2

Related work

This chapter reviews related work on stress testing Web-based systems. Section 2.1 presents findings from literature on Web workload characteristics. In particular, the section considers several workload attributes that can impact the performance of Web-based systems and details their observed statistical characteristics. Section 2.2 reviews existing stress testing tools for Web-based systems. It also discusses the limitations of existing approaches for stress testing session-based systems. Section 2.3 provides summary and conclusions.

2.1 Characteristics of Web workloads

As mentioned in Chapter 1, Web-based systems can be categorized as traditional Web-based systems and session-based systems. Section 2.1.1 and Section 2.1.2 present important results from existing studies on traditional and session-based Web workloads, respectively.

2.1.1 Characteristics of traditional Web workloads

Traditional Web systems predominantly serve static content, such as HTML, image, and multimedia files that are stored in their file systems. The following attributes are commonly used to characterize such workloads.

1. File popularity – This attribute captures the proportions of total requests made to the unique files in the server. File popularity influences server performance since popular files are likely to get served from the server's caches (e.g., the file system cache) rather than the physical disk. Studies of traditional Web workloads [3, 7, 48] have shown that

file popularity follows a Zipf-like [65] distribution. A distribution is said to be Zipf-like if the number of requests to the i^{th} most popular file is proportional to $1/i^\beta$, where β , called the Zipf exponent, is non-negative. The presence of a Zipf-like distribution indicates that a large proportion of the total requests is for a small set of highly popular files. The larger the β value, the more pronounced the concentration of requests is towards the popular files.

The value of β was found to be nearly 1 for the World Cup '98 (soccer) Web site workload [7] and the CNBC (financial news) Web site workload [48]. This implies that the n^{th} most popular file in these sites is likely to be accessed twice as much as the $2n^{\text{th}}$ most popular file. However, studies have shown that Web proxy workloads exhibit a comparatively lesser skew in file popularity since their β values typically ranged from 0.6 to 0.9 [8]. A discussion on possible reasons for this phenomenon can be found in [48].

2. Temporal locality - Temporal locality is the property by which a recently requested file is more likely to be requested again in the near future. Temporal locality is important for performance evaluation since it affects the behaviour of a system's caches. A workload with good temporal locality increases the likelihood of files being found in a cache.

Temporal locality is characterized by using the notion of an *LRU* (Least Recently Used) *stack distance* [56]. LRU stack distances are calculated as follows. The unique files in the system are stored in a pushdown stack. Given a sequence of requests to files, each request in the sequence is considered in order. The file corresponding to the request is located in the stack and moved to the top of the stack. The depth from the top at which the requested file was located before it was moved to the top is the request's LRU stack

distance. A workload is said to exhibit good temporal locality if most of the stack distance observations correspond to the top portions of the stack.

Studies have shown that traditional Web workloads exhibit good temporal locality. For example, in the World Cup '98 workload [7] 90 % of the requests could be located in the top 4% of the stack. Studies have also established that stack distances follow the lognormal distribution [10].

Recent studies have suggested that temporal locality is caused by two distinct factors, namely long-term popularity of “hot” files, and short-term temporal correlations in requests to files [30, 16]. Mahanti *et al.* performed simulations using real workloads to study the impact of the short-term component on the performance of proxy Web caches [40]. Their results show that the effect of the short-term component is pronounced only at small cache sizes. The short-term component is found to have negligible impact on cache performance when the capacity of the cache increases to accommodate larger percentages of the unique files hosted by the server. Breslau *et al* have also observed a similar behaviour based on their study of a different set of real Web proxy workloads [13]. Based on this result, both studies have suggested that it might be adequate to consider the long-term component alone for evaluating the performance of Web-based systems employing realistically sized caches. This issue is studied in detail in Section 3.5.

3. Spatial locality – Spatial locality is the property by which files that are statically related to a recently accessed file are likely to be accessed in the immediate future. For example, a request to an HTML file is likely to be followed by requests to its embedded

files. Spatial locality can be exploited to achieve better performance by pre-fetching the related files and storing them in caches. Willis *et al.* argue the importance of collectively characterizing the files that are typically retrieved together for designing better Web caching algorithms [64]. Abhari *et al.* devised cache replacement algorithms that deal with Web objects [1]. A *Web object* is the set of files making up a Web page. The authors show that Web-object-aware caching algorithms yield better performance than traditional algorithms that deal with individual files.

4. Embedded references – Embedded references denotes the number of files embedded within a HTML response. Embedded references have significant influence on TCP connection establishment and maintenance overheads [10]. Based on their study of several real Web workloads, Crovella *et al.* found that the distribution of number of embedded references follows a Pareto distribution [10].

5. File size- Characterizing file sizes is important since they impact the performance of a server's file system. Several studies on traditional Web workloads reveal that the distribution of file sizes is heavy-tailed. A random variable X is said to have a *heavy-tailed* distribution if its tail decays according to a power law. In other words,

$$P[X > x] \approx x^{-\alpha} \text{ for } x \rightarrow \infty, 0 < \alpha < 2$$

The parameter α determines the “heaviness” of the tail. The lower the value of α , the higher are the probabilities for very large values of the random variable, and hence, heavier the tail. Random variables with heavy-tailed distributions exhibit very high variability.

Barford *et al.* considered several real Web server workloads and showed that the tail of the file sizes follows a Pareto distribution, which is a heavy-tailed distribution [17,10]. A similar behaviour was also noticed in the World Cup 98 workload [7]. However, recent work by Downey [20] questions these results and argues that file sizes are better modeled through a lognormal distribution, which is a not a heavy-tailed distribution. The exact nature of the file size distribution is currently a matter of great debate [20, 24, 25].

6. Response size - Response size refers to the size of the file returned by a server in response to a request. The distribution of response sizes can be quite different from the distribution of file sizes since not all files on the server are equally popular. The response size distribution determines the demands placed on the server and networking infrastructures. Crovella *et al.* characterized several real Web server workloads and showed that the tail of the response sizes follows a Pareto distribution [17,10]. However, a lognormal distribution was found to be a good choice for modeling the response size distribution of the World cup '98 workload [7].

7. Think time - *Think time* is defined as the time taken by a user to issue a new request after receiving the response to a particular request. Research has shown that think times influence the performance of Web servers. For example, Crovella *et al.* [10] have shown that long think times can force a Web server to maintain a large number of concurrent TCP connections thereby impacting the loads placed on a server's CPU and memory. Studies of traditional Web workloads have shown that think times follow heavy-tailed distributions [17,10].

8. Self-similarity – Several studies on traditional Web workloads [17, 10] have shown

that the arrival of requests at the system is self-similar. *Self-similar* workloads exhibit significant request correlations or bursts over multiple timescales [17]. A system's ability to handle such bursts is determined by features such as its throughput capacity, scheduling disciplines [4], maximum allowable queue lengths, and timeout values [49]. If a system is not able to support bursts at some timescale, considerable queuing delays may arise [10, 12, 49, 60]. Consequently, stress testing methodologies must provide capabilities for emulating such behaviour if it may be present in a system's expected workload.

Leland *et al.* show that self-similar traffic can result when there is a large number of concurrent ON/OFF sources exhibiting heavy-tailed durations of activity, called *ON* times, and heavy-tailed durations of inactivity, called *OFF* times [39]. A typical Web user can be represented as an ON/OFF source where the ON times correspond to the times during which files are transferred and the OFF times correspond to user think times. Based on their characterization of real workloads from over 30 Web sites, Crovella *et al.* note that the heavy-tails observed in file sizes, and think times cause the ON and OFF times of Web users to be heavy-tailed, respectively, which according to the authors provides a possible explanation for the self-similarity in Web workloads [17].

To summarize, this section reviewed workload characteristics of traditional Web-based systems. File popularity, temporal locality, spatial locality, embedded references, file size, response size and think time are attributes commonly used to characterize workloads in these systems. The studies provide evidence that the distribution of file sizes, response sizes, and think times are heavy-tailed. It is widely believed that the bursty or self-

similar nature of request arrivals in Web workloads may be a direct result of the heavy-tailedness of these attributes.

2.1.2 Characteristics of session-based workloads

A brief review of existing research on the characteristics of session-based workloads is presented in this section. In contrast to traditional Web workloads, there exist fewer published studies on session-based workloads since the workload data are often considered confidential by those operating these systems. Also, in all the workloads researched, requests for dynamic objects (e.g., Search, Pay) far outnumbered requests for static objects (e.g., HTML and image files) with the dynamic requests typically constituting 80% or more of the total requests [43, 5].

Menasce *et al.* characterize workloads from an online bookstore and an online auction site [43]. They propose a multi-level analysis scheme that characterizes attributes in the *session layer*, *function layer*, and *HTTP request layer*. At the session layer, the authors characterize *session length* defined as the number of requests submitted by a user in a single session. Results show that the session length distribution is heavy-tailed for both the bookstore and auction site. However, the tail decay is slower in the bookstore which contains some sessions with as many as 10,000 requests. The authors attribute this to automated agents or *robots* that access the system endlessly to collect information such as the pricing of products. In a follow up work, the authors propose techniques to identify robots so that their access can be blocked [42].

The function layer characterization categorizes requests into request types based on the application function invoked. For the bookstore, eight distinct request types, namely

Home, Browse, Search, View, Add, Pay, and Account are identified. Analysis shows that over 70% of the requests in the bookstore are for browsing product information. In contrast, the percentage of requests associated with purchasing a product is only 1.19%. The auction site also displayed predominance of browse-related requests but not to the extent seen in the bookstore site. Kant *et al.* who studied the workload of a Business-to-Consumer (B2C) e-commerce system also report a predominance of browse-related activity [31]. However, the authors find that the percentage of purchase-related requests is significantly larger in a Business-to-Business (B2B) e-commerce system than the B2C system.

At the request layer, session-based workloads are found to exhibit self-similarity [43,32, 61]. The study by Menasce *et al.* reports bursty request arrivals over several timescales. The authors argue that the observed self-similarity could possibly be due to the heavy-tailed nature of the session lengths and user think times in the workloads considered. Vallamsetty *et al.* also observed self-similar behaviour in the B2C and B2B workloads they studied. As expected, the authors found the ON and OFF times of sessions (i.e., users) to be heavy-tailed. However, an interesting aspect of the B2C workload is that the ON times are heavy-tailed even though the response size distribution is *not* heavy-tailed. This indicates that response sizes are not a good predictor of ON times, unlike as in traditional Web workloads. The authors instead found that the ON times are influenced more by the service times for requests at the backend systems (e.g., application and database servers), which were found to be highly variable.

Arlitt *et al.*, who characterized the workload of an online shopping system, confirm many of the results reported in the above studies [5]. Over 95% of total requests in the

workload they studied are for dynamic objects. An analysis of object popularity revealed that a large fraction of the total requests is for a small number of very popular objects. The study also reported a significant presence of robots, which is consistent with the results reported by Menasce *et al.* [43]. Finally, the study found that there is a large difference in the resource demands for the different request types in the system. Detailed findings of this study are reported in Chapter 3.

The studies reviewed in this section, except that of Arlitt *et al.*, did not characterize temporal locality. A possible reason may be the predominance of dynamic objects which typically are hard to cache. However, session-based systems incorporating frameworks to cache dynamic data are not uncommon [5, 28]. Characterization of temporal locality can be important for such systems. Chapter 3 presents a characterization of temporal locality in the workload of a large e-commerce system that employs an application server cache to store dynamic objects. The studies have also not considered spatial locality. The main reason is that techniques that exploit spatial locality (such as pre-generating dynamic responses) are not commonly employed by session-based systems. Consequently, the thesis does not focus on this attribute any further.

To summarize, session-based workloads have not been researched extensively on account of their confidential nature. Existing studies point to the presence of robots which cause the session length distribution to be heavy-tailed. For B2C e-commerce systems the percentage of purchase-related requests is very small when compared to the percentage of browse-related requests. B2B systems also exhibit a similar behaviour though not to the same extent as B2C systems. Finally, self-similarity is observed in the session-based workloads reviewed. There exists some evidence to suggest that high variability in

request resource demands is widely prevalent and that it could be one of the factors that contribute to self-similarity in session-based systems.

Workload characterization is useful for developing workload models. For stress testing, techniques are required to derive a synthetic workload from a workload model and submit it to a system under test. These techniques are embodied in tools called *workload generators*. The following section reviews existing workload generators for Web-based systems.

2.2 Generating synthetic Web workloads

Generating synthetic Web workloads typically involves two steps: 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 review existing solutions for trace generation and request generation.

Section 2.2.1 describes solutions for traditional Web workloads while Section 2.2.2 describes solutions for session-based workloads. Section 2.2.3 discusses the limitations of existing workload generators for stress testing session-based systems.

2.2.1 Tools for generating traditional Web workloads

SURGE [10] is a popular workload generator used for stress testing Web servers. An offline trace generation engine is used to create a trace of HTTP requests for a set of files on the Web server under study. The tool allows control over distributions for file size,

response size, file popularity, temporal locality, and think time. Users can specify the values of parameters for the distributions or use default values that were deduced by characterizing a large number of real Web server workloads. The request generation engine submits requests in the trace to the Web server by employing ON/OFF sources called user-equivalents that mimic a user's browsing behaviour. *User-equivalents* are essentially software processes that alternate in an endless loop between submitting requests in the trace, and lying idle for a period determined by the think time distribution. SURGE can be used to determine the capacity of a system in terms of user equivalents; the number of user equivalents can be increased until the responsiveness of the system exceeds a threshold.

The user equivalents approach has a number of limitations. Firstly, it is not possible to know beforehand the values for metrics such as the HTTP requests issued per second or the bytes requested per second during a test. This is because the rate at which a user-equivalent issues requests depends on system state; long delays for requests resulting from a heavily loaded server will increase the time between successive requests from a user-equivalent. More generally, due to dependence on server state, the approach does not permit control over characteristics of the aggregate arrival process of requests observed at the server. For example, it is not possible for a user to specify the exact instants at which requests have to arrive at the server under study. Secondly, since the number of user-equivalents can only remain constant during a test run, it is difficult to parameterize the workload generator to match real observed workloads where the number of concurrent users typically changes over time. Lastly, with a one-process-per-user-equivalent implementation, a large number of processes may be required to cause heavy

loads on the server.

Request generation engines that employ *aggregate workload generation* address some of these limitations. In this scheme there is no notion of a user-equivalent. Instead, requests are issued at the appropriate time instants so that a specified request arrival process is achieved at the server. Unlike the user-equivalents approach, the arrival of requests is *independent* of system load; a heavily loaded server does not slow down the request rate. Studies have shown that the number of processes or threads required to drive a server to its capacity is much smaller with aggregate workload generation than with the user-equivalents approach [9, 46]. However, due to its very nature, aggregate workload generation can run into difficulties whenever fine control over user-behavioural aspects is required. Examples include emulating the retrieval of embedded references, HTTP 1.1-style persistent connections [27], HTTP 1.1 pipelining [27], HTTP cookies [26], and user think times. Also, this approach is less intuitive than the user-equivalents approach since the workload cannot be viewed in terms of a user browsing behaviour.

Banga *et al.* developed an aggregate request generation engine for Web servers called *S-Clients* [9]. *S-Clients* decouples the HTTP request issuing and HTTP response handling functionalities within the workload generator. A connection establishment thread generates HTTP requests according to a specified request rate distribution. A separate connection handling process dequeues the HTTP requests scheduled by the connection establishment thread and handles the actual communication with the Web server. With this approach the connection establishment thread does not stall waiting for replies from the server. As a result very high connection establishment rates can be achieved independent of the server's state.

Mosberger and Jin developed *httperf*, a highly scalable request generation engine for Web servers [46]. In addition to aggregate workload generation, the tool also supports hybrids of the aggregate and user-equivalents approaches. The hybrid aggregate approach supports the emulation of persistent connections and pipelining. However, this functionality is achieved at the expense of exact control over the server request arrival process. The hybrid user-equivalents approach starts HTTP sessions, which contain sequences of HTTP requests, specified in a trace file at intervals chosen from a specified distribution. The sessions are started independent of system load; a busy server does not slow down the arrival of new sessions. Each session started behaves as a user-equivalent by alternating between submitting requests and remaining idle. The hybrid user-equivalents approach retains the advantages of the user-equivalents approach while eliminating some of its disadvantages. For example, due to its load-independent session arrivals, it allows users to precisely know beforehand the HTTP requests per second that will be issued during a test¹, and permits the number of concurrent users to vary during a test, as in real workloads. This thesis relies on *httperf* for request generation. A detailed description is provided in Section 4.4.

To summarize, SURGE is a popular workload generator for Web servers. It can produce synthetic HTTP traces with desired distributions for file size, response size, file popularity, embedded references, think time, and temporal locality. SURGE employs user-equivalents, which are software processes emulating ON/OFF behaviour, to submit requests in the trace to a server under study. The user-equivalents approach does not permit control over characteristics of the aggregate request arrival process seen at the

¹ Under steady-state conditions, the requests issued per second during a test is the product of the specified session arrival rate and the mean number of requests per session.

server. Request generators that employ the aggregate workload generation approach address this limitation but suffer from difficulties in emulating user behaviour. Finally, the hybrid user-equivalents request generation approach improves upon the user-equivalents approach by offering control over the requests issued per second during a stress test.

The workload generators reviewed in this section were designed primarily for systems that do not have inter-request dependencies. The next section reviews existing tools that address the problem of generating synthetic workloads for session-based systems.

2.2.2 Tools for generating session-based workloads

The request generation techniques reviewed in the previous section are still applicable for session-based workloads since these workloads essentially consist of HTTP and/or HTTPS requests. However, the trace generation step must address the issue of handling inter-request dependencies. Recalling from Chapter 1, requests from a session must be submitted and completed in a proper order to be considered valid. For example, considering an e-commerce system, a session can issue a `Purchase` request only after a product has been added to the shopping cart through a previous `Add` request. It should also be noted that requests for session-based systems can be split into two parts: 1) a request type; and 2) a name-value list. Typically a *request type* instructs an application server to execute a particular server-side script and the *name-value list*, which follows the request type, provides input data for the script. For example, the request `/Add?product_id=1&number=2` has `Add` as its request type and `product_id=1&number=2` as its name-value list.

The common technique to handle inter-request dependencies within sessions has been the use of first-order Markov chains [36, 44, 33]. Markov chains are described by states and transitions between states. The states of the Markov chain represent a system's request types. Transitions between the states model the user behaviour of navigating from one request type to another within a session. Transitions are associated with probabilities and the sum of a state's outgoing transition probabilities is one.

Figure 2.1 shows an example Markov chain for an e-commerce system. A user submits a Home request type for visiting the homepage of the system. She then follows it up with requests to View a product and Add a product to her shopping cart. Next she has the option of invoking either Delete or Purchase or View. Both Delete and Purchase are equiprobable with their individual transition probabilities being 0.4 while View is invoked with a probability of 0.2. From Delete or Purchase, the user can either View more products or Exit the system with equal probability. The Exit state represents the end of the session. The transition from Exit to Home signifies the start of a new session. The transition probabilities determine the number of "visits" to each state in the chain and hence the mix of request types.

The Markov chain handles inter-request dependencies as follows. With a first-order Markov chain, transition to a particular state depends on the current state. For example, from Figure 2.1, Delete and Purchase can occur in a session only after an Add has occurred. Menasce *et al.* describe algorithms to deduce the first-order Markov chain for a system from its Web server access log files [44]. By traversing the deduced Markov chain a trace of sessions can be created for stress testing the system. The limitations of

this approach are discussed in the next section.

Figure 2.1: Example of a Markov chain for an e-commerce system

To the best of this author's knowledge, the only workload generator described in the literature that can generate session-based workloads with controllable characteristics is GEIST [33]. The tool employs the aggregate workload generation approach. First, the trace generation engine of GEIST creates a trace of request arrival instants that exhibits the desired arrival process characteristics. GEIST provides statistical models that allow control over properties of the marginal distribution of the time between successive requests and the correlation between successive request arrivals (e.g., self-similarity, multifractality).

Entries in GEIST's trace refer to time instants at which requests need to be submitted. The successive arrival instants in the trace are assigned equiprobably to one of K *sender threads*. Each sender thread now has information regarding the times at which requests

must be issued but does not know the identity of the requests. GEIST employs a user-specified first-order Markov chain to assign request names to request arrivals in each thread. During request generation, the sender thread issues a request at the specified time instant. A separate receiver thread independently handles the response for this request. Therefore, the sender thread is free to issue the next request at the next specified time instant irrespective of whether or not the response to its previous request has been completely received by the receiver thread. This allows the tool to achieve the arrival process desired. The paper describing GEIST [33] presents a case study for a Web server system with static objects to illustrate the tool's usage but does not provide a similar example with a session-based system.

The authors of GEIST list several limitations of the tool. In its current form, GEIST does not support HTTP POST requests, and mechanisms for handling user abandonment and retries of requests. Also, fine control over user-behavioural aspects cannot be realized due to aggregate workload generation and consequently features such as persistent connections, pipelining, and cookies are not supported. These and other limitations of the tool are discussed further in the next section.

Apart from the research tools reviewed, there are also several commercial tools [38, 53] for stress testing session-based systems. However, to the best of this author's knowledge these tools only address request generation issues and do not provide any automated trace generation capabilities. Typically, stress 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.

To summarize, this section reviewed existing stress testing techniques for session-based systems. Very few methodologies exist for session-based systems, which is surprising given the increasing predominance of these systems on the Web. Many existing studies have proposed first-order Markov chains as a mechanism for handling inter-request dependencies. GEIST is the only general-purpose workload generation tool that is intended for session-based systems. It relies on a first-order Markov chain to ensure validity of sessions and uses the aggregate workload generation approach. The next section describes the limitations of the approaches reviewed here and motivates the need for the techniques proposed in this thesis.

2.2.3 Limitations of existing tools

This section discusses the limitations of the techniques reviewed in the previous section for stress testing session-based systems. Section 2.2.3.1 focuses on the problems arising out of using the first-order Markov chain approach for generating traces. Section 2.2.3.2 describes the other limitations. A brief overview of the solutions proposed is provided in Section 2.2.3.3.

2.2.3.1 Limitations due to the Markov chain based approach

As mentioned in the previous section, Menasce *et al.* proposed a technique to construct a first-order Markov chain based on sessions captured in a system's access logs [44]. The previous section also reviewed GEIST, a workload generator for session-based systems, that also uses a first-order Markov chain to ensure validity of sessions [33]. However, this technique suffers from several limitations which are discussed next.

1) Inability to fully address inter-request dependencies - First-order Markov chains assume that transition to a state depends only on the current state. In general, transition to a state may depend on the previous states as well. For example, consider the following two valid sessions [Home, View, Add, View, Add, Delete, Purchase], and [Home, View, Add, Purchase] for an e-commerce system. The first session describes a shopper who adds two products to the shopping cart, then deletes one of the products from the shopping cart before purchasing the other product. In contrast, the second shopper adds a product to the shopping cart and purchases that product. The first-order Markov chain constructed based on these sessions is shown in Figure 2.2. Note that the transition from Exit to Home is used to signify the start of a new session. This chain would incorrectly deduce a dependency between Delete and Purchase based on the first session, without recognizing that the Purchase resulted due to one of the two previous Add states. As a result, the Markov chain of Figure 2.2 will yield a session (i.e., [Home, View, Add, Delete, Purchase]) where a shopper invokes Purchase without any item in the shopping cart thereby violating inter-request dependencies for the system. Higher-order Markov chains could be used to address this deficiency. For example, a chain that has the state [Add, View, Add, Delete] can ensure that Purchase is invoked only when there is an item in the shopping cart. However such a technique is not described in the literature.

2) Lack of flexibility – Apart from governing the sequence of requests in sessions, the Markov chain also determines the mix of request types and the session length distribution. As discussed in Section 2.1.2, studies of real session-based workloads have suggested that the characterizations of these attributes may influence system

performance. It is therefore important to provide capabilities that would allow stress test teams to achieve arbitrarily desired request type mixes and session length distributions (e.g., to match a workload observed at a particular system). However, it follows from the fundamental theorem of Markov chains [59] that a given first-order Markov chain will yield a unique workload mix and session length distribution². Consequently, changing the transition probabilities of the chain to control the workload mix will also affect the session length distribution. Similarly, changing the transition probabilities to control the session length distribution will affect the workload mix as well. Therefore many different Markov chains, each employing different state spaces, must be deduced for studies that need to vary workload mix and session length distributions independently of one another. This author is not aware of any techniques that deduce such chains.

Figure 2.2: Example of an incorrect Markov chain

²This is true if the chain has a finite number of states and is ergodic (i.e., every state in the chain can be reached starting from all other states). Both these conditions are expected to be valid in general for session-based systems.

2.2.3.2 Other limitations

This section discusses the limitations of GEIST since it is the only publicly available tool intended for session-based systems.

As described in Section 2.2.2, the objective of GEIST is to issue requests at pre-determined time instants to achieve a desired arrival process at the system. The tool employs many concurrent sender threads, which act as “virtual” sessions, for issuing requests to the system under study. It delegates the responsibility of receiving responses to separate receiver threads so that the sender threads are free to issue their next requests at their pre-determined time instants. In other words, it is possible to submit a request in a session even before the response to the previous request has been received. This can cause problems with session-based systems for a number of reasons described next.

Firstly, inter-request dependencies may be violated if the system starts processing the n^{th} request in a session before the $(n-1)^{\text{th}}$ request has completed. For example, considering Figure 2.1, Delete cannot be issued until Add has been processed completely and the system has updated the shopping cart state for the concerned session. Secondly, scenarios where the response of the $(n-1)^{\text{th}}$ request needs to be parsed to determine parameter values to be passed with the n^{th} request cannot be handled. Examples of such parameters include session identifiers, transaction identifiers, and HTTP cookies. The system cannot handle user requests in a session correctly if stale values of these parameters are passed with the subsequent requests, as is possible under such an aggregate workload generation approach.

As noted by its authors, GEIST is intended to model characteristics of the aggregate

request arrival process and not user-behavioural aspects. As discussed in Section 2.1.1 and Section 2.1.2, user-level characteristics such as the distributions of think time and session length can have an impact on system performance. It will not be possible to study the effects of these characteristics using GEIST.

Finally, the request generation architecture of GEIST may impose additional problems. GEIST splits the original workload trace among multiple workload generation client machines in a round-robin manner. The authors of the tool note that a more sophisticated mechanism may be needed for session-based systems to maintain inter-request dependencies. Also, due to the multithreaded architecture, the ability to issue requests at the correct instants may be hindered by operating system scheduling policies [46].

2.2.3.3 Proposed solution

The techniques proposed in this thesis address several of the limitations addressed in the previous section.

A trace generation algorithm is described in Section 4.3 that avoids problems associated with the Markov chain based approach. Given an input trace of valid sequences of request types for a system, the algorithm selects a compact subset of sequences for producing a synthetic workload trace with the correct inter-request dependencies and desired characterizations for a chosen set of workload attributes. The problem of selecting the subset of sequences is posed as a linear programming problem. The chief advantages of this approach are that the input trace can be deduced from Web access logs alone and the subset of sequences can be selected to match arbitrary workload mixes and distributions of session lengths.

The proposed technique allows the flexibility of using either a hybrid user-equivalents mode or an aggregate workload generation mode depending on the objectives of a study. For example, the hybrid user-equivalents mode can be used to investigate characteristics of user-level attributes such as think time and session length while the aggregate approach can be used to investigate the effect of correlations in the request arrival process. In *both* modes, a request in a session is submitted only after the response to the previous request has been completely received. This ensures that inter-request dependencies are strictly enforced and allows us to support session-based features such as those discussed in Section 2.2.3.3. Section 4.4 describes the workload generation modes.

Finally, the techniques proposed are layered on top of *httperf* [46], a highly accurate and scalable request generator. The design of *httperf* minimizes dependencies on the operating system which prevents the likelihood of poor context switching and scheduling decisions affecting the accuracy of workload generation. *httperf* uses non-blocking socket calls that provide the ability to generate and sustain overloads on a system in laboratory environments without having to dedicate a prohibitively large number of hardware and software resources. *httperf* also supports a comprehensive set of features such as persistent connections, pipelining, SSL, cookies, user abandonments and retries, and testing through proxy servers.

2.3 Summary and conclusions

This chapter reviewed existing work on workload characterization and stress testing of Web-based systems. Studies show that several attributes of traditional Web-based system such as file sizes, response sizes, and think times exhibit heavy-tailed behaviour. This

property is claimed to be one of the factors that contribute to the bursty or self-similar request arrival patterns observed consistently across several real workloads.

Analyses of several real workloads show that the self-similar property is also present for session-based systems. The studies suggest heavy-tailed session lengths, caused due to the presence of robots, heavy-tailed user think times, and high-variability in service times for requests to be probable causes for the observed burstiness in session-based workloads.

This chapter also reviewed synthetic workload generation tools for Web-based systems. The tools provide capabilities for systematically investigating the influence of several workload attributes believed to have an impact on performance. Surprisingly, very few tools were found that support stress testing for session-based systems. The limitations of these existing methodologies and tools were described. The solution proposed in this thesis that overcomes these limitations was described briefly.

To conclude, this chapter reviewed existing studies to identify several important attributes that can impact the performance of session-based systems. Existing methods for generating synthetic workloads lack flexible control over such attributes. Furthermore, existing workload generation tools do not properly support an aggregate workload generation method for session-based systems. Consequently, new methods are needed for generating synthetic workloads and submitting the requests to session-based systems.

Chapter 3

Workload characterization of a large Web-based shopping system

This chapter presents a workload characterization of a large e-commerce system [5]. The study considers several workload attributes and discusses the impact of their observed characteristics on the system's performance. Consequently, the results help in choosing the attributes that need to be considered while stress testing session-based systems.

Section 3.1 provides a description of the e-commerce system and the measurement data used. Section 3.2 provides a high-level summary of the workload data in terms of the types of responses served by the system. An analysis of request arrivals at the system is presented in Section 3.3. Section 3.4 analyzes session-level workload attributes. It presents empirical characterizations of session inter-request times and session length. Request-level attributes such as classes of requests, request popularity, and temporal locality are examined in Section 3.5. Section 3.6 provides summary and conclusions.

3.1 The system under study

This section is organized as follows. Section 3.1.1 describes the high-level architecture of the system under study. Section 3.1.2 describes the measurement data utilized for the study.

3.1.1 System description

The e-commerce system in this study is the online presence of one of the largest retailers in North America. It has a typical multi-tiered architecture as illustrated in Figure 3.1. The system hosts several hundreds of thousands of unique products. The name of the

retailer and details such as the number of products hosted by the site cannot be revealed in this study due to a non-disclosure agreement.

Figure 3.1: Logical diagram of the system under study

As shown in Figure 3.1, customer requests access the system via the Internet. Load balancers divide incoming requests across a tier of Web servers. The Web servers serve all requests for non-dynamic resources (such as static HTML files) and propagate requests for dynamic resources to application servers. Application servers make use of database servers to support information retrieval, personalization, and transactional order

management. The application servers prepare responses in HTML format for rendering on shopper browsers. These responses are then propagated to the customer via the Web server tier.

The system handles only the requests for the HTML portions of each Web page. All requests for the embedded graphics are provided by a separate “graphics” system. No workload data on the graphics system is available; as a result, it is not analyzed in this study.

Application servers can either dynamically generate HTML responses for shoppers or serve responses from an application server request cache. Each application server has its own request cache, as shown in Figure 3.1. Dynamically generating responses creates the opportunity for personalizing content for shoppers but can use significantly greater processing resources than serving responses from a request cache.

Dynamically generated responses are classified into two categories, *dynamic cacheable responses* and *dynamic uncacheable responses*, depending on whether they are served from the application server cache or not. A dynamic cacheable response is generated once by the application server, by executing queries and retrieving results from the database, and then stored in the application server cache. Subsequent requests requiring the same response are served by retrieving the content from the cache rather than regenerating the data from the back end subsystems. Responses containing details about a product are examples of responses belonging to this category. The system has mechanisms in place to invalidate cached responses when necessary (e.g., when a product’s details have been modified in the database). Creation of dynamic uncacheable responses, on the other

hand, always causes requests to the back end and database tiers; they typically include information such as order identifiers that can only be assigned dynamically.

Each user session is associated with a unique identifier in order to maintain state (e.g., shopping cart contents) between requests. The first request of a session does not have a session identifier and is defined as a *session id-less request*. A Web server that receives a session id-less request uses a load-balancing algorithm to choose an application server. It then forwards the request to the application server, which assigns a session identifier. Subsequent requests in the session may go to any Web server but are always directed back towards the application server that issued the session identifier.

In this study the initial session id-less requests are considered to be part of their sessions. However it is not possible to positively identify the initial request of all distinct sessions. Although the Web access logs do contain the IP address for the client that issued each request, there are a significant number of proxies present, which access the system on behalf of some number of real users. Therefore, client IP addresses could not, in general, be correlated with session identifiers.

When characterizing session lengths the initial session id-less request is compensated for by simply adding one to all reported session lengths. It is assumed that the time between this initial request and the second request in the session is equal to the mean of such times measured for all sessions. An analysis determined that these start of session requests accounted for most of the session id-less requests.

Finally, to conserve system resources, application servers timeout sessions (and reclaim their resources) after fifteen minutes of inactivity. All requests associated with an expired

session identifier receive a 'timeout' response. At this point the user may return to the home page to obtain a new session identifier, or may choose to depart.

3.1.2 Measurement Data

The measurement data utilized in this study was collected from four different sources. Three of these sources were part of the system under study. The primary source of workload data is the access logs, which were collected from each of the Web servers in the system. Additional workload information is provided by the access logs of each of the application servers. Supplemental performance data is available in the MeasureWare [62] logs. These logs describe server resource usage and are available for all of the servers in the site. The contents of these logs are described in more detail later in this subsection.

The fourth source of data was a test environment, which was a replica of the actual site (in terms of the hardware, software, and configuration), albeit on a much smaller scale. It had fewer numbers of load balancers, Web servers, application servers, and database servers than the actual site. This test environment enabled further characterization of the performance of various requests. For example, it was possible to obtain estimates for the mean CPU demands of important requests using exercisers (i.e., mini-benchmarks). Unless explicitly stated this data is not used when reporting on how the actual site was used.

Measurement data was collected from the actual site on two separate occasions. The first measurement period was on March 9th, 2000 (the March data set); the second measurement period lasted from midnight on July 21st, 2000 until midnight on July 25th,

2000 (the July data set). There were few changes to either the software or hardware configurations of the system under study between the two measurement periods.

The Web server access logs contain a single line of information for each HTTP request issued to the site. The recorded information includes the following: the IP address of the client that issued the request, the time at which the request was received (1 second resolution), the method (e.g., GET) to be applied to the resource, the name of the resource (i.e., the URL), a session identifier (for sessions already in progress), the HTTP protocol version and the HTTP response code.

The application server access logs contain a single line of information for each request handled by the application server. This information includes the response completion timestamp, the elapsed time for completing the response, a flag to indicate if the response was served from the application-level cache and the (application-level) name of the request type.

The MeasureWare logs provide a wide range of data on server performance. For this study the “global” metrics (aggregated over five minute intervals) reported for each server were utilized. The data used for the study includes, a timestamp for the start of the measurement interval, the mean CPU utilization over all the CPUs of a server, the peak disk utilization across all the disks of a server, the physical memory utilization of a server and the number of packets sent and received on all the network interfaces of a server.

Despite the vast amount of data that was collected, a lot of interesting information was not available. For example, the Web server logs do not contain any information on the size of request or response headers or bodies, nor do they include a measure of the time

required to complete a response. In addition, the timestamps for request arrivals have only a one-second resolution, which limits the accuracy of several of the reported analyses. The application server access logs do not include a session identifier that would allow matching the requests in the Web server access logs to those in the application server access logs.

Due to the non-disclosure agreement, absolute values (e.g., the number of requests in the measurement period) are not reported since these could be used to derive (confidential) information about the site's effectiveness in terms of marketing and merchandising (e.g., the number of customers visiting the site or the number of purchases made during the measurement interval).

3.2 Summary of Workload Data

This section provides a high-level summary of user requests in the e-commerce site. Each unique URL in the data is defined to refer to a unique object³ in the system. Objects are categorized into *object types* by examining their URLs. Two approaches are utilized to determine the object type. One approach examines the extension of the URL. For example, URLs ending in '.jpg' or '.gif' are placed in the *Image* category. The second approach searches for the presence of a parameter list (e.g., /example.html?parameter_list). All requests containing a parameter list are placed in the *Dynamic* category. Table 3.1 provides the breakdown of requests by object type.

Table 3.1 reveals that almost all requests (over 95% in both data sets) were for dynamic objects. Even after accounting for the absence of graphics, this represents a significant

³ The words object and response are used interchangeably in this chapter.

difference between this workload and traditional Web workloads. For the remainder of the study only the requests for dynamic objects are considered when analyzing the system workload.

Table 3.1: Breakdown of requests by object types

Object type	March 2000 (%)	July 2000 (%)
Dynamic	96.58	95.73
HTML	2.58	3.67
Image	0.37	0.16
Text	0.03	0.01
Other	0.44	0.43
Total	100.00	100.00

3.3 Analysis of request arrivals

The analysis presented in this section examines the overall pattern of request arrivals to the system during each of the two measurement periods. Figure 3.2 shows the arrivals at the system during each of the two measurement periods. The curve in each graph represents the request rate to the site during each minute of the measurement period. The request rate has been normalized with respect to the median request rate of the measurement period. Figure 3.2 (a) presents the system usage for a single day in March. This figure indicates that the workload follows a typical time of day pattern; i.e., the system is busiest during mid day, and least busy during the early morning. The peak request rate during this measurement period was 1.75 times the median request rate, and 7 times the lightest load, which occurred between 3am and 6am.

Figure 3.2 (b) provides the usage of the system during the July measurement period. This figure reveals that in addition to the time of day factors, the day of the week also affects the request rate to the site. For example, the system is busier on the weekdays than on the weekend. On the weekdays the workload is busier during the day than the evening, which

is still significantly busier than the early morning. On Saturday the request rate is quite consistent throughout the day and evening hours. On Sunday the evening hours are busier than the daytime hours. In all cases, the lightest loads occur in the early morning hours.

Figure 3.2: Traffic volume: (a) March 2000 (b) July 2000

In Figure 3.2 (b) the peak request rate was twice the median request rate and eight times the minimum request rate. While both data sets are quite consistent in terms of the peak to median ratios, it is important to note that there were no special events during either of these periods that would have caused a “flash crowd” (i.e., a sudden and significant increase in the number of user arrivals). Examples of special events that can affect e-commerce workloads are new advertising campaigns, special promotions (e.g., issuing coupons), or the approach of holidays such as Valentine’s Day, Easter, Mother’s Day, Father’s Day and Christmas.

From the results, there are significant differences between the request arrival rates observed at the system during different time periods. For instance, the request rate during the two measurement periods varied by up to a factor of eight. This motivates the need

for mechanisms that would allow a system's performance to be studied under many different request arrival rates of interest.

3.4 Characterization of sessions

This section describes two session-level characteristics for the system: inter-request time and session length. Session *inter-request time* is defined as the time between subsequent request arrivals within a session. The primary source of requests for large Web-based systems (such as the one under study) is users (i.e., consumers). Robots are another source of requests. Common uses of robots include determining a site's liveliness, measuring a site's responsiveness, and collecting a site's pricing information. It is important to distinguish these sources of requests from those of consumers in order to isolate their performance impact.

It is difficult to positively distinguish between robots and consumer sessions. While proper 'netiquette' requires robots to access the '/robots.txt' file to identify themselves to the site as being a robot (and to learn about the site rules for robots) [34], the workload data considered suggests that most robots do not access this file. While some robot implementers/users may be unaware of this expected behaviour, some may choose not to access this resource as numerous sites refuse (further) service to positively identified robots. A few robots can be identified by the user-agent information they provide, while others can be identified from their fully qualified domain name.

Unfortunately, these methods do not allow us to identify many of the (suspected) robots. For this study a simplistic assumption is used which states that robot sessions issue significantly more requests than do typical human users. In this study, sessions consisting

of more than 30 requests are deemed to be generated by a robot; shorter sessions are deemed to be human users. A detailed analysis found that 9% of those sessions characterized as robots performed "checkout" operations which are not usually associated with robot behaviour. These are false positives for this simple approach.

Sessions seen in the first or last hour of the (aggregated) Web server access log are discarded from the analyses to reduce the impact of end effects. A session timeout length of 15 minutes is used in the analyses. This is the same value that is used by the system under study. A timeout value is used to prevent a few, long inter-request times from skewing the results (e.g., the mean session inter-request time). Many of these long inter-request times are caused by robots that re-start their "crawl" of the site. As a result of using this timeout value, there are a number of sessions that get fragmented in the analyses. Typically the fragmented portion of the session consists of only a single request. This occurs because the application server would issue a response requesting that the user restart its session (by clicking on a link to the home page, thereby acquiring a new session identifier). Thus sessions of length one are ignored.

Analysis of session inter-request times is considered in Section 3.4.1. The analysis of session length is presented in Section 3.4.2.

3.4.1 Inter-request time

Figure 3.3 (a) and Figure 3.3 (b) show the session inter-request time distribution for both users and robots in March and July, respectively. The inter-request times for each unique session are measured, and then the results are aggregated for all user and all robot sessions respectively.

Figure 3.3: Cumulative distribution of session inter-request times a) March b) July

The x-axis in Figure 3.3 uses a log scale in order to show the full range of values for the inter-request times. In both measurement periods the results are quite similar. For example, in March the mean inter-request time for users was 53.9 seconds and the median (i.e., the 50th percentile) was 28 seconds. In July the values were slightly lower, with a mean of 48.2 seconds and a median of 26 seconds. The inter-request times for robots were typically shorter than they were for users. In March the mean inter-request time for the robots was 38.9 seconds while the median was 22 seconds. In July the results were almost identical, with a mean of 40.4 seconds and a median of 22 seconds. In all cases the maximum inter-request arrival time was limited to 900 seconds.

As shown in Figure 3.3, the tail of the inter-request time distribution covers a wide range of inter-request times. For example, considering the July data set, 90% of the user inter-request times observed were less than 60 seconds. However, the rest 10% of the observations, constituting the tail, ranged from 60 seconds to as high as 900 seconds. Therefore there is a significant probability of observing very long inter-request times in the system.

Since the site did not host embedded images, the session inter-request times roughly correspond to user think times. As mentioned in Section 2.1, Crovella *et al.* state that increased probabilities for long think times (i.e., heavy-tailed think times) can cause the arrival of requests at the system to become more bursty [17]. Consequently, to achieve the appropriate burst behaviour during a stress test, careful attention must be paid to the choice of the think time distribution. For example, considering the case of stress testing the system under study, using a distribution that associates a smaller probability for long think times than that of Figure 3.3 may cause lower than a realistic degree of bursty behaviour thereby yielding overly optimistic estimates of the system's performance.

The issue of generating synthetic workloads with desired think time distributions is addressed in Section 4.3. The sensitivity of a session-based system's performance to changes in think time distributions is investigated in Section 5.4.

3.4.2 Session length

The next session-level attribute considered is the session length. Figure 3.4 shows the distribution of session length for both users and robots. In both Figure 3.4 (a) (March data set) and Figure 3.4 (b) (July data set) the x-axis uses a log scale in order to show the full range of values. The maximum number of requests in a user session is limited to 30 due to the assumptions regarding users and robots.

The nearly horizontal segments of the robot session length distributions shown in Figure 3.4 reveal that in spite of the assumption regarding robot sessions there are some robot sessions with *fewer* than 30 requests. This occurs for the following reasons. The analysis began by dividing all of the unique sessions into two sets (users and robots) based on the

number of requests that contained each distinct session identifier. Sessions with more than 30 requests were assigned to the set of robot sessions. The 900-second limitation on session inter-request times was then applied to all sessions in a particular set. As a result of this approach a single session can become fragmented.

Figure 3.4: Cumulative distribution of the session length: (a) March (b) July

With a well-behaved client the fragment would always be of length one; that request would direct the client to restart the session and obtain a new identifier. However, some robots are not well behaved, and instead continue to reuse an expired session identifier, even though the application server continually informs the client to restart the session and acquire a new session identifier. For this reason, some of the fragmented robot sessions can contain more than one request. Some of these fragments contain less than 30 requests which explains the nearly horizontal segments in the robot session length distributions (Figure 3.4). This is an example of a robot consuming resources at the site while not performing any useful service for the robot operator, as the response simply contains information on how to acquire a new session identifier.

Figure 3.4 shows that the number of requests per session for both users and robots is quite consistent across the March and July measurement periods. In March the average

number of requests for a user session was 8.0 and the median 6.0, while in July the mean was 7.14 and the median 6.0. For robot sessions the mean number of requests per session was 51.0 in March and 52.6 in July, while the median in both data sets was 42.0. In March the maximum number of requests in a robot session was 473; in July it was 3,312 (recall that the measurement period in July was 4 times longer than in March).

Combined (though not illustrated in the figure), the session length distribution has two modes leading to a long tail. This is due to the presence of robots. For example, considering the users and robots in the July data set together, while 90% of sessions had a session length of less than 88, the remaining 10% had session lengths ranging from 88 to 3,312. As mentioned in Section 2.1, similar behaviour was also noticed in the session-based systems studied by Menasce *et al.* [43]. This observation has important consequences for stress testing. Firstly, large session lengths result in sessions that last longer. Consequently, for a given rate of arrival of sessions to a system, a workload with a longer session length tail will see more of an overlap among sessions than a workload with a shorter tail. The increased overlap can in turn lead to increased contention among sessions for system resources, thereby degrading performance. Secondly, it is not uncommon for an application server to perform garbage collection or other housekeeping operations at the termination of each session. In such cases, workloads with larger session lengths cause lower session management overheads when viewed as amortized on a per request basis.

It follows from the above discussions that it is important to consider the session length distribution while stress testing session-based systems. Section 4.3 addresses the problem of generating synthetic workloads with the desired session length distribution

while Section 5.4 explores the sensitivity of a session-based system's performance to changes in the session length distribution.

3.5 Characterization of requests

This section characterizes request-level attributes and discusses their impact on the system's performance. Section 3.5.1 presents a measurement study on a testbed system similar to the system under study. The study is used to classify user requests into *request classes* based on their resource demands. Section 3.5.2 characterizes the distribution of requests to unique objects (URLs) in the site. An analysis of temporal locality is presented in Section 3.5.3.

3.5.1 Request classes

This section groups user requests for dynamic objects into three distinct request classes based on their significantly different resource requirements. The three classes of requests are cacheable, non-cacheable-search, and non-cacheable-other. The *cacheable* class of requests represents requests for the dynamic cacheable responses described in Section 3.1.1. Of the requests for dynamic uncacheable responses, *non-cacheable-search* requests let shoppers search for products by keyword. The rest of the requests for uncacheable responses are classified under the *non-cacheable-other* request class. Requests for cacheable responses that are not in the application server cache cause the dynamic generation of responses. These are defined as *request cache misses*. They are different from non-cacheable requests which are never cached.

From the MeasureWare logs, it was found that the application server CPUs were the bottleneck for the system under study. Bottlenecks limit the throughput of a system [29].

For this reason the request classes are characterized by their application server CPU demands. However, the logs from the actual site did not contain these demand values. Moreover, the operation of the actual site could not be interrupted for measuring the demands. Consequently, the demands obtained from a testbed system that is a scaled down replica of the actual site were used as estimates for the analysis presented in this section. As mentioned in Section 3.1.2, this system had the same hardware, software, and configuration settings as the actual site but had fewer numbers of servers in its tiers.

Estimates of the request class demands were provided by those running the testbed environment. These demand values were obtained based on the use of exercisers and MeasureWare CPU demand reports from the testbed system. The exercisers were used for submitting requests belonging to a particular request class to the system. The MeasureWare report for the exerciser's execution period was then used to obtain the mean application server CPU demand for the request class. The measured CPU demand does not include the time spent waiting for I/O operations or any other remote access; it is time spent consuming CPU resources alone. The demand values measured from the testbed system were found to be consistent with elapsed time measurements for the actual system during low load conditions.

Table 3.2 lists the ratios of application server CPU demands for the request classes with respect to the cacheable class CPU demand. For the system under study, dynamically generating a cacheable response requires approximately 500 times the application server CPU demand of serving a response from the request cache. This is due to the use of interpreted scripts and illustrates the performance cost of request cache misses.

Mechanisms that integrate fragments of cached pages [45] aim to reduce the impact of such misses but they were not implemented in the system under study.

Table 3.2: Relative mean application server CPU demands for request classes

Request class name	Ratio of application server CPU demand
Cacheable (response can be cached)	1 (cache hit)
Cacheable (response can be cached)	500 (cache miss)
Non-cacheable-other	100
Non-cacheable-search	40

The wide range of the resource demand ratios suggests that the request class mix of the workload has a significant impact on the average application server CPU demand per request. *Request class mix* specifies the fraction of total requests that belongs to each class. The remainder of this section characterizes the request classes for the data sets and demonstrates the sensitivity of system performance to request class mix and request cache hit rate.

Table 3.3 gives a breakdown of requests by request class for the March and July data. The table shows that the request class mix remained relatively stable over the five-month period.

Table 3.3: Request class mix for March and July data

Request class name	March 2000	July 2000
Cacheable (response can be cached)	73.30%	70.53%
Non-cacheable-other	15.89%	17.94%
Non-cacheable-search	10.81%	11.53%

A simple utilization analysis demonstrates the sensitivity of the maximum application server throughput (capacity) to the mix of requests for the three classes. Utilization analysis (for a single resource) exploits the utilization law [29] that states:

$$U = X D \quad (3.1)$$

where U is the utilization of a resource, X is the number of completions per time unit, and D is the demand in time units per completion. Since U has a maximum value of 1 per resource the maximum value for X is determined by D . For a given U , as D increases the maximum number of completions per time unit decreases. Conversely, as D decreases the maximum number of completions per time unit increases. Calculating the ratio of the demand from different time intervals quantifies the change in system capacity:

$$\text{Change in capacity} = \frac{D_{old}}{D_{new}} \quad (3.2)$$

As an example, suppose the system has the request class mix for the July data given in Table 3.3. Assuming no cache misses, multiplying the proportions of request classes by the corresponding ratios of average application server CPU demands given in Table 3.2, the average CPU demand D per request is 23.3 times that of a cacheable request. Let the percentage of cacheable requests increase by 5% accompanied by a 2.5% decrease to both non-cacheable-search and non-cacheable-other. Computing the product of these new proportions and the demands of Table 3.2, the average application server CPU demand per requests decreases from 23.3 to 19.8 times that of a cacheable request. Similarly a 5% decrease in cacheable requests accompanied by 2.5% increases to both non-cacheable-search and non-cacheable other causes the average CPU demand per request to increase to 26.9 times that of a cacheable request. Therefore, substituting D_{old} with 23.3 and D_{new} with 19.8 and 26.9 in Eq. (3.2), a swing of $\pm 5\%$ in cacheable requests with differences applied equally to non-cacheable-search and non-cacheable-other causes the capacity of each application server to vary by a factor of 1.18 to 0.87.

Next, the effect of higher request cache misses is considered. Let the July mix of Table 3.3 be perturbed to create mixes with an increase of 1%, 4%, and 8% in request cache misses (i.e., cacheable requests that cause responses to be generated). Taking the product of demands from Table 3.2 and their corresponding proportions in these mixes, the average CPU demands per request are 28.2, 43.2 and 63.2 times that of the average CPU demand of a request satisfied by the cache, respectively. Applying Eq. (3.2), for these cases the relative capacity per server decreases from 1 (for no misses), to 0.83, 0.54, and 0.37, respectively.

For a large numbers of users, increased request cache misses combined with a decrease in cacheable requests can lead to very high demands. For example, an 8% increase in request cache misses along with a 5% decrease in cacheable requests causes the mean application server CPU demand per request to increase to 66.6 times that of a cacheable request. From Eq. (3.2), the system capacity decreases by approximately 2.9 times when compared to the July mix scenario.

To summarize, for the system under consideration, the resource demand at the bottleneck device for a cache miss is 500 times that for a cache hit. Similarly, the demands for requests belonging to the non-cacheable-other and non-cacheable-search request classes are 100 and 40 times, respectively, the demand for a cache hit. Due to this wide fluctuation in demands, even small changes in the cache hit rate or the request class mix of a workload can have a relatively large impact on the demand at the bottleneck device and hence the system's performance (e.g., utilization, throughput, and response time). Though not all systems will have requests with such large ratios in demands, different requests may also exercise different parts of the system. This highlights the need to

consider the request mix while stress testing session-based systems. Section 4.3 describes a method to generate synthetic session-based workloads with desired mixes. Section 5.4 presents an experimental study that illustrates the impact of request mix on the performance of a session-based system.

Two workload attributes that are related to request class mix and cache hit rate are the distribution of requests to unique objects in the system, and temporal locality. These are discussed in the following subsections.

3.5.2 Distribution of requests to unique objects

In this section the distribution of requests to unique dynamic objects is studied. In particular, the popularity of unique dynamic objects in the site is examined. As defined in Section 3.2 a unique dynamic object has a unique URL. The URL consists of a request type part and a name-value list part, as described in Section 2.2.2. The relative popularity of unique objects determines the request class mix and hence influences the performance of the system.

As mentioned in Section 2.1.1, popularity of objects has often been characterized by a Zipf-like distribution in several previous studies [3, 7, 48]. To test if a distribution is Zipf-like, a log-log plot of the number of requests for each unique dynamic object in the system as a function of the rank of the object is created. The rank of an object is computed by assigning 1 to the object with the most number of requests, 2 to the next most frequently accessed object, and so on. If the distribution is Zipf-like, the plot should appear linear with a slope near $-\beta$, where β is called the Zipf exponent. As mentioned in Section 2.1, the value of β was found to range from 0.6 to 1.0 based on characterizations

of several real Web workloads. The closer the value of β is to 1, the more concentrated requests are to a few unique objects in the system [65].

Figure 3.5: Distribution of requests to objects: (a) March (b) July

Figure 3.5 (a) and (b)⁴ show the relative popularity of the most requested dynamic objects in the March and July data sets respectively. These plots indicate that the most popular objects in the system appear to follow a Zipf-like distribution. The estimates of the slope are $\beta=1.06$ for the March data set (Figure 3.5 (a)) and $\beta=1.04$ for the July data set (Figure 3.5 (b)). For both sets of data, the coefficient of determination (R^2) is computed to ascertain the degree of fit of the empirical distributions to the synthetic distributions. The closer the R^2 value is to 1 the better the fit [29]. In both cases the R^2 value is 0.99, which indicates a very good fit between the empirical and synthetic distributions.

The near-unity estimates of the Zipf exponent indicate that a very small number of objects in the site account for a significant fraction of all requests. Previous studies on systems whose objects are cacheable have shown that this property corresponds to good performance since it causes a system's caches to yield considerably lower cache misses for objects than otherwise [14]. This in turn suggests that significant performance gains

⁴ The x-axis and y-axis values are not shown in keeping with the non-disclosure agreement.

could be achieved through frameworks that cache dynamic objects in session-based systems.

To summarize, the distribution of requests to unique objects influences the request mix and the cache hit rate. Based on the discussions in the previous section, it follows that stress tests must support detailed modeling of this attribute to facilitate control over the resource demands placed on the system under study.

3.5.3 Temporal locality

As mentioned in Section 2.1.1, temporal locality is the property by which a recently requested object is more likely to be requested again in the near future. Characterizing temporal locality is important in the context of performance evaluation since it can impact the behaviour of a system's caches.

Jin *et al.* [30] and Cherkasova *et al.* [16] explain that temporal locality is caused by two distinct factors: 1) the long-term popularity of certain objects (i.e., “hot” objects); and 2) the short-term temporal correlations in requests to certain objects. The long-term popularity of objects causes significant temporal locality since requests to a highly popular object tend to occur close to one another in a request stream. The short-term component arises when successive requests to a particular object occur close to each other even though the object in itself is not very popular in the long-term. For example, such a situation can arise when certain objects become temporarily popular due to special promotional events initiated by the site. The long-term component can be characterized through the use of Zipf-like distributions, as discussed in the previous section. This section studies the extent of the short-term component in the workload.

The approach followed in [3, 13, 40] is used to characterize the extent of the short-term component. This approach exploits the well-known result from the operating systems area [19] which states that if i) the workload has only the long-term component and ii) the object sizes are the same, then the optimum page replacement policy is the Least Frequently Used (LFU) policy [6]. It follows from this result that a workload where the long-term component is predominant will give rise to lower cache misses with frequency-based policies such as the Least Frequently Used (LFU) [6] policy. Conversely, if the temporal locality in the workload is predominantly due to the short-term component, then lower cache miss rates are achieved with recency-based policies such as the LRU policy [56].

Trace-driven simulations of the application server cache are performed to compare the cache miss rates observed with recency-based and frequency-based policies for the system under study's workload. Three policies are considered. The LRU policy is considered as an example of a recency-based policy while the LFU[6] and LFU-Aging [6] policies are chosen as examples of frequency-based policies. The LRU policy evicts the least recently used object from the cache when the cache is full. The LFU policy maintains a running count of accesses to objects. The least frequently used objects become the candidates for replacement when the cache is full. The LFU-Aging [6] policy is similar to the LFU policy but it scales down the counts of objects at fixed intervals. This enables certain temporarily popular objects that accumulated high request counts, but which are not popular any more, to become candidates for replacement. It should be noted that the original system employed a fully primed cache and hence did not have to employ any page replacement policy.

The July workload data is considered for this study. Since temporal locality is relevant only for cacheable objects, a workload trace consisting purely of requests to dynamic cacheable objects is created. In terms of trace length, the trace considered is comparable to those used in the cache simulation study of [13]. The long trace length (4 days worth of data) ensures that the initial conditions will not affect the results significantly. Each entry of the trace is made up of the name of the object (i.e., its URL) and its size. The Web server access logs did not record the size of data transferred so all the objects are assumed to have the same size; a size estimate was given by the system administrators. This assumption is consistent with the objectives of this study. As mentioned previously, the aim is to characterize the extent of the short-term component of temporal locality by relying on the well-known performance behaviour of LFU under workloads that have same sized objects.

The simulation is performed as follows. The application server considers the requests in order from the trace. The application server cache is assumed to be initially empty. When a request is received, the application server searches for the requested object in its cache. A *cache hit* occurs when the object is present in the cache; otherwise a *cache miss* is recorded. When there is a cache miss, the application server generates the response by calling the back end systems and stores a persistent copy in the cache. Some of the existing objects in the cache may need to be removed in order to accommodate new responses. The performance metric of interest is the *cache miss ratio* of the application server cache, which is the ratio of the number of cache misses to the total number of requests in the trace. For each case of the simulation, the cache capacity is increased from that required to store a single object to that required for storing all the unique

objects in the trace. Finally, the LFU-Aging policy scales down the object counts by a factor of two after approximately every 2.75% of total requests has been serviced. Other parameters were also tried and they yielded similar trends in the results.

Figure 3.6: Cache miss ratios with LRU, LFU, and LFU-Aging policies (July data)

Figure 3.6 compares the cache miss ratios experienced by the application server cache with the three cache replacement policies. The x-axis shows the cache size normalized with respect to the size required to fit all the unique objects⁵. It can be observed from the figure that the LRU policy results in *higher* cache misses than LFU and LFU-Aging for very small cache sizes. LFU-Aging, which favours both recency and frequency, performs marginally poorer than LFU at very small cache sizes. All three policies yield almost the same cache miss ratios for higher cache sizes. The same trend was observed

⁵ The actual cache sizes could not be shown since it may be possible to derive the total number of unique products hosted in the site from the maximum cache size value.

while considering the July workload data on a day-by-day basis. Since the overall cache performance is better with the frequency-based policies than with LRU, it can be concluded that the short-term temporal locality component is not significant in the workload.

The result presented in the previous paragraph has an important consequence for stress testing. Since the short-term component is negligible, it is adequate to consider the long-term popularity component alone while modeling the workload of the system. This means a distribution which captures the long-term popularity of objects (e.g., a Zipf-like distribution), is sufficient for generating synthetic workloads for the system. Furthermore, based on their analysis of several real Web proxy workloads, Mahanti *et al.* have shown that even when the short-term component is strong, its performance-impact is significant only for systems whose cache sizes are very small (considerably smaller than the sum of the sizes of the unique objects in the workload) [40]. Though small cache sizes may not be uncommon in Web proxy caches, most real session-based systems, such as the one being studied, typically employ large memory and disk-based application server caches. These caches can accommodate a large fraction of unique objects, and thereby diminish the effect of short-term temporal locality. For example, based on the estimates of page sizes obtained from the site under consideration, it only takes a few tens of GB to store the hundreds of thousands of cached pages expected to be served by real systems. Consequently, distributions that merely capture the popularity of unique objects in the system may be adequate for stress testing realistically sized session-based systems.

3.6 Summary and conclusions

This chapter presented an analysis of workload observed on a real e-commerce system during 5 days spread over March 2000 and July 2000. The study discusses the qualitative impact of the observed workload characteristics on the performance of the system. This in turn provides insights into the workload attributes that need to be considered for stress testing session-based systems in general.

The arrival of requests to the system was found to exhibit time-of-the-day and day-of-the-week patterns. For example, the rate of arrival of requests was more around mid day than during the early morning. Similarly, the activity at the site was significantly more during weekdays than during weekends. For both measurement periods, the peak request rate was eight times that of the minimum request rate. Consequently, it is crucial to provide support for controlling request rates during stress tests.

The study also revealed that the session inter-request time distribution displays a long tail. Previous studies [17,10] have shown that such behaviour can degrade the performance of a Web server system by causing requests to arrive in bursts. Consequently, careful consideration must be given to the session inter-request time distribution during stress tests.

An interesting feature of the workload was the presence of robots. Robots are autonomous software agents which are used to gather information such as a site's responsiveness and pricing information. A previous study by Menasce *et al.* had also revealed the existence of robots in the workloads of two different e-commerce systems, which seems to suggest that robots are a ubiquitous phenomenon in such environments

[43]. Robots typically issue a large number of requests per session and hence have the effect of lengthening the tail of the session length distribution. Longer session length tails can adversely impact performance since they can cause increased contention among sessions for a system's resources. As a result, this thesis argues that stress tests must support control over the session length distribution to observe the impact it can have on performance.

For the system under study, the application server CPUs constituted the performance bottleneck. A study was conducted to measure application server CPU demands for requests belonging to three request classes: cacheable, non-cacheable-search, and non-cacheable-other. For the cacheable request class, the CPU demands were characterized separately for a dynamic object served from the application server cache and for a dynamic object that was generated on the fly. The measurements show that the resource demands for the request classes differed significantly. Also, the application server CPU demand to service a cache miss was 500 times than that needed to service a cache hit. The wide variation in measured resource demands implies that the performance of the system is very sensitive to variations in request class mix and application server cache hit rate. This suggests the importance of including workload attributes that influence these quantities while stress testing session-based systems.

The distribution of requests to the unique dynamic objects in the site was found to approximately follow the Zipf's law. The Zipf exponent estimated is very close to 1 which indicates that a large proportion of the requests were directed to a small subset of "hot" objects.

Simulation-based analysis revealed that short-term temporal locality is not significant in the workload considered. It is expected that short-term temporal locality is unlikely to influence the performance of realistically sized systems that do not have severe constraints on cache sizes. As a result, it is believed that distributions that merely capture the long-term popularity of objects will in general be adequate for stress testing session-based systems.

The next chapter describes a tool for generating synthetic workloads to stress test session-based systems. The tool uses a compact and flexible model to represent complex session-based workloads. It implements techniques to create synthetic workloads based on the model, and to submit them in a controlled manner to a system under study. The characterizations used in the model can be chosen to incorporate characteristics observed in real workloads into the synthetic workloads used in stress tests. The tool also permits sensitivity analyses on the characteristics of attributes within the model. This feature allows the impact of workload changes on system performance to be studied in a systematic manner.

Chapter 4

The Session-based Web Application Tester

Chapter 3 justified the selection of certain workload attributes for stress testing session-based systems. Based on the discussions in Chapter 3, in order to obtain reliable results from stress testing exercises the characterizations used for these attributes must be well chosen and must reflect the real workload expected at the system. Unfortunately, such characterizations are rarely known with certainty. A sensitivity analysis that perturbs these characterizations can reduce the risk of incorrect conclusions with regard to stress testing exercises. However, for session-based systems, handling inter-request dependencies and simultaneously controlling the characteristics of many workload attributes in an ad-hoc manner is difficult. Since setting up experiments and managing stress tests already pose significant challenges, practitioners are often left with little time to focus on this aspect of testing.

This chapter describes Session-based Web Application Tester (SWAT), a workload modeling and generation tool for stress testing session-based systems. SWAT is built on top of the open source *httperf* Web request generator [46]. It has features to deduce a compact and flexible representation of complex session-based application workloads, control the generation of synthetic workload traces for sensitivity analysis exercises, and to efficiently submit synthetic workload traces to a system under study. SWAT permits the characterizations of one or more workload attributes to be varied at a time so that the individual and joint impact the changes have on a system's performance can be observed. In this way SWAT can be used to increase confidence in the results of stress testing exercises.

The chapter is organized as follows. Section 4.1 presents a high-level overview of SWAT. The workload model employed by SWAT is presented in Section 4.2. The trace generation engine and the request generation engine of SWAT are described in Section 4.3 and Section 4.4, respectively. Section 4.5 provides summary and conclusions.

4.1 Overview

Following the approach commonly adopted in literature, SWAT splits workload generation into an offline trace generation process and a request generation process. Figure 4.1 shows the components of SWAT. The workload model consists of a chosen set of workload attributes that are likely to affect the performance of session-based systems. Statistical characterizations can be specified for each attribute in the model. The characterizations can be varied to emulate a variety of workload conditions including those observed at real systems. The trace generation engine produces a synthetic trace of sessions with the correct inter-request dependencies and the desired characteristics for the attributes in the workload model. Finally, SWAT uses a modified version of the open-source *httperf* workload generator to submit the sessions in the trace to a system under study.

Figure 4.1: Components of SWAT

4.2 Workload Model

As discussed in Section 2.2, hybrid user-equivalents and aggregate workload generation are two different approaches for submitting requests to a system under study. Recalling from Section 2.2, the choice between the two depends on the objectives of the study. Aggregate workload generation is useful for studies that require explicit control over the characteristics of attributes such as the distribution and correlation structure of inter-arrival times between successive requests to the system. For example, this method can be used to recreate the exact inter-arrival time sequence observed at a real system (e.g., as illustrated in Figure 3.2). Alternately, various sets of synthetic inter-arrival time sequences can be used to establish the relationship between the timescale of correlations in the request arrival process and system performance. The hybrid user-equivalents method does not offer this flexibility but is better suited for understanding the impact of user behaviour on system performance. This method supports emulation of user-level details such as think times, requests for embedded objects, and the HTTP 1.1 pipelining mechanism [27]. It is also useful for studies that require the notion of distinct user classes [44, 5].

SWAT provides the flexibility of employing either a hybrid user-equivalents session mode or an aggregate workload generation approach, which is referred to in this thesis as request mode. In the session mode, new sessions are generated according to a session inter-arrival time (defined later) distribution. Each generated session behaves as a user by issuing a request, waiting for the complete response from the system, and then waiting for an inactive period (think time) before issuing the next request. The inactive periods are chosen according to a think time distribution while the number of requests per session is

governed by a session length distribution. Finally, also included as attributes are the workload request type mix and the name value list distributions used within requests.

Request mode causes requests to arrive according to a specified request arrival process. The request arrival process is specified as a desired sequence of request inter-arrival times (defined later). In general, the workload attributes for request mode are the same as for session mode. However, session inter-arrival time and think time distributions are not specified. They are a consequence of the request arrival process. The implementation of the workload generation modes is described in Section 4.3. The rationales for choosing these workload attributes and their relationships to SWAT are discussed below.

Session inter-arrival time – *Session inter-arrival time* is defined as the time between successive arrivals of new sessions at a system under study. Mean session inter-arrival time is chosen to specify a session throughput for the system. Session mode follows an open model [29] for session arrivals. This causes the arrival rate of new sessions to be independent of system responsiveness. In other words, an overloaded system does not slow down the arrival of sessions. As with most real systems, this permits the number of concurrent sessions to change over time. This approach is in contrast to the “closed” pure user-equivalents approach [10, 41, 58] that emulates a fixed number of users. As described in Section 2.2, with the closed approach session throughput is affected by the responsiveness of the system instead of being specified directly.⁶

Session length – It can be recalled from Section 3.4.2 that there are two important reasons for modeling session length. Firstly, the session length distribution can influence

⁶ A closed approach with a very large number of user equivalents can approximate an open system but may require a large workload generation infrastructure.

the contention among different sessions for a system's resources. Secondly, session length can also influence the session management overheads incurred at the system.

Think time – As discussed in Section 2.1.1 and Section 3.4.1, it is important to model think time distribution since it influences the burstiness of a workload.

Request inter-arrival time– *Request inter-arrival time* is defined as the time between successive request arrivals at the system. Request inter-arrival time sequences are used in the request mode for studying the performance-impact of characteristics of the aggregate request arrival process. Request arrivals follow an open model. Consequently the arrival rate of requests is determined entirely by the specified sequence of request inter-arrival times and is independent of existing system load. Algorithms to derive inter-arrival time sequences exhibiting various correlation characteristics are discussed in [33, 52, 63]. As with the session mode, the number of concurrent sessions changes over time in the request mode.

Workload mix – Workload mix is defined as the overall proportions of the different request types in the workload. Different request types can pose significantly different resource demands upon the system. For example, from Section 3.5.1, the CPU demand at the bottleneck resource for a request type belonging to the “search” request class is 40 times that of a request type belonging to the “cacheable” request class. As a result small changes in workload mix can have a big impact on system utilization levels and response time distributions.

Name-value list distributions – For each request type, the corresponding name-value list strings are characterized. Later when generating loads, it can be ensured that these are

replayed with the appropriate distribution. The name-value list distributions can be used to control the locality properties of name-value pairs (e.g., products, keyword search terms). The generation of name-value lists is discussed in more detail in Section 4.3.

Some workload attributes not included in the model and the reasons for their exclusion are now discussed. In particular short-term temporal locality, response sizes and embedded objects are considered.

Short-term temporal locality - Section 3.5.3 showed that short-term temporal locality was not significant in the workload considered. Furthermore, recalling from Section 3.5.3, Mahanti *et al.* [40] have shown that short-term temporal locality has a pronounced impact on performance only for systems that employ small caches with respect to the maximum required cache size. From this author's experience, real session-based systems tend to have very large caches (physical memory caches of several hundreds of megabytes and disk caches of several tens of gigabytes). Short-term temporal locality is unlikely to have much impact for such systems. For systems where this assumption does not hold, the SWAT workload is likely to cause more cache misses than normal resulting in more conservative response time measures than otherwise.

Response sizes - Response size is not included as an attribute in the model. In general, each instance of a URL (request type, name-value list) can be associated with a fixed response size. Hence, the response size distribution of SWAT workloads is determined by the workload mix and name-value distributions specified. It follows that a representative distribution of response sizes can be achieved by using a representative workload mix and representative name-value list distributions.

Embedded objects – The SWAT workload does not include requests for objects such as image and multimedia files that are embedded within the HTML responses of the system. This is because such content is often hosted on external “graphics” servers, as with the system considered in Chapter 3, or at content delivery networks [35]. However, it is straightforward to introduce such requests into the workload if necessary. For example, the request generator can be made to parse such requests from the HTML response and then submit them to the system.

Finally, SWAT does not recommend any “representative” or default characterizations for the attributes in the workload model. It instead enables tests that demonstrate the sensitivity of a system’s capacity to changes in the characterizations used for the attributes.

Summarizing, SWAT can generate synthetic workloads according to either a user-equivalents-like session mode or an aggregate workload generation mode, called the request mode. The workload model used by SWAT consists of a set of workload attributes whose parameters can be specified during stress tests. Workload mix, name-value lists, and session length are attributes common to both modes. Session inter-arrival time and think time are attributes used in the session mode while request inter-arrival time is an attribute for the request mode. The next section describes the trace generation engine of SWAT. Its purpose is to create a synthetic workload trace with the desired characteristics for the model attributes.

4.3 Trace generation

The objective of trace generation is to produce a trace of sessions that can be submitted to stress test a system under study. Two important problems have to be addressed during trace generation. Firstly, to be valid, the sessions in the trace must follow the correct inter-request dependencies for the system. Secondly, the trace must match the desired characteristics for the workload model attributes. Section 4.3.1 discusses how SWAT ensures the correctness of sessions. Solutions for matching the desired characteristics are described in Section 4.3.2. Section 4.3.3 summarizes the trace generation approach.

4.3.1 Creating valid sessions

Section 2.2.3 discussed the limitations of the first-order Markov chain based approaches described in literature for handling inter-request dependencies within sessions. SWAT adopts an alternate approach that uses sessionlets to enforce inter-request dependencies. A *sessionlet* is a valid sequence of request types for a given system under study. For example, the sequence of requests in a session with an e-commerce system may correspond to the following sequence of request types.

```
[View_Home_Page, View_Product, Add_Product_To_Shopping_Cart,  
View_Product, Display_Shopping_Cart, Purchase_Product]
```

This sequence of request types is a sessionlet for the system. A sessionlet differs from a session in that it does not contain name-value lists. The trace generation engine of SWAT takes as input a trace containing many such sessionlets for the system under study and produces an output trace of sessions for the stress test. To ensure the validity of sessions, SWAT ensures that each session in the output trace has the same sequence of

request types as one of the sessionlets in the input trace. As a result, every session in the output trace is associated with exactly one sessionlet in the input trace.

The input trace of sessionlets required for the technique can be extracted in a straightforward manner from logs of real user activity. For example, it can be obtained from the access logs of a “live” version of the system under study by grouping request types based on the session identifier and sorting the request types in the groups in chronological order of their occurrence. In the absence of such logs, valid sessionlets can be created for a system by, for example, using a browser and mimicking the expected request type sequences of real users. Valid sessionlets can also be obtained from test cases used for the functional testing of a system. The proposed approach can then be applied upon such sessionlets to obtain a workload with the specified characteristics.

4.3.2 Matching desired workload characteristics

Figure 4.2: Schematic diagram of trace generation process

The trace generation process is divided into two steps as shown in Figure 4.2. Step 1, described in Section 4.3.2.1, employs an iterative algorithm that uses linear programming (LP) to take an input trace of sessionlets and produce an intermediate output trace of sessionlets that exhibits the desired workload mix and session length distribution. Step 2, described in Section 4.3.2.2, transforms sessionlets in the intermediate trace into sessions to produce a trace of sessions for the stress test. This involves selecting name-value lists for request types to form URLs and, for the session mode, inserting think times between successive URL requests in a session. Finally, the trace of sessions along with either the desired session inter-arrival time distribution (for session mode) or the desired request inter-arrival time sequence (for request mode) are input to *httperf*.

4.3.2.1 Workload mix and session length distribution

The basic objective of this step is to identify a subset of sessionlets from an input trace of sessionlets and compute playback counts for sessionlets in the subset in order to match the desired workload mix and session length distribution. The *playback count* of a sessionlet is defined as the number of occurrences of that sessionlet in the intermediate trace of Figure 4.2. This section presents a more formal statement and solution of this problem.

Problem statement

Let the system under study support K request types and let $\mathcal{S}$ be an input trace containing N sessionlets for the system. Each sessionlet i , for $i=1$ to N , in $\mathcal{S}$ can be associated with a $K \times 1$ *request type counts vector* $\mathbf{C}_i$ that records the K request type counts for that sessionlet. Let $C_i(k)$ represent the number of times the k^{th} request type is requested by

sessionlet i . With this notation the trace of sessionlets $\mathcal{S}$ can be described as a $K \times N$ matrix $\mathbf{T} = [\mathbf{C}_1 \ \mathbf{C}_2 \ \dots \ \mathbf{C}_N]$.

Let $\mathbf{W}$ be a $K \times 1$ vector, denoting the desired workload mix, such that $\sum_{k=1}^{K} W(k) = 1$. $W(k)$ denotes the proportion of the total number of requests that should belong to the k^{th} request type. Let the desired session length distribution be specified by means of a histogram with O bins as follows:

$$\mathbf{L}_1 = [\underline{L}_1, \overline{L}_1] : P_1 : L_1, \mathbf{L}_2 = [\underline{L}_2, \overline{L}_2] : P_2 : L_2, \dots, \mathbf{L}_O = [\underline{L}_O, \overline{L}_O] : P_O : L_O$$

where $\sum_{i=1}^O P_i = 1$

In the above histogram, each *session length bin* $\mathbf{L}_i$ is a 3-tuple consisting of an interval $[\underline{L}_i, \overline{L}_i]$, a probability P_i , and a mean L_i . P_i denotes the probability that the length L of a session in the synthetic workload satisfies the inequality $\underline{L}_i \leq L \leq \overline{L}_i$. L_i denotes the average session length of sessions belonging to the interval $[\underline{L}_i, \overline{L}_i]$.

Finally, let N_s be the total number of sessions required in the synthetic workload. N_s should be large enough for the achieved workload characteristics to be close to the specified characteristics⁷.

Each histogram bin is handled separately. Thus, the problem is to find a combination of sessionlets from $\mathcal{S}$ that would give the desired mix, session length distribution, total number of sessions, and hence the desired overall request type counts for each histogram

⁷ Statistical tests such as the chi-square test [29] and the Kolmogorov-Smirnov test [29] can be used to verify whether a given N_s is adequate for achieving the specified mix and distributions.

bin $\mathbf{L}_i$. Let $\mathbf{B}_i$ be a $K \times 1$ vector that gives the desired overall request type counts for $\mathbf{L}_i$. $\mathbf{B}_i$ is obtained as follows⁸:

$$\mathbf{B}_i = N_s P_i L_i \mathbf{W} \quad (4.1)$$

The trace generation problem reduces to determining for every $\mathbf{L}_i$ a linear combination of sessionlets chosen from $\mathcal{S}$ that results in the desired $\mathbf{B}_i$ and L_i . More specifically, assume that M_i such sessionlets can be chosen for every $\mathbf{L}_i$ from the N sessionlets in $\mathcal{S}$. Let $\mathbf{A}_i$ be a $K \times M_i$ matrix whose columns are the request type counts vectors of these sessionlets. Let $\mathbf{X}_i$ be a $M_i \times 1$ vector whose elements are the playback counts desired for the sessionlets described by $\mathbf{A}_i$. Then, the trace generation problem is to determine for every $\mathbf{L}_i$ an $\mathbf{A}_i$ and a $\mathbf{X}_i$ such that the following conditions are satisfied:

$$\mathbf{A}_i \mathbf{X}_i = \mathbf{B}_i \quad (4.2)$$

$$\frac{\sum_{m=1}^{M_i} X_i(m) \sum_{k=1}^K A_i(k, m)}{\sum_{m=1}^{M_i} X_i(m)} = L_i \quad (4.3)$$

$$X_i(m) \geq 0 \quad \forall m \quad (4.4)$$

The left hand side of Eq. (4.2) denotes a matrix multiplication. Eq. (4.2) specifies that the request type counts achieved by combining the sessionlets in $\mathbf{A}_i$ according to the playback counts in $\mathbf{X}_i$ should equal the desired request type counts given by $\mathbf{B}_i$. Eq. (4.3) states that the average session length obtained by combining $\mathbf{A}_i$ according to $\mathbf{X}_i$ must be equal to the desired value given by L_i . Eq. (4.4) restricts the computed sessionlet playback counts to be non-negative values. An algorithm for finding such sessionlets and playback counts is discussed below.

⁸ Eq. (4.1) can also be generalized to allow different $\mathbf{W}$ (mix) vectors for different session length bins.

Solution

An algorithm is given that iteratively searches for an $\mathbf{A}_i$ matrix and $\mathbf{X}_i$ vector that satisfy the conditions given by Eq. (4.2) to Eq. (4.4). At each step of the iteration, linear programming (LP) is used to find a value of $\mathbf{X}_i$ for a given $\mathbf{A}_i$ such that the difference between the desired request type counts ($\mathbf{B}_i$) and the achieved request type counts ($\mathbf{A}_i\mathbf{X}_i$) is minimized. Eq. (4.3) and Eq. (4.4) form two of the constraints of the LP problem. The third constraint is obtained by relaxing the condition specified by Eq. (4.2) to

$$\mathbf{A}_i\mathbf{X}_i \leq \mathbf{B}_i \quad (4.5)$$

Eq. (4.5) states that request type counts achieved by combining the sessionlets in $\mathbf{A}_i$ according to the playback counts in $\mathbf{X}_i$ should be less than or equal to the corresponding desired request type counts given by $\mathbf{B}_i$. In other words, every element of the vector $\mathbf{A}_i\mathbf{X}_i$ should be less than the corresponding element in $\mathbf{B}_i$. As described next, this change facilitates an iterative solution by which $\mathbf{A}_i$ is progressively modified until an $\mathbf{X}_i$ that satisfies the stricter constraint given by Eq. (4.2) (i.e., corresponding to a better match between $\mathbf{B}_i$ and $\mathbf{A}_i\mathbf{X}_i$) is found.

The algorithm consists of the following steps carried out for each $\mathbf{L}_i$:

- 1) To begin the iterative process, an initial $\mathbf{A}_i$ matrix is determined. This is achieved by identifying a compact subset of sessionlets from the input trace $\mathcal{S}$ and forming the $\mathbf{A}_i$ matrix corresponding to the subset.

As mentioned previously, the basic idea behind the algorithm is to represent the synthetic workload as a linear combination of sessionlets. To reduce the number of special cases that need to be handled through system-specific URL formation rules (Figure 4.2), a

minimum number of sessionlets should be chosen from $\mathcal{S}$. Cases that may require such rules are described in Section 4.3.2.2. Additionally, the sessionlets chosen should be distinct from one another in terms of their request type counts so that their combination is capable of yielding counts for all request types. The algorithm uses the concept of an algebraic basis set to identify an initial minimal subset of distinct sessionlets. The *algebraic basis set* of a set of vectors (spanning a linear space) is a minimal subset whose elements are linearly independent [37]. The basis set has the property of containing no redundant vectors. Linear combinations of the vectors in the basis set can be used to construct all vectors in the original set.

As per this approach, the algorithm first considers a set T_i whose elements are the request type counts vectors of sessionlets in $\mathcal{S}$ belonging to the session length bin $\mathbf{L}_i$. It then computes the basis set of T_i . From the definition in the previous paragraph, any request type counts vector in T_i can be constructed as a linear combination of request type counts vector present in this basis set. Finally, $\mathbf{A}_i$ is constructed as a matrix whose columns are the request type counts vectors in the basis set. It must be noted, from the definition of a basis set, that the desired request type counts can be achieved through a linear combination of the sessionlets in the basis set alone. However, in order to satisfy Eq. (4.3) (match of the average session length L_i) and Eq. (4.4) (no negative combinations) additional sessionlets may need to be added to the basis set. This is addressed in Step 6, which is discussed shortly.

2) $\mathbf{X}_i$ is computed using LP so as to minimize the difference between the desired and achieved request type counts, subject to the constraints given by Eq. (4.3) to Eq. (4.5).

3) If $\mathbf{A}_i \mathbf{X}_i = \mathbf{B}_i$, the conditions specified by Eq. (4.2) to Eq. (4.4) are satisfied and the algorithm terminates.

4) If $\mathbf{A}_i \mathbf{X}_i < \mathbf{B}_i$, a linear combination that would yield the desired request type counts could not be found while satisfying Eq. (4.3) and Eq. (4.4). The difference between the desired request type counts and the achieved request type counts is calculated as a $K \times 1$ *slack vector* $\mathbf{E} = \mathbf{B}_i - \mathbf{A}_i \mathbf{X}_i$.

5) The algorithm terminates if the mismatches in request type counts (given by the elements of $\mathbf{E}$) are less than or equal to user-specified mismatch thresholds for request type counts or if a user-specified maximum number of iterations is reached.

6) If any of the elements of $\mathbf{E}$ is greater than the corresponding mismatch thresholds, the sessionlet that offsets $\mathbf{E}$ the most is identified from $\mathcal{S}$ and added to the sessionlets subset. The algorithm is repeated from step 2 using the $\mathbf{A}_i$ matrix for the new set.

For choosing a sessionlet that offsets $\mathbf{E}$ the most, the Euclidean distances between $\mathbf{E}$ and the request type counts vectors in $\mathcal{T}_i$ are computed. The Euclidean distance d between a request type counts vector $\mathbf{C}$ and the slack vector $\mathbf{E}$, is computed as

$$d = \sqrt{\sum_{k=1}^{K} (C(k) - E(k))^2}$$
. The request type counts vector that yields the smallest d value is selected and appended to $\mathbf{A}_i$ as an additional column.

When the algorithm terminates, the sessionlets corresponding to the columns of $\mathbf{A}_i$ are written to an intermediate trace file. A sessionlet's playback count determines the number of times it gets repeated in the trace. The algorithm is guaranteed to terminate when the objective of trace generation is to match the workload mix and session length

distribution observed in the input trace $\mathcal{S}$. This is because in the worst case all sessionlets from $\mathcal{S}$ will be included to achieve the match. The algorithm may not converge exactly for other cases. This limitation along with possible solutions is discussed in detail in Section 4.5.

The formulation of the LP problem is summarized in Appendix A and empirical measures of the algorithm's time complexity are given in Appendix B.

Example

Figure 4.3: Comparison of achieved and desired workload mixes

Two examples are presented to illustrate the algorithm. An input trace of 100,000 sessionlets obtained from a system that supports 14 request types is considered. For both examples N_s is specified to be 100,000. For the first example the algorithm is applied to match the workload mix and session length distribution observed in the trace. The algorithm selected 1,779 sessionlets from the trace and computed playback counts for them. Figure 4.3 (a) shows that the workload mix achieved by repeating these sessionlets according to their playback counts is very close to the desired workload mix. A chi-

square goodness of fit test [29] showed that the achieved mix is statistically the same as the desired mix at a significance level of 0.05.

For the second example, a different workload mix is specified to the algorithm while keeping the session length distribution unchanged. The mix from the first example is perturbed such that the requests for request types 1, 2, 4, 5, 6, 9, and 10 are distributed equally among request types 7, 11, 12, and 13. For this example, the algorithm identified a subset of 3,238 sessionlets from the input trace. Figure 4.3 (b) shows that the achieved workload mix is close to the desired workload mix. The chi-square test showed that the achieved mix is statistically the same as the desired mix at a significance level of 0.05.

Figure 4.4: Comparison of achieved and desired session length distributions

Figure 4.4 compares the desired and achieved session length distributions for both examples. From the figure, there is a near-perfect match between the desired and achieved session length distributions for both examples. Aggregating the session length values into session length bins (in Figure 4.4) better illustrates the small differences

between the distributions. The Cumulative Distribution Function (CDF) plots were indistinguishable from each other. Kolmogorov-Smirnov goodness of fit tests [29] showed that the CDFs obtained for both examples are statistically indistinguishable from the desired CDF at a significance level of 0.05.

To summarize, this section presented an algorithm for generating an intermediate trace of sessionlets. The algorithm takes as input a trace of sessionlets from the system under study. Each sessionlet represents a valid sequence of request types for the system and is used to enforce inter-request dependencies. The algorithm identifies a compact subset of sessionlets from the input trace and computes playback counts for sessionlets in the subset. When the sessionlets identified are played back according to their playback counts, the resulting intermediate trace of sessionlets has the desired inter-request dependencies, workload mix, and session length distribution.

4.3.2.2 Name-value lists and think time distributions

This section describes how the sessionlets in the intermediate trace are transformed into sessions for stress testing the system under study.

The first step is to form URLs by selecting name-value list strings for request types in the sessionlets. As shown in Figure 4.2, the name-value list selection is driven by system specific URL formation rules. The cases that these rules must govern are discussed below.

Some name-value lists can be determined in their entirety during trace generation. For example, consider an e-commerce system. The product ids and product category ids in the system can typically be known beforehand. Such name-values can be generated

according to distributions specified in the workload model (e.g., based on those observed in access logs).

Some other name-value lists may have values that become known only during request generation. Considering an e-commerce example again, the unique order identifier that is to be passed along with a `Purchase_Product` request type is typically assigned dynamically by the system only when the session is in progress. The URL formation rules have to identify such instances to the trace generation engine. The trace generation engine in turn places a special tag, called a *placeholder tag*, in the concerned value fields that instructs *httpperf* to parse those values from the system's responses during request generation. This feature is named as *dynamic URLs* and its implementation within *httpperf* is described in Section 4.4.

Finally, for the session mode, think times are chosen from the desired think time distribution and inserted between successive URL requests in a session. An excerpt from a sample trace file is provided in Appendix C.

4.3.3 Summary

SWAT uses sessionlets, valid sequences of request types caused by real users of a system, to enforce inter-request dependencies. It employs a two-step trace generation approach to produce a synthetic trace of sessions that simultaneously displays the desired workload mix, session length distribution, think time distribution (for session mode studies), and name-value lists distributions. The trace of sessions along with either the desired session inter-arrival time distribution (for session mode studies) or the desired request inter-arrival time sequence (for request mode studies) are then input to *httpperf* for

stress testing the system.

4.4 Request generation

SWAT uses a modified version of *httperf* [46] to submit the sessions for the stress test. *httperf* consists of a core HTTP engine, workload generation modules, and a statistics collector. The HTTP engine is single-threaded process that uses non-blocking socket calls to send requests to and receives replies from the server under study. Non-blocking calls let the single-threaded HTTP engine emulate many concurrent sessions. The workload generation modules deal with the task of initiating URL requests at the appropriate time instants. The statistics collector module collects various performance metrics of interest during an experiment. An event-driven mechanism is used to coordinate the three parts in a mutually transparent manner. Finally, *httperf* supports the SSL protocol and HTTP cookies, which are often used by session-based systems.

The *wsesslog* workload generation module of *httperf* implements the session mode described in Section 4.2. The module starts sessions specified in an input file at intervals chosen according to a user-specified distribution of session inter-arrival time. For every session started, the module begins by initiating the first URL request. When it receives an event signaling the complete receipt of the reply, it sets off a timer based on the think time value that is specified after the request in the input session file. The module returns to the session when the timer expires and repeats the steps for subsequent requests in the session. *httperf* provides built-in support for exponential, uniform, and deterministic (constant) session inter-arrival time distributions. In order to support other more arbitrary distributions, an additional option was implemented which allows session arrival instants to be read from a user-specified file.

A new module named *wsessreq* was also added to *httperf* to support the request mode for session-based systems. Recalling from Section 4.2, request mode involves submitting requests at pre-determined time instants. However, there is a constraint that a request in a session cannot be issued before the response to the previous request has been completely received. Consequently, the initiation of requests from multiple concurrent sessions must be coordinated suitably to satisfy the specified sequence of request inter-arrival times. The algorithm that achieves this is described as follows.

wsessreq maintains a queue of inactive sessions, i.e., sessions not waiting for a response from the server. A timer is set off based on a request inter-arrival time chosen from the specified sequence of request inter-arrival times. When the timer expires, and if the inactive sessions queue is non-empty, the session from the head of the queue is chosen and the HTTP request corresponding to the current URL in the session is initiated. If the queue is empty, a new session from the trace is chosen and the HTTP request corresponding to the first URL in the session is initiated. When *wsessreq* receives the event corresponding to the complete receipt of an HTTP response, it identifies the session associated with the response. The session is added to the tail of the queue if it has more requests to be issued or terminated otherwise. This process is continued until all sessions in the trace are completed or the experiment duration is completed. In effect, *wsessreq* regulates the session inter-arrival times and the think times such that the desired sequence of request inter-arrival times is achieved. For this reason, as mentioned in the section 4.2, the user has no control over these two attributes while using this module.

It must be noted that the sessions that drive *httperf* are copied into memory prior to the execution of the experiment. This reduces the likelihood that advancing through the

sequence of sessions causes input-output delays that affect response time measures. In this author's experience, this approach did not cause significant memory requirements even for traces with a large number of sessions. Also, since *httperf* is single-threaded, the coordination of requests from multiple sessions in *wsessreq* does not pose any synchronization overheads. Overheads incurred for synchronizing multiple threads can make this scheme infeasible with workload generators that employ multiple threads to emulate concurrent sessions. Consequently, it is very difficult to support request mode studies for session-based systems with such workload generators.

httperf was also modified to support dynamic URLs introduced in Section 4.3.2.2. When *httperf* encounters placeholder tags in a URL, it extracts the actual values corresponding to the tags by parsing the response to the previous request and replaces the tags with the extracted values. *httperf* then issues the request corresponding to the resolved URL at the appropriate time instant. A last modification to *httperf* causes the statistics collector to report a trace and histogram of response times rather than just a mean value.

Finally, the maximum load that could be placed on a system can be increased by executing *httperf* simultaneously on multiple workload generation hosts. Before starting such tests, the inter-arrival time (request or session) values, and the sessions need to be distributed among the workload generation hosts. For example, successive inter-arrival time values, and sessions could be assigned to the hosts in a round-robin fashion.

4.5 Summary and conclusions

This chapter described SWAT, a workload modeling and generation tool for session-based systems. SWAT creates synthetic workload traces that simultaneously satisfy

inter-request dependencies and match desired characteristics for many other workload attributes. It employs a modified version of the highly scalable and efficient *httperf* Web request generator. *httperf* permits a high degree of workload generation accuracy and reproducibility of results.

SWAT addresses the limitations of existing workload generation tools arising due to the use of first-order Markov chains, which were described in Section 2.2.3.1. It uses a trace of sessionlets instead of the first-order Markov chain, which ensures that correct inter-request dependencies are maintained. SWAT also offers flexible control over the characterizations of attributes in the workload model to facilitate sensitivity analysis studies. The characterizations of one or more attributes can be changed simultaneously, without altering those of the others, so that the individual and joint impact the changes have on a system's performance can be measured. Sensitivity analysis is also helpful for understanding the impact of workload changes on system performance and for reducing the risk of incorrect conclusions due to a poor choice of characterizations.

Depending on the objectives of the study, SWAT provides the flexibility of employing either a hybrid-user-equivalents session mode or an aggregate request mode. As discussed in Section 2.2, aggregate workload generation tools, including GEIST [33] that are intended for session-based systems, submit requests without waiting for the response to the previous request. As discussed in Section 2.2.3.2, such an approach can cause problems since inter-request dependencies can be violated and dynamic URLs cannot be handled. However, SWAT ensures that a request in a session is submitted only after the previous response has been received even in the request mode thereby avoiding these problems.

The characterizations of all attributes used by SWAT can be estimated from Web server access logs. This facilitates the creation of synthetic workloads corresponding to real workloads recorded in these logs. Creating such workloads is difficult with the user-equivalents approach. As discussed in Section 2.2.1, the user-equivalents approach requires the number of concurrent sessions to be constant. Hence the session inter-arrival time distribution of the synthetic workload cannot be specified to be the same as that observed in the logs.

Finally, SWAT handles the tasks of maintaining inter-request dependencies, matching attribute characteristics, and dealing with dynamic URLs in an automated manner. In contrast, most (commercial) tools typically require stress test teams to manually code user emulation scripts to address these issues.

The current limitations of SWAT are as follows.

- 1) Some systems may not have access logs to obtain an input trace of sessionlets for SWAT. However, it is expected that an input trace can be easily created for a system by mimicking the request type sequences of its real users, based on information about the system's application logic. Also, as mentioned in Section 4.3.1, valid sessionlets can also be obtained from scripts used for functional testing of a system. SWAT can then use these sessionlets to generate workloads with desired characteristics.
- 2) An exact match to the desired workload mix and session length distribution may not always be possible with a given trace of sessionlets. The following approaches can be used in such cases.
 - a) The input trace can be augmented with additional sessionlets to obtain a better match.

For example, new sessionlets can be obtained by concatenating or truncating existing sessionlets.

b) The match of the workload mix can be relaxed to consider only those request types that are likely to have the greatest impact on performance. For example, the counts for resource intensive request types can be matched more closely than those for the less resource intensive request types. The degree of match between the desired and achieved workload mix can be controlled through the mismatch thresholds for request type counts, as discussed in Section 4.3.2.1.

c) The session length distribution can be matched more exactly in certain regions than others. The motivation for such an approach stems from previous studies that have argued that the regions of a distribution that contribute the larger values influence performance more than the other regions [10, 24]. The match over a particular session length region can be relaxed by specifying a histogram that employs wider session length bins for that region.

3) Extracting sessionlets from access logs may require preprocessing of the logs to cleanse sessions of erroneously submitted requests and to handle requests without session identifiers.

Chapter 5

Stress testing study of a TPC-W system

This chapter demonstrates the approach proposed in Chapter 4 with a stress testing study of a TPC-W-compliant [58] e-commerce system operating within a controlled environment. The objective of the study is multifold. First, the study validates the choice of workload attributes by showing that they can have an impact on the performance of session-based systems. Second, it demonstrates how the approach can be used to simultaneously control the characteristics of several attributes for evaluating a system's performance and effective capacity under various workload conditions. Last, the study explores how the characterizations used for the attributes influence performance and discusses the implications for stress testing exercises.

The chapter is organized as follows. Section 5.1 describes the system under study. The experiment methodology is described in Section 5.2. Section 5.3 validates SWAT for the study. Section 5.4 presents the results of the study. Section 5.5 summarizes and discusses the results.

5.1 System under study

An e-commerce system executing the TPC-W bookstore application [58] is chosen for the study. SWAT is used to submit controlled workloads to the system and observe the sensitivity of system performance to changes in the characteristics of workload attributes. Section 5.1.1 provides a brief description of TPC-W. Section 5.1.2 describes the TPC-W experiment setup.

5.1.1 TPC-W

TPC-W is a benchmark for e-commerce systems proposed by the Transaction Processing Performance Council (TPC), a consortium that includes industrial members such as Microsoft, Oracle, IBM, Sun, and HP. The benchmark is intended to facilitate the performance comparison of diverse e-commerce solutions regardless of the hardware and software platforms on which they are based.

The TPC-W application mimics the operation of an online bookstore. The TPC-W specification details implementation rules for the application. The bookstore supports fourteen request types – Home, Shopping Cart, Customer Registration, Buy Request, Buy Confirm, Order Inquiry, Order Display, Search Request, Search Results, New Products, Best Sellers, Product Detail, Admin Request, and Admin Confirm. Descriptions of their functionality can be found in [58].

The TPC-W specification groups the request types into two classes, *Browse* and *Order*. Section 5.2.1 of the TPC-W specification describes which request types belong to *Browse*, and which to *Order*. The specification defines three workload mixes based on these two request classes. These are 1) TPC-W *Browsing*, which consists of 95% *Browse* requests and 5% *Order* requests 2) TPC-W *Shopping*, which consists of 80% *Browse* requests and 20% *Order* requests and 3) TPC-W *Ordering*, which consists of 50% *Browse* requests and 50% *Order* requests.

TPC-W requires separate test runs to be carried out for each workload mix. The TPC-W specification provides a Markov chain that defines each of the three workload mixes. A

TPC-W test run uses a fixed number of user-equivalents called Emulated Browsers (EB). An EB generates sessions by traversing the Markov chain. Successive requests in a session are separated by a think time chosen from an exponential distribution with a mean of 7.0 seconds. The minimum duration of a session is chosen from an exponential distribution with a mean of 15.0 minutes. An EB starts a new session after the minimum session duration has elapsed and the Home state of the Markov chain has been reached. The maximum number of EBs is chosen such that the specified 95-percentile response times for the request types are met.

It must be noted that the experiments in this study are not TPC-W test runs. The TPC-W bookstore system merely serves as an example system for the study. The trace generation algorithms provided by TPC-W are used to produce an initial trace of sessionlets that is taken as an input for SWAT. The trace generation algorithm of SWAT is applied on this initial trace to create synthetic workloads with desired characteristics for attributes in the workload model.

5.1.2 Experimental setup

Figure 5.1 illustrates the experiment setup for the study. The 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. The switch provides dedicated 100Mbits/second connectivity to each node. It also realizes a “quiet” network that ensures the experiments are unhindered by network traffic from processes of other users.

The *client* node is a 300 MHz Pentium II machine with 256 MB of RAM running under the Linux (version 2.2.14-6.1.1) operating system. This node is exclusively dedicated to

executing *httperf*. All non-essential processes are turned off.

Figure 5.1: Experiment setup

The TPC-W application has a two-tier architecture. The *Web/application server* node is a 200 MHz dual-processor Pentium II machine with 1 GB of RAM running under the Windows NT Server (version 4.00.1381) operating system. It executes a Web server and an application server. The Web server consists of an HTTP listener process (*httpdL*) that forwards the requests from *httperf* to a user-configurable number of pre-spawned Web server threads. When a request involves the creation of dynamic content, the Web server threads communicate with a user-configurable number of pre-forked application server processes. The application server implements the TPC-W application and communicates with a remote TPC-W database. It creates the dynamic content and passes it to the Web

server thread that handles the request. The Web server thread in turn passes the response back to *httpperf*.

The database node is a 500 MHz Pentium III machine with 512 MB of RAM running under the Windows NT (version 4.00.1381) operating system. It executes the database server managing the TPC-W database. A pool of pre-spawned database server threads handles requests from the application server. The database server sets the number of threads in the pool to be twice the number of application server processes forked.

The Web, application, and database servers are part of a commercial off-the-shelf e-commerce product. The name of the e-commerce product has been withheld due to a non-disclosure agreement with the vendor. The choice of machines was limited to the ones that were available on the quiet network. Numerical values of results will likely depend on the specific hardware and software platforms on which systems are based. However, qualitative insights regarding the performance-impact of the various workload attributes considered in this chapter are expected to be valid for session-based systems.

Finally, a performance monitoring utility which collects a user-specifiable set of performance measures from both server nodes at regular user-specified sampling intervals is employed. The utility is implemented using the Win32 performance data helper API.

5.2 Experimental methodology

For this study experiments are conducted only in the session mode. The study evaluates the sensitivity of system performance to the following attributes: session length distribution, think time distribution, and workload mix.

The primary performance metric of interest are the user-perceived mean and 95-percentile of *response time* for the requests at the TPC-W system. These metrics are of interest for system sizing, capacity planning, and service level management exercises. Response time is defined as the time between sending the first byte of a HTTP request and receiving the first byte of the corresponding HTTP response. The measured response time is a good indicator of the delay suffered by the request at the TPC-W system, provided network latencies are negligible and the client workload generator node is not saturated.

Table 5.1 shows the configuration settings for the system. The goal of this study is to isolate the impact of high-level workload characteristics such as the session length distribution, think time distribution, and the workload mix. For this reason, it is desirable to keep the workload behaviour at the lower HTTP-level the same across all the workloads studied. The choice of settings for the following parameters reflects this requirement. The HTTP 1.1 protocol, which employs persistent TCP connections, is used for all experiments. The maximum number of requests accepted by the Web server per persistent connection is set to be greater than the largest session length encountered in the experiments. The maximum time allowed for the Web server to complete a request is set sufficiently high to prevent it from resetting connections when the system is overloaded. The persistent connection timeout, which is the maximum duration allowed for a persistent connection, is set to a very large value. The combined effect of the settings is to force a single connection to be used 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 differences in higher-level workload characteristics. Finally, as with most session-based systems, HTTP requests and responses are encrypted using the Secure Sockets Layer (SSL) protocol.

Table 5.1: Configuration settings for the TPC-W system

Parameter	Value
Protocol	HTTP 1.1 over SSL
Maximum number of requests to receive on a persistent connection	150
Persistent connection timeout (seconds)	3600
Maximum time allowed for the Web server to complete a request (seconds)	120
Number of Web server threads	1000
Number of application server processes	16
TPC-W scale factor	1000

The number of server processes and the threading levels are chosen so as to reduce the likelihood of starvation for processes and threads. The number of Web server threads is set to be 1000, which proved to be 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 *TPC-W scale factor*, defined as the number of unique books in the store database, is fixed at 1000. Larger database sizes were found to shift the system bottleneck from the Web/Application server node to the database server node. From experience, this is typically not the case with real systems since they usually employ database servers of greater capacity with respect to the Web/Application servers.

Finally, the performance monitor is configured to report CPU utilization, disk utilization, and memory paging activity for both server nodes. The sampling interval is chosen to be

60 seconds.

Table 5.2 lists the workload characteristics used for the study. Session arrivals are assumed to be uncorrelated which is consistent with previous studies (e.g., [50]). Consequently, the 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 server nodes.

Table 5.2: Workload characteristics

Attribute	Distribution
Session inter-arrival time	Exponential
Session length	Exponential, Empirical, and Bounded Pareto ($\alpha=1.16, k=3$)
Think time	Exponential, Empirical, and Bounded Pareto ($\alpha=1.10, k=12$)
Workload mix	HV-Mix and LV-Mix
Name-value lists	Uniform (except for dynamic URLs)

Three distributions are considered for session length and think time. These are *empirical*, *exponential*, and *bounded Pareto*. These are used to represent expected, best, and worst cases for variability, respectively. The empirical distributions are obtained from workload data collected from the real e-commerce system of Chapter 3. Since that system did not serve 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 exponential distribution is the distribution specified for TPC-W and is often employed by practitioners during stress testing exercises. The bounded Pareto distribution, a “heavy-tail-like” distribution, is used to study the impact of distributions that have a heavier tail relative to the empirical distribution.

A pure heavy-tailed distribution, such as the Pareto distribution, is not used since such experiments require prohibitively long runs to achieve the desired workload characteristics, on account of the very high variability of the distributions [18]. The bounded Pareto distribution is characterized by three parameters: α which governs the rate at which the tail of the distribution decays, k the smallest possible observation and p the largest possible observation. The probability density function, $f(x)$, for the bounded Pareto distribution is given by

$$f(x) = \frac{\alpha k^\alpha}{1 - \left(\frac{k}{p}\right)^\alpha} x^{-\alpha-1} \quad k \leq x \leq p$$

Though not strictly a heavy-tailed distribution, since all its moments are finite, the bounded Pareto distribution can still exhibit considerable variability when $k \ll p$ and as $\alpha \rightarrow 0$. The parameters of the bounded Pareto distributions are deduced as follows. The parameter p is set to the maximum observation obtained from the empirical distribution. The k and α values are then chosen such that mean of the empirical distribution is matched. From Table 5.2, the α values computed are 1.16 for the session length distribution and 1.10 for the think time distribution.

Table 5.3 shows the minimum, maximum, and mean of observations obtained with the distributions for the synthetic workloads used in the study. Table 5.3 verifies that all distributions result in the same mean session length and think time. However, it can be seen that the exponential distributions yield a smaller range of session length and think time observations compared to the empirical and bounded Pareto distributions. The bounded Pareto and the empirical distributions yield almost the same range of observations. However, their tails exhibit a different behaviour, as illustrated by Figure

5.2 which shows the log-log complementary distribution (LLCD) plots of observations from the three distributions.

Table 5.3: Session length and think time statistics

		Empirical	Exponential	Bounded Pareto
Session length (Requests/session)	Minimum	3	3	3
	Maximum	120	75	120
	Mean	9.44	9.44	9.44
Think time (seconds)	Minimum	0	0	12
	Maximum	900	595	900
	Mean	46.54	46.54	46.54

The LLCD plot is commonly used for studying tail behaviour of distributions and is obtained by plotting the complementary distribution (i.e., $P[X > x]$) of the observations (i.e., the x values) on log-log axes. Smaller slopes for LLCD curves at the tail region indicate heavier tails (i.e., higher probabilities for the larger observations). Accordingly, it can be inferred from Figure 5.2 (a) and Figure 5.2 (b) that the bounded Pareto distributions have the heaviest tails⁹ followed by the empirical and exponential distributions. In effect, the bounded Pareto distributions have larger numbers of very small and very large observations when compared to the exponential and empirical distributions.

Table 5.4 shows the workload mixes used in the study along with the mean “no-load” response times measured for each request type when the number of concurrent sessions is one. These no-load response times reflect the end-to-end resource demands across all resources for request types at the TPC-W system. The TPC-W Shopping mix [58] is named here as a high variation mix (*HV-Mix*). In addition, a slightly different mix with

⁹ This is partially violated in the think time region of 300 to 600 seconds due to the multi-modal nature of the empirical think time distribution.

Figure 5.2 (a): LLCD plots of session lengths

Figure 5.2 (b): LLCD plots of think times

Table 5.4: Mean no-load response times of request types and workload mixes

	No-load response time (seconds)	HV-Mix (Percentage of request types)	LV-Mix (Percentage of request types)
Home	0.09	16.00	9.00
New products	0.18	5.00	5.00
Best sellers	0.18	5.00	5.00
Product detail	0.23	17.00	27.80
Search request	0.07	20.00	20.00
Search results	0.13	17.00	17.00
Shopping cart	0.24	11.60	11.60
Customer registration	0.21	3.00	3.00
Buy request	0.63	2.60	0.00
Buy confirm	0.25	1.20	0.00
Order display	0.18	0.66	0.66
Order inquiry	0.05	0.75	0.75
Admin request	0.09	0.10	0.10
Admin confirm	0.14	0.09	0.09
Mean no-load response time (seconds)		0.16	0.16
Coefficient of variation of no-load response time		0.62	0.41

lower variation in demand (*LV-Mix*) is defined. To construct such a mix, the 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 and a corresponding increase to the proportion of requests belonging to the Product detail request type is caused, relative to the HV-Mix (see Table 5.4). Table 5.4 shows the mean no-load response time and the coefficient of variation of no-load response time computed for both mixes. It can be seen that the change effected causes the LV-Mix to have a lower coefficient of variation than the HV-Mix while maintaining the same mean for both mixes. As noted in Table 5.2, name-value list strings for requests are chosen according to a uniform distribution for both mixes except for dynamic URLs whose values can be determined only when the session is in progress (Section 4.3.2.2).

Finally, for this study, requests for images embedded in HTML responses are not submitted. Such content is often hosted on servers for image and multimedia files or on separate content delivery networks [5, 35]. This was the case for the system from where the empirical data was obtained. The empirical inter-request time distribution reflects user behaviour as influenced by such offsite images. Accordingly, the response time measures do not include the time to download such images. They represent only the portion of request response time due to the system under study.

To begin the process of Figure 4.2, a trace of 100,000 sessionlets from the TPC-W bookstore are created by following the trace generation algorithms provided in the TPC-W specification. This corresponds to the initial trace from the system under study. The trace generation technique of SWAT is applied to these sessionlets to create traces that exhibit the desired workload mix and distributions of session length, think time, and name-value lists.

For each experiment, four statistically independent replications are performed instead of a single longer run, to reduce the time lost due to experiment failures (e.g., Web server crashes). Each replication uses a different seed for each random number generator for each workload attribute. There are 10,000 sessions submitted in each replication. The duration of each experiment varied from approximately 12 hours to 15 hours depending on the mean session inter-arrival time and mix used. Each experiment yielded around 380,000 response time observations.

The mean response times (R_{mean}) reported are the average of the mean response time of individual replications. The cumulative distribution (CD) of response times is obtained

after merging response time observations from all replications. The 95% Confidence Intervals (CI) for the population 95th percentile of response time (R_{95}) are computed by applying the *distribution-free tolerance limits* technique [11] on the merged observations. This technique does not impose any restrictions on the observations (e.g., independence, normality) other than the continuity of the population distribution sampled. A disadvantage of the technique is that it requires storing all response time observations for an experiment.

The Web/Application server node CPUs were observed to be the system bottleneck for all workload cases considered. This study uses U_{WebCPU} to denote the mean utilization of the Web/Application server node CPUs as measured over a run's measurement intervals. The study imposed an upper limit on the mean session arrival rate considered in the experiments. The maximum value used for each workload case corresponded to an U_{WebCPU} of 77% over an experiment replication. With higher mean session arrival rates, it was observed that certain bursty workloads yielded sustained periods of very long response times which in turn caused the average number of request completions to drop significantly below the average number of request arrivals even over very long experiment durations.

To summarize, the main performance metric of interest for the study are the user-perceived mean and 95th percentile of response time for the requests handled by the TPC-W system. The study evaluates the sensitivity of the system's responsiveness to changes in session length distribution, think time distribution, and workload mix. Three distributions are considered for session length and think time: exponential, empirical, and bounded Pareto. The distributions have the same mean but are slightly different in terms

of their tail behaviour. In particular, the bounded Pareto distribution yields larger numbers of very small and very large observations when compared to the exponential and empirical distributions, causing higher variability. Finally, the study considers two mixes, HV-Mix and LV-Mix, exhibiting different variability in request type demands.

5.3 Validation

A stress testing tool can be validated by demonstrating that the system performance obtained with the synthetic version of a real workload is similar to the performance under the real workload. Such a study requires access to a production system for conducting the experiments. Since access to such a system was not available SWAT is validated using the TPC-W system of Figure 5.1. The objective of the validation is to establish confidence in using SWAT for the sensitivity analysis study. For the purpose of the validation two workloads are considered: an *Original* workload, and its SWAT equivalent, called the *Synthetic* workload. Both workloads are compared in terms of their statistical characteristics and the impact they have on the performance of the TPC-W system.

For the *Original* workload, a manually coded script that satisfies TPC-W's inter-request dependencies was used to create an offline trace of sessions having the empirical session length and think time distributions of the production system as shown in Figure 5.2. The sessions in this trace are then submitted to the system according to an exponential session inter-arrival time distribution using *httperf*. Such an approach was necessary since TPC-W does not provide such traces of user behaviour.

A detailed log of the requests submitted is collected from *httperf*. From this log, a trace of

sessionlets is extracted and the observed workload mix and distributions of session inter-arrival time, session length, think time, and name-value lists are constructed. Finally, the sessionlets along with the observed distributions and mix are input to SWAT to create the Synthetic workload.

Figures 5.3 (a) to 5.3 (d) compare the session inter-arrival time distribution, session length distribution, think time distribution, and workload mix of the Original and Synthetic workloads extracted from log files generated by *httpperf*. From the figures, it can be verified that there is a very close match between the two sets of workload characteristics. Moreover, chi-square and Kolmogorov-Smirnov goodness of fit tests [29] confirm that for both the workloads the achieved mix and distributions are statistically indistinguishable from the specified mix and distributions at a significance level of 0.05.

Figure 5.3 (a): Cumulative distribution of session inter-arrival time for Original and Synthetic workloads

Figure 5.3 (b): Cumulative distribution of session length for Original and Synthetic workloads

Figure 5.3 (c): Cumulative distribution of think time for Original and Synthetic workloads

Figure 5.3 (d): Workload mixes of Original and Synthetic workloads

Table 5.5: Comparison of Original and Synthetic workloads

	Original	Synthetic
Number of sessions generated	40,000	40,000
Number of unique request type sequences	838	420
Number of requests submitted	366,283	367,360
Mean number of replies received/connection	9.41	9.40
Mean reply size (KB)	6.48	6.47
$R_{connect}$ (seconds)	0.07	0.07
R_{mean} (seconds)	0.86	0.87
R_{95} (seconds)	2.93	2.98
N	340.91	339.11
N_{95}	386	384
U_{WebCPU} (%)	76.76	76.77
$U_{WebDisk}$ (%)	7.02	6.97
U_{DbCPU} (%)	32.65	31.84
U_{DbDisk} (%)	4.71	4.52

The Original and Synthetic workloads are submitted to the TPC-W system as per the network of hosts setup shown in Figure 5.1. Both the experiments took approximately 12 hours each to complete. Table 5.5 provides several descriptive statistics for both the

workloads. From the table, the number of unique request type sequences (sessionlets) is much lower for the **Synthetic** workload. This is because SWAT uses only a compact subset of the sessionlets derived from the **Original** workload for constructing the **Synthetic** workload (Section 4.3.2.1). However, this difference does not cause any significant change in the manner in which the workloads stress the system. The table shows that both workloads cause comparable utilizations for the system's devices. The mean utilizations of the Web server CPUs (U_{WebCPU}), the Web server disk ($U_{WebDisk}$), the database server CPU (U_{DbCPU}), and the database server disk (U_{DbDisk}) are almost the same with both workloads. Moreover, the mean number of replies received per connection is also almost the same verifying that the HTTP 1.1 persistent connection and the Web server connection timeout behaviours are the same for both workloads.

From Table 5.5, a close match is also observed for the means of response sizes, times to establish a TCP connection ($R_{connect}$), response times (R_{mean}), and number of concurrent sessions accessing the system (N). The 95th percentiles of response time (R_{95}) and the number of concurrent sessions (N_{95}) are also in close agreement for the **Original** and **Synthetic** workloads. Figures 5.4 (a) to 5.4 (c) compare the cumulative distributions of response size, response time, and number of concurrent sessions, respectively for the workloads extracted from *httpperf* logs. From Figure 5.4 (a), there is a slight mismatch between the response size distributions at the extreme tail. Similarly, as shown in Figure 5.4 (c), the distribution of number of concurrent sessions for the **Synthetic** workload deviates a little from that of the **Original** workload. However, as shown in Figure 5.4 (b), these deviations did not cause any significant difference between the response time distributions produced by the workloads.

Figure 5.4 (a): Cumulative distribution of response size for Original and Synthetic workloads

Figure 5.4 (b): Cumulative distribution of response time for Original and Synthetic workloads

Figure 5.4 (c): Cumulative distribution of number of concurrent sessions for Original and Synthetic workloads

To summarize, the synthetic workload generated by SWAT is representative of the original script-based workload it models. Firstly, there is a very close match between the statistical characteristics of the synthetic workload and the original workload. Secondly, both workloads result in nearly identical device utilizations and response time distributions.

5.4 Sensitivity analysis

This section presents results that illustrate the sensitivity of the system's performance to changes in session length distribution, think time distribution, and workload mix. The characteristics used for these attributes were described in detail in Section 5.2.

The following observations were consistent across all experiments. Chi-square and Kolmogorov-Smirnov tests [29] showed that the achieved workload characteristics (measured from *httpperf* logs collected from experimental runs) are statistically

indistinguishable from the specified workload characteristics. This verifies that the number of sessions generated was sufficiently large and that the client node was not saturated in the study. The worst-case mean network traffic generated by *httperf* during the experiments was 0.4 Mbits/second which constitutes only 0.4% of the maximum rated bandwidth of any of the Fast Ethernet switch ports. This indicates that network latencies were negligible. The disks at both server nodes were very lightly utilized. Virtually no memory paging activity was observed at either server nodes. No connections were reset by the Web server during the experiments. Finally, job flow balance was achieved for all experiments with the number of request completions equaling the number of request arrivals.

In general, the R_{mean} values are used for illustrative purposes only since their confidence intervals (CI) at a 95% confidence level overlap for many cases. The primary focus is on the R_{95} values. Quantitative comparisons between workloads are made with ratios computed from R_{95} values. For all the comparisons presented, the R_{95} values have non-overlapping confidence intervals at a 95% confidence level. In the worst case, the width of the confidence interval was found to be less than 3% of the corresponding R_{95} value.

Section 5.4.1 presents experiments to investigate the impact of session length and think time distributions. The impact of workload mix is studied in Section 5.4.2. Section 5.4.3 provides results that illustrate how variability in workload attributes affects effective system capacity.

5.4.1 Impact of session length and think time distributions

Section 5.4.1.1 compares the system's performance under the empirical and exponential distributions shown in Figure 5.2. The impact of the heavy-tailed distributions is studied in Section 5.4.1.2. Section 5.4.1.3 summarizes the results of this section.

5.4.1.1 Comparison of exponential and empirical distributions

For the experiment in this section, SWAT is used to create two different workloads, named *Empirical* and *Exponential*. They are both based on the HV-Mix and follow the empirical and exponential distributions of session length and think time shown in Figure 5.2, respectively. It may be recalled from Section 5.2 that the empirical distributions have a slightly heavier tail when compared to the exponential distributions. The mean session inter-arrival time for the workloads is chosen such that U_{WebCPU} is close to 77% which, as mentioned in Section 5.2, is the highest utilization value the system could be operated at without violating job flow balance. Both the workloads are submitted to the TPC-W system using the network of hosts setup shown in Figure 5.1.

Figure 5.5 (a) and Figure 5.5 (b) show the cumulative distribution of response time and number of concurrent sessions accessing the system for both workloads. From the figures, it can be observed that the cumulative distributions for the two workloads are practically indistinguishable from one other. Table 5.6 shows that the R_{mean} and R_{95} obtained with the *Exponential* workload are only slightly less than those obtained with the *Empirical* workload. Despite having lighter session length and think time tails, the *Exponential* workload seems to adequately capture the performance-impact of the *Empirical* workload. A possible explanation for this is provided in Section 5.4.1.3.

Figure 5.5 (a): Cumulative distribution of response time for Empirical and Exponential workloads

Figure 5.5 (b): Cumulative distribution of number of concurrent sessions for Empirical and Exponential workloads

Table 5.6: R_{mean} and R_{95} values for the Empirical and Exponential workloads

		Empirical	Exponential	R_{95} ratio
R_{mean} (seconds)		0.86	0.83	0.98
R_{95} (seconds)	95% CI high	2.93	2.86	
	Estimate	2.90	2.83	
	95% CI low	2.88	2.82	

5.4.1.2 Effect of heavy-tailed distributions

This section considers the impact of heavy-tailed session length and think time distributions on the response time distribution. The bounded Pareto distributions shown in Figure 5.2 are used for this purpose. The first experiment studies the impact of the bounded Pareto session length distribution. The next experiment investigates the influence of the bounded Pareto think time distribution. The last experiment establishes the combined impact of the bounded Pareto session length and the bounded Pareto think time distributions. All the workloads are based on the HV-Mix. The mean session inter-arrival time for the workloads is chosen such that U_{WebCPU} is close to 77%. Table 5.7 summarizes the results from these experiments. The workloads are submitted to the system using the network of hosts setup shown in Figure 5.1.

Table 5.7: Effect of the heavy-tailed session length and think time distributions

Workload	R_{mean} (seconds)	R_{95} (seconds)			$R_{95}(\text{Workload})/R_{95}(\text{Empirical})$
		95% CI High	Estimate	95% CI Low	
Empirical	0.86	2.93	2.90	2.88	1.00
Heavy-tailed session length	1.09	4.33	4.27	4.24	1.47
Heavy-tailed think time	0.96	3.44	3.40	3.38	1.17
Combined-heavy-tailed	1.14	4.02	3.96	3.93	1.37

Heavy-tailed session length workload

For this experiment, SWAT is used to create a *Heavy-tailed session length* workload where the session length follows the bounded Pareto distribution of Figure 5.2 (a). This is achieved by specifying the session length values obtained by sampling the bounded

Pareto distribution to the trace generation algorithm described in Section 4.3. All other characteristics of this workload are the same as the Empirical workload of the previous section. Figure 5.6 (a) and Figure 5.6 (b) compare the cumulative distribution of response time and the cumulative distribution of number of concurrent sessions for the Empirical and Heavy-tailed-session length workloads. From Figure 5.6 (a), one can observe that the Heavy-tailed session length workload yields smaller numbers of lower response time observations than the Empirical workload. From Table 5.7, the Heavy-tailed session length workload yields a higher R_{mean} and R_{95} than the Empirical workload even though the system operated at almost the same U_{WebCPU} (77%) for both workloads. The R_{95} value with the Heavy-tailed session length workload is 1.47 times that of the Empirical workload.

The reason for the degradation in response time can be explained as follows. As mentioned in Section 5.2, the bounded Pareto session length distribution will result in larger numbers of very long duration and very short duration sessions when compared to the empirical session length distribution. Since the session arrival rate is the same for both workloads, the Heavy-tailed session length workload can hence cause increased probabilities for observing very large and very small numbers of concurrent sessions at the system when compared to the Empirical workload, as seen in Figure 5.6 (b). 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 this case, be significant enough to cause periods of heightened contention for system resources during which requests incur very long response times.

Figure 5.6 (a): Cumulative distribution of response time for Empirical and Heavy-tailed session length workloads

Figure 5.6 (b): Cumulative distribution of number of concurrent sessions for Empirical and Heavy-tailed session length workloads

Heavy-tailed think time workload

Next, the impact of heavy-tailed think time distributions is considered. SWAT is used to create a *Heavy-tailed think time* workload which is similar to the Empirical workload in all respects except that the think times follow the bounded Pareto distribution shown in

Figure 5.7 (a): Cumulative distribution of response time for Empirical and Heavy-tailed think time workloads

Figure 5.7 (b): Cumulative distribution of number of concurrent sessions for Empirical and Heavy-tailed think time workloads

Figure 5.2 (b). Figure 5.7 (a) and Figure 5.7 (b) compare the cumulative distribution of response time and number of concurrent sessions, respectively for both workloads. From the figure, the tails of the response time and number of concurrent sessions distributions are longer with the Heavy-tailed think time workload. From Table 5.7, both R_{mean} and

R_{95} for this workload are greater than those of the Empirical workload confirming the findings reported in [10] that longer think times degrade the responsiveness of the system.

The reason for this behaviour can be explained by noting that heavy-tailed think times also cause long-lasting sessions causing a phenomenon similar to heavy-tailed session lengths. Noting that the mean session arrival rate is the same for both workloads, this leads to increased request bursts at the system that can degrade response time, as explained previously. However, from Table 5.7, comparing the R_{95} ratio of 1.17 for the Heavy-tailed think time workload with the corresponding ratio of 1.47 for the Heavy-tailed session length workload, it can be inferred that the impact of the heavy-tailed think time distribution is modest when compared to the impact of the heavy-tailed session length distribution even though both distributions have comparable tail indices (Table 5.2).

Combined-heavy-tailed workload

For the next experiment, SWAT is used to create the *Combined-heavy-tailed* workload where both the session length and the think time distributions follow the bounded Pareto distributions of Figure 5.2 (a) and Figure 5.2 (b), respectively. Figure 5.8 (a) and Figure 5.8 (b) compare the cumulative distributions of response time and number of concurrent sessions, respectively for the Empirical and the Combined-heavy-tailed workloads. As expected, the Combined-heavy-tailed workload lengthens the tail of both distributions. From Table 5.7, comparing the R_{mean} values for this workload and those for the Heavy-tailed session length and Heavy-tailed think time workloads with the R_{mean} value for the Empirical workload, it can be observed that the combined impact is near the sum of

Figure 5.8 (a): Cumulative distribution of response time for Empirical and Combined-heavy-tailed session length workloads

Figure 5.8 (b): Cumulative distribution of number of concurrent sessions for Empirical and Combined-heavy-tailed session length workloads

their individual impacts. The individual increases caused to R_{mean} by heavy-tailed session lengths and heavy-tailed think times are 0.23 and 0.10 seconds, respectively. The sum of these values (i.e., 0.33 seconds) is close to the 0.28 second increase to R_{mean} caused by the combination of heavy-tailed think times and heavy-tailed session lengths.

Figure 5.9: LLCD plots of response time for the Combined-heavy-tailed and Heavy-tailed session length workloads

From Table 5.7, it can be observed that the R_{95} value with the Combined-heavy-tailed workload (3.96 seconds) is marginally *less* than the R_{95} value with the Heavy-tailed session length workload (4.27 seconds). This seemingly counter-intuitive result is an artifact of the use of R_{95} to quantify the responsiveness of the system. To better compare the performance-impact of the two workloads, the LLCD plots of the response time observations shown in Figure 5.9 are used. Figure 5.9 shows that the Combined-heavy-tailed curve decays slower than the Heavy-tailed session length curve from the 0.0215 mark on the y-axis (denoting the 97.85th percentile¹⁰) corresponding to a value of 7000 ms on the x-axis. In other words, the Combined-heavy-tailed workload causes an increase to the probability of observing response times greater than 7000ms (or, in general, any value greater than that). The increased probability of observing very long response times verifies the fact that the Combined-heavy-tailed workload can place

¹⁰ The 95% confidence intervals for all percentiles up to the 99.99th percentile did not overlap for both the workloads.

more stress on the system when compared to the Heavy-tailed session length workload.

5.4.1.3 Summary

From the results of this section, it can be inferred that system performance can be sensitive to the distributions used for session length and think time. As the tail of the distributions lengthen, there is a simultaneous increase to the probabilities of observing very large and very small numbers of concurrent sessions in the system. This variability, in turn, causes requests to arrive at the system in a more uneven or bursty fashion. The increase in burstiness can lead to periods of increased contention for system resources during which requests incur very long response times.

From the results, one can infer that the degree of impact an increase in tail density has on the response time distribution depends on the extent of variability it is able to introduce in the number of concurrent sessions accessing the system. For example, from Figure 5.5 (b), the increase in tail density from the Exponential workload to the Empirical workload does not cause a significant change to the distribution of number of concurrent sessions. As a result, the R_{95} values are almost the same for both workloads. In contrast, the increase in tail density from the Empirical workload to the Combined-heavy-tailed workload introduces more variability in the number of concurrent sessions (Figure 5.8 (b)), causing R_{95} to increase by a factor of 1.37 (Table 5.7). It must be noted that the degree of sensitivity to the distributions also depends on system utilization. Previous studies have shown that the impact of bursts is more pronounced at higher system utilizations [49]. Section 5.4.3 presents experiments that investigate the relationship between the workload attributes and system utilization.

5.4.2 Impact of workload mix

The previous sections studied the impact of session length and think time distributions and found that they influenced the burstiness of the request arrival process. If significant enough, burstiness can cause increased system queue lengths leading to longer response times. The responsiveness of a system can also be affected if some requests in the workload require significantly more processing time than others [29]. This provides the motivation to study the effect of workload mix on the response time distribution.

To study the effect of workload mix, SWAT was used to generate an Empirical workload that exhibits the LV-Mix described in Section 5.2 instead of the HV-Mix. This is achieved by specifying the LV-Mix to the trace generation algorithm of Section 4.3. It may be recalled that the LV-Mix replaced certain infrequently occurring yet resource intensive requests in HV-Mix with less demanding requests. Based on the no-load response times given in Table 5.4, the LV-Mix has a lower coefficient of variation of no-load response time when compared to the HV-Mix. The session arrival rates chosen for both mixes are the same and cause the system to operate at an U_{WebCPU} of 77%. Table 5.8 shows the R_{mean} and R_{95} values for this case.

It can be observed from Figure 5.10 (a) that the LV-Mix yields a better response time distribution than the HV-Mix. From Table 5.8, the R_{95} value with the HV-Mix is 1.30 times that of the LV-Mix. From Figure 5.10 (b), it can also be seen that the cumulative distribution of the number of concurrent sessions exhibits a slightly longer tail for the HV-Mix than the LV-Mix. The result suggests that the higher variability associated with the HV-Mix due to the presence of request types with large demands relative to the mean

demand (e.g., Buy request) causes increased contention among sessions (Figure 5.10 (b)) and degradation in the responsiveness of the system (Figure 5.10 (a)).

Figure 5.10 (a): Cumulative distribution of response time for HV-Mix and LV-Mix (empirical case)

Figure 5.10 (b): Cumulative distribution of number of concurrent sessions for HV-Mix and LV-Mix (empirical case)

Table 5.8: R_{mean} and R_{95} values for LV-Mix and the HV-Mix (empirical case)

		LV-Mix	HV-Mix	R_{95} ratio
R_{mean} (seconds)		0.71	0.86	1.30
R_{95} (seconds)	95% CI high	2.26	2.93	
	Estimate	2.23	2.90	
	95% CI low	2.22	2.88	

The reason for this increased contention among sessions 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 HV-Mix workload gives rise to increased probabilities of observing very long duration and very short duration sessions. Noting that the mean session arrival rate is the same for both workloads, this leads to periods of increased contention among sessions as explained in Section 5.4.1.

The next experiment illustrates the interaction between workload mix, and session length and think time distributions. Specifically, the effect of mix when the distributions are empirical is compared to its effect when the distributions are heavy-tailed. Accordingly, SWAT is used to create an HV-Mix workload and a LV-Mix workload whose session lengths and think times follow the bounded Pareto distributions of Figure 5.2. Table 5.9 shows the R_{mean} and R_{95} values for this case.

Table 5.9: R_{mean} and R_{95} values for the LV-Mix and the HV-Mix (heavy-tailed case)

		LV-Mix	HV-Mix	R_{95} ratio
R_{mean} (seconds)		0.85	1.14	1.35
R_{95} (seconds)	95% CI high	2.98	4.02	
	Estimate	2.94	3.96	
	95% CI low	2.93	3.93

Figure 5.11 (a): Cumulative distribution of response time for HV-Mix and LV-Mix (heavy-tailed case)

Figure 5.11 (b): Cumulative distribution of number of concurrent sessions for HV-Mix and LV-Mix (heavy-tailed case)

Figure 5.11 (a) and Figure 5.11 (b) compare the cumulative distribution of response times and number of concurrent sessions, respectively for both the workloads. Comparing Figure 5.11 with Figure 5.10, the differences between the cumulative distributions of response time and number of concurrent sessions are more pronounced for this case than the previous case. The ratio of R_{95} for the HV-Mix to the LV-Mix increases to 1.35 for the heavy-tailed case (Table 5.9) from 1.30 for the empirical case (Table 5.8). This result shows that higher variability due to heavy-tailed session length and think time distributions can exacerbate the response time degrading effect of mixes having highly variable demands. Similarly, mixes with higher variability in demands can intensify the response time degradation due to heavy-tails. This can be seen by computing the ratio of R_{95} values of corresponding columns of Table 5.8 and Table 5.9. The ratio of R_{95} for the heavy-tailed workload to the empirical workload is 1.37 for the HV-Mix whereas it is only 1.32 for the LV-Mix.

5.4.3 Impact of workload conditions on system capacity

This section presents a system sizing case study using SWAT. The study is motivated by the work of Ranjan *et al.* [51]. The authors consider Quality of Service (QoS) driven server migration for horizontally scalable applications operating within utility computing environments. In such environments applications can acquire and release servers on demand. They present application level workload characterizations for large e-commerce and search engine sites and explore the effectiveness of an online algorithm called Quality of Infrastructure on Demand (QuID). The algorithm is an example of an application control system that decides when to acquire and release servers based on a target operating point for server CPU utilization. The authors show that QuID was able

to maintain a specific R_{95} while making efficient use of system resources by adaptively acquiring and releasing servers in response to changes in server CPU utilization. This section demonstrates how SWAT can be used to explore the relationships between workload conditions and the choice of such target operating points. As mentioned in Section 5.2, the Web/Application server node CPU was the system bottleneck for this study and consequently its utilization exerts the most influence on R_{95} when compared to the utilization of other system resources. For this reason, the operating point for the TPC-W system refers to a target U_{WebCPU} value.

Results from the previous sections show that the response time distribution obtained at a given U_{WebCPU} is sensitive to the session length distribution, the think time distribution, and the workload mix. Furthermore, the degree of impact of a given attribute depends on the characteristics of the other attributes. For example, as discussed in Section 5.4.2, the impact of the HV-Mix on R_{95} is more with the heavy-tailed session length and think time distributions than with the empirical distributions. Consequently, the target operating point chosen depends both on the target R_{95} desired and the characteristics of the workload attributes. To illustrate this, the R_{95} values obtained at various target utilizations under two extreme types of workloads are compared. The *Low-variability* workload has the LV-Mix and the empirical distributions of session length and think time whereas the *High-variability* workload has the HV-Mix and the bounded Pareto distributions for session length and think time. Table 5.10 shows the R_{95} values for the workloads when the system operates at three different U_{WebCPU} levels.

Let us assume that the TPC-W system has to deliver a R_{95} of 2.25 seconds or less to its users. From Table 5.10, with the Low-variability workload, the system could

comfortably operate at an U_{WebCPU} of 77% while satisfying this requirement since the R_{95} is only 2.23 seconds. However, from Table 5.10, the High-variability workload would yield a R_{95} of 3.96 seconds at the same operating point thereby violating the R_{95} requirement. In this case, from Table 5.10, the operating point has to be lowered to 70% to satisfy the R_{95} requirement. Repeating the exercise for a maximum R_{95} requirement of 1.60 seconds, from Table 5.10, the operating point for the Low-variability workload is 70%. However, the High-variability workload would cause the system to operate at a lower utilization (e.g., 64%).

Table 5.10: Effect of workload conditions on R_{95}

U_{WebCPU} (%)	R_{95} (seconds)	Low-variability	High-variability
77	95% CI high	2.26	4.02
	Estimate	2.23	3.96
	95% CI low	2.22	3.93
70	95% CI high	1.59	2.25
	Estimate	1.58	2.22
	95% CI low	1.57	2.21
64	95% CI high	1.18	1.48
	Estimate	1.16	1.46
	95% CI low	1.16	1.45

From the results, high variability due to heavy-tailed session length and think time distributions coupled with mixes characterized by highly-variable demands have the effect of forcing a system to operate at a reduced mean utilization level in order to satisfy a desired level of responsiveness. In this case, the High-variability workload causes a 7% drop in utilization to satisfy a R_{95} requirement of 2.25 seconds. From another perspective, since system throughput is directly proportional to the utilization of the bottleneck device [29], high variability in these attributes implies a reduction in effective system capacity.

5.5 Summary and conclusions

This chapter presented a stress testing case study of a TPC-W system. The study used controlled workloads generated by SWAT to investigate the sensitivity of system performance to changes in session length distribution, think time distribution, and workload mix.

Results presented in Section 5.4.1 show that the distributions of session length and think time can have an impact on a system's response time distribution. Based on the distributions studied, the results show that higher the variability of the distributions the poorer the system's responsiveness. This observation emphasizes the importance of choosing proper distributions for session length and think time. Using distributions with lower variability for stress testing when empirical evidence points to distributions with significantly higher variability can lead to overly optimistic performance estimates. In the absence of empirical data for a system under study, it may be necessary to conduct tests with a range of distributions with different degrees of variability.

Results from Section 5.4.2 illustrate that system responsiveness is degraded with workload mixes that cause variability in request demands. In this study, a 3.8% increase in resource-intensive requests (i.e., Buy request and Buy confirm) caused R_{95} to increase by up to 35% (Table 5.9) at a given utilization of the system's bottleneck device. From Table 5.4, the ratio of the maximum no-load response time to the minimum no-load response time is 12.77 for the TPC-W system studied. However, Chapter 3 showed that the ratio of resource demands for two different request types could be as high as 100 for a large session-based system. For such systems, the response time distribution can be

expected to be even more sensitive to swings in the workload mix. Consequently, careful consideration must be given to the choice of workload mix in stress tests.

The results also show that the performance impacts due to individual attributes interact. For example, the degree of impact due to higher variability in session length and think time distributions is greater with a high-variability mix than otherwise. As a result, the choice of target utilization for the system to satisfy a particular response time requirement can be very sensitive to the characteristics of workload attributes. For example, for the TPC-W system, the permissible utilization of the Web/Application server to satisfy a given 95-percentile of response time decreases when the variability of the workload increases. SWAT is useful for such system sizing studies since it can associate a range of operating points with a range of possible workload attribute characterizations and response time requirements.

Chapter 6

Summary and conclusions

This chapter summarizes the thesis, discusses its contributions, and provides directions for future research.

6.1 Summary

This thesis devised a systematic approach for generating synthetic workloads to stress test session-based systems in a representative, flexible, and automated manner. Based on the approach a tool called SWAT was developed. The tool was used to conduct a sensitivity analysis study that illustrates how the characteristics of several workload attributes influence the performance and effective capacity of session-based systems.

To gain insights regarding the workload model to be used for the approach, the thesis characterized the workload of large real-world session-based system. Several workload attributes that could influence performance were considered and their observed characteristics were presented. In particular, the study found evidence of significant variability in the characteristics of session length, session think times, and the resource demands for the different types of requests in the system. The thesis discussed how such variability can have an adverse impact on a system's performance and consequently recommended the inclusion of these attributes in the workload model.

The thesis employs a workload modeling technique that supports two different workload generation modes depending on the type of arrival characterization desired. The session mode permits control over the instants at which sessions arrive at a system under study whereas the request mode allows control over the arrival instants of requests. The

session mode is useful for studying how changes in user behaviour impact a system's performance. The request mode cannot offer such insights but is more suited for studying the performance-impact of characteristics of the aggregate request arrival process at the system. Depending on the mode employed, the workload model also offers control over the session length distribution, the think time distribution, the workload mix, and the name-value lists' distributions.

A trace generation approach that addresses the complexities of handling inter-request dependencies, and simultaneously incorporating the desired characteristics for session length, workload mix, think time, and name-value lists into the synthetic workload was developed. The approach relies on an input trace of valid request type sequences for the system under study, called sessionlets, to enforce inter-request dependencies. It employs a two-step process to create a synthetic trace of sessions.

During the first step, an iterative algorithm based on linear programming is used to identify a compact subset of sessionlets from the input trace and compute the playback counts of the sessionlets in the subset for simultaneously achieving the desired session length distribution and workload mix. An intermediate trace of sessionlets is formed by repeating the selected sessionlets according to their playback counts.

The second step transforms the intermediate trace into a trace of sessions for the stress test by appending name-value list strings to request types to form URLs, and inserting think time values between successive URLs. Name-value lists that can be determined a priori are chosen according to distributions specified in the workload model. A different approach is used for name-value lists characterized by values that become known only

during the course of a stress test. In such cases, a special placeholder tag is used for the concerned value fields. The tag instructs the request generation engine to determine the values for such fields based on the system's responses when the stress test is in progress. URLs formed in this manner are termed as dynamic URLs. Finally, the think time values are chosen according to the desired think time distribution. The proposed approach was embodied into a tool called SWAT.

SWAT uses a modified version of the open source *httperf* tool as its request generation engine. The request generation engine submits the synthetic trace of sessions to the system under study according to a desired session inter-arrival time distribution (for session mode studies) or a desired request inter-arrival time sequence (for request mode studies). The major extension carried out to *httperf* to support the proposed approach was the addition of a workload generation module for the request mode. The module performs the task of submitting requests at specified time instants to achieve the desired request arrival process while simultaneously ensuring the valid behaviour within a session of submitting a request only after the response to the previous request has been completely received. *httperf* was also modified to support dynamic URLs, arbitrarily desired inter-arrival time distributions, and more detailed tracing and performance logging capabilities.

The thesis also discussed the advantages of the proposed approach which are summarized briefly.

- Unlike first-order Markov chain based techniques described in the literature, the approach proposed ensures correct inter-request dependencies, and allows flexible control

over the session length distribution and workload mix.

- The characterizations for all the attributes in the workload model can be estimated and realized from Web server access logs.
- In contrast to existing aggregate workload generation schemes, the approach proposed achieves a desired request arrival process while simultaneously ensuring that a request in a session is submitted only after the previous response has been received.
- The trace generation engine could be used in conjunction with commercial stress testing tools to automate the development of user emulation scripts and allow finer control over workload characteristics.
- The use of *httperf* ensures accuracy and scalability of request generation, and reproducibility of results.

A stress testing case study was conducted to understand the impact of session length distribution, think time distribution, and workload mix on the performance of session-based systems. The proposed approach was used to create controlled workloads designed to observe how a system's response time distribution is affected when the variability associated with each of the three characteristics changes.

The results showed that high variability in the session length distribution or the think time distribution could degrade a system's responsiveness significantly. For a given mean, a distribution with higher variability will result in larger numbers of very long duration and very short duration sessions when compared to a distribution with lower variability. Consequently, under the same session arrival rate, a workload having the higher

variability distribution can cause increased probabilities for observing very large and very small numbers of concurrent sessions at the system. This phenomenon can in turn cause requests to arrive at the system in a bursty fashion. The increased burstiness can lead to periods of heightened contention for system resources during which requests incur very long response times.

The results also showed that workload mixes that place highly variable demands on a system could have an adverse impact on performance. For the same session arrival rate and mean demand, a higher variability mix was found to cause increased contention among sessions and poorer system responsiveness than a lower variability mix. The results also showed that the degree to which system performance is affected due to higher variability in the mix is greater when there is a higher variability in the session length and think time distributions. Similarly, the effect of higher variability in the session length and think time distributions is more pronounced with a high variability mix in comparison to a low variability mix.

The case study also demonstrated the impact of workload conditions on the effective capacity of a system. The results showed that high variability in session length and think time distributions coupled with mixes characterized by highly-variable demands has the effect of forcing a system to operate at reduced utilization in order to satisfy a desired level of responsiveness. Since system throughput is directly proportional to the utilization of the bottleneck resource, a drop in utilization implies a reduction in system throughput and hence a reduction in effective system capacity.

The thesis also discussed the implications of the results with regard to the conduct of

stress tests. Careful consideration must be given to the characterizations used for workload attributes since they influence the response time and effective capacity estimates obtained from stress tests. For example, using characterizations that result in lower variability for testing a system when empirical evidence points otherwise could lead to overly optimistic performance estimates. In the absence of empirical workload data, a sensitivity analysis along the lines of that presented in this thesis may be required to increase confidence in the results of the stress testing exercise.

The approach proposed in this thesis was validated in a number of ways. Firstly, Section 4.3 and Section 5.4 showed that the approach can create traces that match specified session length distributions and workload mixes. Secondly, Section 5.3 demonstrated that a workload generated by SWAT caused nearly identical performance-impact as the original workload it modeled. Lastly, the experiments presented in Section 5.4 revealed that system performance is sensitive to the characteristics of attributes included in the workload model.

6.2 Contributions

This section summarizes the important contributions of this research. These were described briefly in Section 1.3.

- 1) The research identified why existing methodologies are not adequate for stress testing session-based systems (Chapter 2, Section 2.2.3).
- 2) Based on an analysis of workload observed at a large e-commerce system, the research identified several workload attributes that are of importance for evaluating the performance of session-based systems (Chapter 3).

3) The thesis developed a new method for generating synthetic workloads to stress test session-based systems. The method addresses the limitations found in existing methodologies (Chapter 4). It addresses the requirements of representativeness, flexibility, and automation described in Chapter 1 as follows.

Representativeness

- The approach satisfies inter-request dependencies which ensures that the system's application functions are exercised correctly.
- The approach provides mechanisms for incorporating bursty request arrivals, a behaviour which has been shown to be typical of real Web workloads, within synthetic workloads. For example, burstiness can be introduced by using a highly variable session length distribution in the session mode. Alternately, a bursty inter-arrival time sequence can be specified under the request mode.
- Since the characterizations for all of the attributes in the workload model can be estimated from Web server access logs, it is possible to create synthetic workloads whose characteristics match those of real workloads. This feature was demonstrated in Section 4.3 where synthetic workloads that match the session length distribution and workload mix observed in an input trace were created.

Flexibility

- The trace generation engine allows the flexibility of realizing arbitrarily desired characteristics for attributes in the workload model. As demonstrated in Chapter 5, this allows various workloads that differ from each other with respect to one or more chosen characteristics to be created for the purpose of a sensitivity analysis.

- The approach supports the flexibility of conducting request mode studies for session-based systems.

Automation

- The tasks of satisfying inter-request dependencies, simultaneously matching the characteristics of many attributes, and handling dynamic URLs are performed in an automatic manner.
 - Synthetic workloads can be automatically created from Web server access logs for a system under study since the trace of sessionlets required by the trace generation engine can be extracted without significant user intervention from such logs.
- 4) The research resulted in the SWAT tool which implements the proposed methods.
- 5) The thesis illustrated the impact of session length distribution, think time distribution, and workload mix on the response time distribution and effective capacity of session-based systems through sensitivity analysis experiments (Chapter 5). To the best of this author's knowledge, this is the first such controlled study on session-based systems. It is expected that the results of the study would provide valuable insights for improving system performance as well as conducting stress testing exercises.

6.3 Future work

The research presented in this thesis can be extended in a number of ways, described below.

Capturing variations in workload characteristics – Previous studies on session-based workloads have provided evidence of pronounced time-of-the-day and day-of-the-week

variations in workload characteristics [2]. Future work will exploit the request mode to realize time varying workload characteristics such as throughput and mix for emulating such conditions.

Modeling correlations among attributes - With the proposed approach, there is no correlation among attributes in the workload model. For example, the think times chosen for a session are independent of the session length. However, a study by Arlitt *et al.* on a large e-commerce system has suggested that certain attributes may be correlated with other attributes [5]. For example, the study found through a cluster analysis of sessions in the system's workload that longer sessions, such as those from robots, typically had shorter think times and issued requests related to browsing products. Future work could focus on introducing such correlations into synthetic workloads and investigating whether they have any impact on system performance.

Facilitating study of Wide-Area-Network (WAN) effects – Several studies [47, 54] have shown that WAN characteristics such as packet losses can impact system response time. Integrating mechanisms to emulate such effects with the proposed approach could be a possible direction for future research.

Lastly, refining the SWAT tool and releasing it to the open source community is part of an ongoing effort.

References

- [1] A. Abhari, S. P. Dandamudi, and S. Majumdar, "Using Web Object to Improve the Performance of Proxy Caching," *WWW Workshop on Web Engineering*, pp. 82-91, May 2001.
- [2] 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, September 2002.
- [3] V. Almeida, A. Bestavros, M. Crovella, and A. Oliveira, "Characterizing Reference Locality in the WWW," *Proceedings of the IEEE International Conference on Parallel and Distributed Information Systems (IEEE PDIS 1996)*, pp. 92-103, December 1996.
- [4] V. Anantharam, "Scheduling Strategies and Long-range Dependence," *Queuing Systems: Theory and Applications*, Vol. 33, No. 1-3, pp. 73-89, December 1999.
- [5] 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, August 2001.
- [6] M. Arlitt, R. Friedrich, T. Jin, "Performance Evaluation of Web Proxy Cache Replacement Policies," *ACM SIGMETRICS Performance Evaluation Review*, Vol. 39, No.1-4, pp. 149-164, February 2000.
- [7] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web Site," *IEEE Network*, Vol.14, No. 3, pp. 30-37, May/June 2000.
- [8] M. Arlitt, R. Friedrich, and T. Jin, "Workload Characterization of a Web Proxy in a Cable Modem Environment," *ACM SIGMETRICS Performance Evaluation Review*, Vol.

27, No. 2, October 1999.

[9] G. Banga and P. Druschel, "Measuring the Capacity of a Web Server Under Realistic Loads," *World Wide Web Journal*, Vol. 2, No. 1, pp. 69-83, May 1999.

[10] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *ACM International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS 1998)*, pp. 151-160, June 1998.

[11] C. A. Bennett and N. L. Franklin, "Statistical Analysis in Chemistry and the Chemical Industry," John Wiley and Sons, pp. 157-162, New York.

[12] J. Beran, R. Sherman, M.S. Taqqu, and W. Willinger, "Long-range Dependence in Variable-bit-rate Video Traffic," *IEEE Transactions on Communications*, Vol. 43, No. 2-4, pp. 1566-1579, March 1995.

[13] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1999)*, pp. 126-134, March 1999.

[14] M. Busari and C. Williamson, "ProWGen: A Synthetic Workload Generation Tool for Simulation Evaluation of Proxy Caches," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Vol. 38, No. 6, pp. 779-794, April 2002.

[15] M. Calzarossa and G. Serrazi, "Workload Characterization: A Survey," *IEEE*, Vol. 81, No. 8, pp. 1136-1150, August 1993.

- [16] L. Cherkasova and G. Ciardo, "Characterizing Temporal Locality and its Impact on Web Server Performance," *IEEE International Conference on Computer Communication and Networks (IC3N 2000)*, pp. 434-441, October 2000.
- [17] M. Crovella, and A. Bestavros, "Self-similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, pp. 835-846, December 1997.
- [18] M. Crovella and L. Lipsky, "Long-lasting Transient Conditions in Simulations with Heavy-tailed Workloads," *Winter Simulation Conference*, pp. 1005-1012, December 1997.
- [19] P. Denning, "Working Sets Past and Present," *IEEE Transactions on Software Engineering*, Vol. 6, No. 1, pp. 64-84, January 1980.
- [20] A. Downey, "The Structural Cause of File Size Distribution," *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS 2001)*, pp. 361-370, August 2001.
- [21] D. G. Feitelson, "Workload modeling for performance evaluation," *IFIP International Symposium on Computer Performance Modeling, Measurement, and Evaluation (Performance 2002)*, pp. 114-141, September 2002.
- [22] D. G. Feitelson, "The Forgotten Factor: Facts on Performance Evaluation and its Dependence on Workloads," *International Euro-Par Conference (Euro-Par 2002)*, pp. 49-60, August 2002.
- [23] D. Ferrari, "On the Foundation of Artificial Workload Design," *ACM International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS)*

1984), pp. 8-14, May 1984.

[24] W. Gong, Y. Liu, V. Misra, and D. Towsley, "On the Tails of Web File Size Distributions," *39th Allerton Conference on Communication, Control, and Computing*, October 2001.

[25] F. Hernández-Campos, J. S. Marron, F. D. Smith, and G. Samorodnitsky, "Variable Heavy Tailed Durations in Internet Traffic, Part 1: Understanding Heavy Tails," *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS 2002)*, pp. 43-52, October 2002.

[26] HTTP State Management Mechanism - <http://www.ietf.org/rfc/rfc2109.txt>

[27] Hypertext Transfer Protocol - HTTP/1.1, <http://www.w3.org/Protocols/rfc2616/rfc2616.html>

[28] A. Iyengar, J. Challenger, and P. Dantzig, "A Scalable System for Consistently Caching Dynamic Web Data," *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM)*, pp. 294-303, March 1999.

[29] R. Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling," John Wiley and Sons, New York, 1991.

[30] S. Jin and A. Bestavros, "Sources and Characteristics of Temporal Locality," *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS 2000)*, pp. 28-37, August 2000.

[31] K. Kant and M. Venkatachalam, "Transactional Characterization of Front-end E-Commerce Traffic," *IEEE Global Telecommunications Conference (GLOBECOM 2002)*,

pp. 2523-2527, November 2002.

[32] K. Kant and M. Venkatachalam, "Modeling Traffic Nonstationarity in E-commerce Servers," *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002)*, pp 949-956, July 2002.

[33] K. Kant, V. Tewary, and R. Iyer, "An Internet Traffic Generator for Server Architecture Evaluation," *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2001)*, pp. 49-56, November 2001.

[34] M. Koster, "A Standard for Robot Exclusion," June 1994.

[35] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the Use and Performance of Content Distribution Networks," *ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM SIGCOMM)*, pp. 169-182, November 2001.

[36] D. Krishnamurthy and J. Rolia, "Site Walker - A Tool Supporting Performance Characterization and Capacity Planning for Electronic Commerce Systems," *IFIP International Working Conference on Electronic Commerce*, pp. 5-15, June 1998.

[37] P. Lax, "Linear Algebra," John Wiley and Sons, New York, 1997.

[38] Load Runner - <http://www.mercuryinteractive.com/products/loadrunner/>

[39] 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, February 1994.

[40] A. Mahanti, D. Eager, and C. Williamson, "Temporal Locality and its Impact on Web Proxy Cache Performance," *ACM SIGMETRICS Performance Evaluation Review*,

Vol. 42, No. 3, pp. 187-203, September 2000.

[41] D. Menasce, "Load Testing of Web Sites," *IEEE Internet Computing*, Vol. 6, No. 4, pp. 70-74, July/August 2002.

[42] D. Menasce, V. Almeida, R. Reidi, R. Fonesca, W. Meira Jr., and F. Reibeiro, "Analyzing Robot Behaviour in E-Business Sites," *ACM International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS 2001)*, pp. 338-339, June 2001.

[43] D. Menasce, V. Almeida, R. Reidi, F. Pelegrinelli, R. Fonesca, and W. Meira Jr., "In Search of Invariants in E-Business Workloads," *ACM International Conference on Electronic Commerce*, pp. 56-65, October 2000.

[44] D. Menasce, V. Almeida, R. Fonesca, and M. Mendes, "A Methodology for Workload Characterization of E-Commerce Sites," *ACM Conference on Electronic Commerce*, pp. 119-128, November 1999.

[45] J. Mogul, F. Douglass, A. Feldmann, and B. Krishnamurthy, "Potential Benefits of Delta Encoding and Data Compression for HTTP," *ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM SIGCOMM 1997)*, pp. 181-194, September 1997.

[46] D. Mosberger and T. Jin, "*httpperf* – A Tool for Measuring Web Server Performance," *ACM SIGMETRICS Performance Evaluation Review*, Vol. 26, No. 3, pp. 31-37, December 1998.

[47] E. Nahum, M. Rosu, S. Seshan, and J. Almeida, "The Effects of Wide-area Conditions on WWW Server Performance," *ACM International Conference on*

Measurement and Modeling of Computer Systems (ACM SIGMETRICS 2001), pp. 257-267, June 2001.

[48] V. Padmanabhan and L. Qiu, "The Content and Access Dynamics of a Busy Web Site: Findings and Implications," *ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM SIGCOMM 2000)*, pp. 111-123, August 2000.

[49] K. Park, G. T. Kim, and M. Crovella, "On the Relationship Between File Sizes, Transport Protocols, and Self-Similar Network Traffic," *International Conference on Network Protocols*, pp. 171-180, October 1996.

[50] 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, 1995.

[51] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "QoS-Driven Server Migration for Internet Data Centers," *International Workshop on Quality of Service (IWQoS 2002)*, pp. 3-12, May 2002.

[52] B.K. Ryu and A. Elwalid, "The Importance of Long-Range Dependence of VBR Video Traffic in ATM Traffic Engineering: Myths and Realities," *ACM Computer Communication Review*, Vol. 26, pp. 3-14, August 1996.

[53] Silk Performer -
http://www.seguate.com/html/s_solutions/s_performer/s_performer.htm

[54] R. Simmonds, C. Williamson, R. Bradford, M. Arlitt, and B. Unger, "Web Server Benchmarking Using Parallel WAN Emulation," *ACM SIGMETRICS Performance Evaluation Review*, Vol. 30, No. 1, June 2002.

- [55] Specweb99, <http://www.spec.org/osg/web99>
- [56] J. Sprin, "Distance String Models for Program Behaviour," *IEEE Computer*, Vol. 9, No. 11, pp. 14-20, November 1976.
- [57] SSL 3.0 Specification - <http://wp.netscape.com/eng/ssl3/>
- [58] TPC-W Web Commerce Benchmark - <http://www.tpc.org/tpcw>
- [59] K. Trivedi, "Probability and Statistics With Reliability, Queuing, and Computer Science Applications," John Wiley and Sons, New York, 2002.
- [60] B. Tsybakov and N. D. Georganas, "On self-similar traffic in ATM queue: Definitions, overflow probability bound, and cell delay distribution," *IEEE/ACM Transactions on Networking*, Vol. 5, No. 3, pp. 397-409, June 1997.
- [61] U. Vallamsetty, K. Kant, and P. Mohapatra, "Characterization of E-Commerce Traffic," *Electronic Commerce Research Journal*, Vol. 3, No. 1-2, pp. 167-192, January/April 2003.
- [62] Vantage Point Performance Agent
<http://www.openview.hp.com/products/vppperformance/>
- [63] M. Wang, T. Madhyastha, N. Chan, S. Papadimitriou, and C. Falustos, "Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic," *International Conference on Data Engineering*, pp. 507-516, February 2002.
- [64] C.E. Willis and M. Mikhailov, "Towards a Better Understanding of Web resources and Server Responses for Improved Caching," *International World Wide Web Conference (WWW 1999)*, pp. 153-167, May 1999.

[65] G. K. Zipf, “Human Behaviour and the Principle of Least Effort,” Addison-Wesley, Cambridge, MA, USA, 1949.

Appendix A

Formulation of the linear programming problem

Recalling from Section 4.3, for each session length bin $\mathbf{L}_i$, the three constraints for the LP problem solved by the trace generation algorithm are as follows.

$$\mathbf{A}_i \mathbf{X}_i = \mathbf{B}_i \quad (\text{A.1})$$

$$\frac{\sum_{m=1}^{M_i} X_i(m) \sum_{k=1}^{K} A_i(k, m)}{\sum_{m=1}^{M_i} X_i(m)} = L_i \quad (\text{A.2})$$

$$X_i(m) \geq 0 \quad \forall m \quad (\text{A.3})$$

From Section 4.3, K is the number of request types in the system and $\mathbf{A}_i$ is a $K \times M_i$ matrix whose columns represent request type counts vectors of M_i sessionlets chosen from the input trace of sessionlets. $\mathbf{X}_i = [X_i(1) \ X_i(2) \ \dots \ X_i(M_i)]^T$ is an $M_i \times 1$ vector whose elements represent playback counts for the sessionlets in $\mathbf{A}_i$. $\mathbf{B}_i$ is a $K \times 1$ vector whose elements represent the desired overall counts for the K request types in the system and L_i is the desired average session length for session length bin $\mathbf{L}_i$.

The LP problem requires an objective function that is a linear combination of the elements of $\mathbf{X}_i$. The objective of solving the LP problem is to compute an $\mathbf{X}_i$ that would minimize the difference between the desired request type counts $\mathbf{B}_i$ and the achieved request type counts $\mathbf{A}_i \mathbf{X}_i$. The following objective function is maximized for this purpose.

$$F(\mathbf{X}_i) = \left[\sum_{k=1}^{K} A_i(k, 1) \sum_{k=1}^{K} A_i(k, 2) \dots \sum_{k=1}^{K} A_i(k, M_i) \right] \mathbf{X}_i \quad (\text{A.4})$$

The function $F(\mathbf{X}_i)$ gives the sum of the request type counts achieved with a given $\mathbf{X}_i$. The maximum value of $F(\mathbf{X}_i)$ will result in the minimum possible cumulative difference between the desired and achieved request type counts, given the constraints of Eq. (A.1) to Eq. (A.3). In other words, maximizing $F(\mathbf{X}_i)$ yields the minimum possible value for

$\sum_{k=1}^{k=K} E(k)$, where $\mathbf{E} = \mathbf{B}_i - \mathbf{A}_i \mathbf{X}_i$. Denoting the $1 \times M_i$ vector in Eq. (A.4) containing the

summations as $\mathbf{S}_i$, the objective function can be written as follows.

$$F(\mathbf{X}_i) = \mathbf{S}_i \mathbf{X}_i \quad (\text{A.5})$$

Rearranging terms, Eq. (A.2) can be written as follows.

$$\left[\left(\sum_{k=1}^{k=K} A_i(k,1) \right) - L_i \left(\sum_{k=1}^{k=K} A_i(k,2) \right) - L_i \dots \left(\sum_{k=1}^{k=K} A_i(k, M_i) \right) - L_i \right] \mathbf{X}_i = 0 \quad (\text{A.6})$$

Denoting the $1 \times M_i$ vector in Eq. (A.6) containing the terms with the summations as $\mathbf{G}_i$,

Eq. (A.6) can be written as follows. $\mathbf{G}_i \mathbf{X}_i = 0 \quad (\text{A.7})$

Finally, Eq. (A.3) can be written in matrix form as follows.

$$\mathbf{X}_i \geq 0 \quad (\text{A.8})$$

From Eq. (A.1), Eq. (A.5), Eq. (A.7), and Eq. (A.8), the LP problem is formulated in matrix form as follows.

<p>Maximize $F(\mathbf{X}_i) = \mathbf{S}_i \mathbf{X}_i$</p> <p>Subject to the following constraints.</p> <p>$\mathbf{A}_i \mathbf{X}_i \leq \mathbf{B}_i$</p> <p>$\mathbf{G}_i \mathbf{X}_i = 0$</p> <p>$\mathbf{X}_i \geq 0$</p>

The trace generation algorithm is implemented with MATLAB and uses the LINPROG function to solve the above LP problem.

Appendix B

Time-complexity of the trace generation algorithm

Table B-1 provides elapsed times measured while executing the trace generation algorithm of Section 4.3 on a PIII 500MHZ Windows NT (version 4.00.1381) machine with 128MB of RAM. Two experiments were performed with input traces of 100,000 and 5,000 sessionlets, respectively. For both the experiments, only one session length bin which encompasses all the sessionlets is considered. The algorithm is applied to match the mean session length and the workload mix observed in the input traces. The number of request types is 21 for both cases.

Table B-1: Elapsed time measures for the trace generation algorithm

Number of sessionlets in trace	Number of iterations	Basis computation time (minutes)	Mean time per iteration (seconds)	Total elapsed time (minutes)
100000	7	18.41	12.91	19.92
5000	8	0.96	1.08	1.10

From Table B-1, the number of iterations needed by the algorithm for the 100,000 sessionlets case and the 5,000 sessionlets case is found to be 7 and 8, respectively. The time taken to compute the basis set dominates the total elapsed time. The total elapsed time increases almost linearly with the number of sessionlets in the trace; a twenty fold increase in the number of sessionlets causes the elapsed time to increase by a factor of approximately 20. This result suggests that trace generation may take a long time with traces containing several hundreds of thousands of sessionlets per session length bin. However, small statistical samples of sessionlets are likely to be adequate for characterizing such systems with very large session throughputs, providing a means to

limit solution time. Another means to limit solution time is to apply the algorithm on a reduced trace that only contains the unique sessionlets from the original trace.

Appendix C

Sample synthetic workload trace file

Session 1

Format: *URL* think=*think_time_value*

Script.sc is the request type. Strings that follow “?” are name-value lists

Visit home page of TPC-W store

/cgi-bin/tpc-w/Home.sc think=30.0

View a product page

/cgi-bin/tpc-w/ProductDisplay.sc?category_id=1&product_id=212&subject=Arts think=24.0

Add product to shopping cart

/cgi-bin/tpc-w/AddToCart.sc?product_rn=1&shipmode_rn=1&quantity=1 think=27.0

Request search page

/cgi-bin/tpc-w/SearchRequest.sc think=204.0

Do a keyword search

/cgi-bin/tpc-w/SearchResponse.sc?search=author&title=b think=13.0

Session 2

/cgi-bin/tpc-w/Home.sc think=60.0

/cgi-bin/tpc-w/ProductDisplay.sc?category_id=1&product_id=109&subject=Religion think=23.0

/cgi-bin/tpc-w/ProductDisplay.sc?category_id=1&product_id=994 think=90.0

/cgi-bin/tpc-w/SearchRequest.sc think=14.0

/cgi-bin/tpc-w/SearchResponse.sc?search=title&title=m think=13.0

/cgi-bin/tpc-w/AddToCart.sc?product_rn=994&shipmode_rn=1&quantity=1 think=290.0

Login as a user

/cgi-bin/tpc-w/Logon.sc?name=TEST&password=test think=37.0

Order the product in the shopping cart

/cgi-bin/tpc-w/BuyRequest.sc?status=P think=37.0

A “\$PARSE” in a value field indicates a dynamic URL

Parse the value of “order_rn” from response of previous request

Replace \$PARSE with the order_rn value parsed

Submit Buy.sc with credit card details and confirm the order

/cgi-bin/tpc-w/Buy.sc?order_rn=\$PARSE&cc=MAST&ccnm=koko&ccnum=123&ccx=07 think=14.0