2005

Measurements based performance analysis of Web services

Venu Datla
West Virginia University

Measurements based Performance Analysis of Web Services

Venu Datla

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Committee Members
Dr. Katerina Goseva – Popstojanova, Ph.D., (Commitie Chair)
Dr. V. Jagannathan, Ph.D.
Dr. James D. Mooney, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2005

ABSTRACT

Measurements based Performance Analysis of Web Services

Venu Datla

Web services are increasingly used to enable interoperability and flexible integration of software systems. In this thesis we focus on measurement-based performance analysis of an e-commerce application which uses Web services components to execute business operations. In our experiments we use a session-oriented workload generated by a tool developed accordingly to TPC-W specification. The empirical results are obtained for two different user profiles, Browsing and Ordering, under different workload intensities. In addition to variation in workloads we also study the applications performance when Web services are implemented using .NET and J2EE. Unlike the previous work which was focused on the overall server response time and throughput, we present Web interaction, software architecture, and hardware resource level analysis of the system performance. In particular, we propose a method for extracting component level response times from the application server logs and study the impact of Web services and other components on the server performance. The results show that the response times of Web services components increase significantly under higher workload intensities when compared to other components. From the hardware resource measurements it is obvious that the higher response times of Web services components are due to parsing XML messages and contention for database resources. The results of our study identify software components and hardware resources which are potential bottlenecks in the system and thus provide valuable information for capacity planning of web and e-commerce applications.

ACKNOWLEDGEMENT

I would like to express my gratitude to my advisor Dr. Katerina Goseva – Popstojanova for her support and guidance through my thesis. I am also grateful to her for introducing me to new and interesting technologies in software development.

I am also grateful to my other committee members, Dr. Jugannathan and Dr. Jim Mooney for their support. I would like to thank NASA IV & V Facility, Fairmont, West Virginia which provided financial support for my graduate studies through NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP). Finally I would like to thank my family and friends for their constant help and support.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1: Introduction

Modern Web applications are large-scale, distributed and depend on various inter-enterprise and intra-enterprise services for execution. Since these services are developed on different platforms, programming languages and technologies their integration with the application becomes a complex task. The Web services architecture facilitates interoperability and flexible integration of systems developed on heterogeneous environments. The interface of a Web service is described in a machine processable format. Other software systems can communicate with the service using XML messages that are conveyed via Internet protocols such as HTTP, SMTP, and FTP. The interface details of a Web service can be published in a repository to allow other users and applications to discover the service. Individual services can be assembled to create composite value added Web services and applications. The technologies that enable Web services description, communication and discovery are WSDL, SOAP and UDDI. For more detailed descriptions the reader is referred to [8], [21].

With service oriented architecture, interoperability and ease of integration, Web services have become a popular choice for developing Web applications. Enterprise application development technologies like .NET and J2EE have incorporated support for Web services in their specifications. Companies like Amazon, Google, and Microsoft have released Web service interfaces for some of their Internet services.

The Web services technology has a lot of potential for application-to-application communication since it promotes interoperability and extensibility among these applications. Of course, Quality of Service (QoS) provided by Web services will play a major role in their success and adoption rate. Although some emerging standards address methods for achieving message delivery guarantees (WS-Reliability [28]) and integrity and confidentiality (WS-Security [37]), the current state of practice in description and discovery of Web services does not include specification of QoS attributes such as performance, reliability, availability, and security. In other words, Web services technology has not yet addressed questions such as will the Web service meet the

performance requirement of 2 ms response time or will the Web service be available when needed? Until these questions are addressed, it is unrealistic to expect that businesses will discover Web services in a UDDI registry based on functional requirements and invoke that service without having any assurance that the QoS requirements will be met.

In this work we present a measurement-based study of performance of an e-commerce application that uses Web services to execute business operations. We focus on software architectural view of the e-commerce prototype and analyze the performance aspects of Web services components under controlled workload conditions. We also measure the impact of the application execution on the hardware resources of the system. As there are many Web services development platforms available, the natural question arises on which software performs better than the other. In our research we compared the performance of Web services implemented and deployed in different application servers. Particularly, we compare Web services performance at hardware and software architecture level in J2EE and .NET platforms.

The thesis is organized as follows. In chapter 2 we describe in detail technologies and standards related to Web services. Related work on performance evaluation of Web services and our contributions are discussed in chapter 3. The description of the prototype, including the software architecture, implementation, and deployment details, is given in chapter 4. The workload used in our experiments and the measurement methodology are described in chapters 5 and 6, respectively. Chapter 7 presents the experimental results. In Chapter 8 we compared the performance of J2EE and .NET Web services. Finally, the concluding remarks are given in chapter 9.

Chapter 2: Related Technologies

Extensible Markup Language (XML) is the basis for most of the Web service languages. It is a standard for representing and exchanging data. XML documents are written in plain text resulting in portability and flexibility. In XML format data is represented in hierarchical constructs called elements. To define the structure of an XML document the syntax of the document is represented using XML Schema language. An XML parser is used to validate an XML document against the XML schema. Commercial and open source implementations of XML parsers are available in C, C++, Java and several other programming languages. XML parsers play an important role in Web services and their performance. Parsing the XML content of messages send to/from a Web service affects the service and response times of a service. Another factor affecting the performance of a service is the size of message.

Web Service Description Language (WSDL) is an XML grammar for specifying the properties of a Web service such as what it does, where it is located, and how it is invoked. It describes the messages exchanged by the service, operations supported by the service, protocol bindings and endpoints of the service, etc. The language uses XML Schemas to define platform independent data types used in the messages [6]. XML namespaces are used to unambiguously describe a data type or message. WSDL also defines the type of SOAP communication used for the service (RPC style or Document Style) [24]. Currently, WSDL does not specify the quality parameters of a service. New standards and frameworks are being developed for specification of QoS in the service definition. Generally WSDL descriptions are published in a service registry for automatic discovery.

Simple Object Access Protocol (SOAP) is a standard for sending messages and making remote procedural calls over the Internet. It is a light weight XML based protocol and is independent of the programming language, object model, operating system, and platform. It uses HTTP as the transport protocol and XML for data encoding. However, other

transport protocols, such as FTP, SMPT, or even raw TCP/IP sockets, may also be used. SOAP defines two types of messages, request and response, to allow service requesters to request a remote procedure and service providers to respond to such requests.

A SOAP message is contained in a SOAP Envelope. A SOAP envelope contains a header and a body. The header element is optional. It is used to convey additional information regarding data such as transactions, billing, formats etc. The SOAP Body is the actual place which contains the XML data to be communicated. A SOAP message must contain a body element .The structure of a SOAP message is shown in Figure 1.

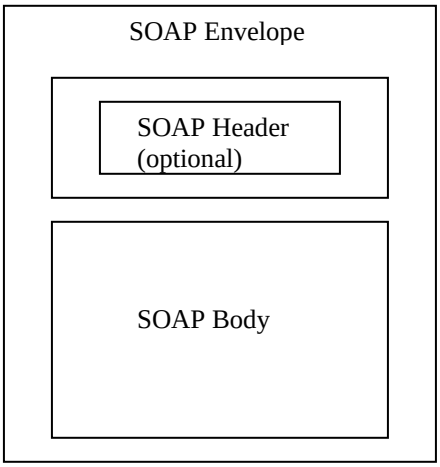

Figure 1: Structure of a SOAP message

Errors can also be represented in SOAP messages using the SOAP Fault element. This element can be used to convey exceptions that occur when servicing a request. SOAP messages use different encoding schemes to structure data. An encoding style specifies a set of rules for serializing data types in the message. Two styles of soap encoding are Document and RPC. When using RPC the structure of SOAP message must conform to the method definition. When using Document style of encoding the serialization rules are specified in the form of an XML schema.

Universal Discovery, Description, and Integration (UDDI) provides a standard way for businesses to publish and discover Web services. Unlike WSDL and SOAP which are standards from W3C, the standardization process of UDDI specification is taken up by

OASIS [27]. The UDDI specification consists of an XML schema for UDDI data structures and description of UDDI APIs specifications. A UDDI stores the service definitions of a business service in XML format. It stores the following information of a service in the business registry [29]:

- *Business Entity:* This contains the information of a business organization that publishes the service and is similar to white pages.
- *Business Service:* This information contains categories of services representing Yellow pages.
- *Binding Template:* This is similar to green pages. Information regarding the technical details of a service is represented using a template.
- *tModels:* The tModel is used to specify the description of service interfaces.

Chapter 3: Related Work and Our Contributions

Software performance can be analyzed using different approaches such as measurements, analytical modeling and simulation. In measurements approach performance metrics are collected by exercising the actual system with a real or synthetic workload. This kind of analysis is not feasible in early stages of system development. In such cases an alternative method for evaluating performance is to build analytical performance models. There are several analytical approaches to building performance models of the system like queuing networks, layered queuing networks, Petri nets etc. The analytical models are solved to obtain performance characteristics of the system. Analytical modeling is a cost-effective method for performance evaluation. One disadvantage of analytical models is that for large scale systems the model might be complex to build and solve. A third alternative approach for software performance analysis is to generate a simulation model which mimics the behavior of the system. The accuracy of the simulation model depends on how closely it represents the original system.

3.1. Related work on Web services performance

Performance is an important quality aspect of Web services because of their distributed nature. Surprisingly, few researchers have focused on performance evaluation of Web services in the past. The throughput and overall system response time of two variants of J2EE Pet store application [34], one implemented using Java Messaging Service (JMS) and the other using Web services, were studied in [11]. The application server used in the experiments is Web Logic Server 7.0 and the database server is Oracle 9i. In this work the workload was generated using the Siege tool [26]. The performance data was collected by logging the timestamps that indicate invocation times, request completion times. It was shown that the JMS version has better performance than the Web services version of the application. The Web services implementation also has higher garbage collection activity.

A similar study was presented in [20]. The authors empirically compared two versions of an electronic book inventory system implemented using Active Server Pages (ASP) and Web services. The Web Server used is Internet Information Server version 5.0. The workload generator used in this study was S-client [2]. Performance of the system is analyzed using load generated from two variants of S-client. In first case the load consists of a fixed number of clients. In the second case the Web server was overloaded with requests. For each version of workload the throughput and response times for each implementation were compared. The results showed that the ASP implementation has higher throughput and lower response time than the Web services implementation.

Analytical performance modeling techniques have been used to identify performance problems in Web applications in [4] and [13]. Layered Queuing Network (LQN) model was used in [4] to calculate response times of a Web service based clinical decision support system. The LQN model was built based on the software architecture. The model was not validated with actual measurements.

Queuing network model for performance evaluation of an e-commerce application was proposed in [13]. The application chosen for study is the one specified in SPECAppServer2002 [2] benchmark. It models an e-business system that has the following functionalities: manufacturing, supply chain management, ordering and inventory management. Performance of this system was measured under three varying workload conditions: low, moderate and heavy. The estimates of the response times, throughput, and utilization were compared with actual measurements. Although this application was not implemented using Web services, the paper describes performance evaluation of a large scale J2EE application which is related to our work. Another related work on analytical modeling of QoS attributes (i.e., response time, reliability, and cost) of workflows and Web service processes was based on reduction rules [3].

A simulation technique for analyzing performance of composite Web services was proposed in [5]. In this paper the authors considered a scenario of an online book store

and used the simulation tool JSIM to build the simulation model of this scenario. The service time, communication latency, and waiting time for each Web service in the scenario were measured by load testing. The results from the simulation model were found to be close to the results obtained from the actual service execution.

With widespread adoption of Web services enterprise application development frameworks like J2EE and .NET have incorporated support for Web services technologies in their architectures. In [16] the authors discuss how results of performance benchmarking applications like Java pet store [34], which favor J2EE technology, might be flawed. The paper discusses in detail about J2EE and .NET platform's support for implementing Web services.

3.2. Related work on quality of Web based systems

Quality of Web based systems has been studied widely in many research works. In this section we explain research studies which address quality of service issues in Web applications.

Reliability and Availability of a large scale J2EE Web application was analyzed in [7]. The application used is Pet-store [34], a sample J2EE application developed by Sun Microsystems. The workload generator used in this study is a variant of TPC-W[35]. This paper describes a method for determining faulty components of the application. In this method a client request is traced as it passes through the system. Data mining techniques are used to identify failure paths from the component traces.

In [10] the authors study performance of an ecommerce Stockbrokerage application implemented using Enterprise Java Beans technology. Performance of two versions of this application was studied by deploying them on five different application servers: Borland Enterprise Server, Interstage Application Server, SilverStream Application Server, WebLogic Server, WebSphere Application Server, and JBOSS. The results show the application exhibits significantly different performance characteristics in each

deployment environment. Our work is different from this one as we use SOAP based Web services in addition to EJB's.

Performance and scalability of J2EE based websites was measured in [5]. The prototype implemented here is an online auction website similar to ebay. The application was implemented in four different versions. The versions differ in the EJB type used for implementing the business logic layer. The auction website was tested by generating a workload similar to the one specified in TPC-W benchmark [35]. For each version the performance measurements are made by deploying the application on different application servers. The application servers considered are JBOSS, JOnAS. The throughput of the system was measured in each case. The results show that JOnAS server performs better that JBOSS.

3.3. Contributions

In this thesis we focus on measurement-based study of Web services performance. For this purpose we developed a three tier e-commerce prototype of an online travel agency. Our intention is not to test stand alone Web services, but to examine how they perform when integrated into applications. The functionalities of our e-commerce system that require interaction with other, most likely heterogeneous, systems (e.g., planning itineraries, currency conversion, and validation of credit card information) are implemented as Web services.

Since the traffic in e-commerce environments is based on sessions, request-based workload generators used in [6], [20] are not suitable for our application. Therefore, we have developed a session-based workload generation tool based on TPC-W benchmark specification [35]. TPC-W is oriented toward business-to-consumer e-commerce interactions and tests many important elements of most e-commerce applications [15]. It should be emphasized that implementing the TPC-W benchmark is a complex task that involves managing a wide spectrum of software and communication technologies [9]. Our implementation of the workload generator adapts the workload designed for an

online bookstore given in TPC-W to suit the requirements of our application (i.e., online travel agency).

Unlike the previous work [4], [11], and [20] which analyzed the overall throughput and response times of Web service based applications, we measure the performance at architectural level, that is, we study the impact of Web services and other components on the performance of the system. For this purpose we have instrumented the application to record the component execution events in the Application server logs and developed scripts in AWK [25] scripting language to automate the task of extracting response times for each component from the Application server logs. To the best of our knowledge, the method for data extraction from Application server logs has not been used earlier for studying Web services performance. In addition to the architectural level measurements, we study the impact of the application on the hardware resources of the deployment environment.

Web services implementation and deployment is supported by several application development platforms. In this thesis we analyze and compare the performance of Web services implemented using .NET and J2EE. Although performance of applications developed in different platform architectures is compared in other works [10], [5] they did not compare performance of Web services in .NET and J2EE.

In our experiments we use two different workload profiles, Ordering and Browsing, and compare the corresponding components response times, as well as hardware resource usage for different workload intensities. It should be noted that although the overall throughput and system response time were measured under increasing load in [11] and [20], different workload profiles were not considered. The empirical results presented in this thesis contribute toward quantifying the overhead introduced by Web services and help identifying software components and hardware resources which are bottlenecks in the system. In particular, we show that Web services components have significantly higher response time under Ordering profile. This information is valuable for system designers due to the fact that customers in Ordering profile tend to have more ordering

activity and generate revenue. From this perspective, our work is complementary to the work presented in [14] which was focused on priority-based resource management policies aimed at increasing the business-oriented metrics such as revenue per second. The results of our research work are presented in [39].

Chapter 4: Prototype description

In this chapter we describe the software architecture, implementation and deployment details of our prototype e-commerce application - an online travel agency which offers flight booking services to its customers. Specifically, the application provides online customers with facilities to search for flights, choose flights that match their preferences, and purchase tickets securely.

4.1. Software architecture

Our prototype is designed in a three-tier architecture which is suitable for development of e-commerce systems because they are distributed and typically span several systems such as Web servers, application servers, and database servers. Based on the logical functionality, in three-tier architecture, the application is organized into user interface layer, business logic layer, and data layer. The user interface layer of our application consists of a set of Web pages: Home page, Search page, Search Results page, Shopping Cart page, Customer Login page, Check Credit page, and Process Order page. The last three Web pages are secured using HTTPS protocol since they transmit sensitive information such as credit card information and passwords.

The business logic layer contains components that implement the core functions of the travel agency application. The main components in this layer are:

Flights-WS is a Web service that takes flight details like start date, end date, origin, destination and number of passengers from the customer and returns a SOAP message containing a list of matching flights. This Web service is hosted locally. The first version of our prototype integrated the publicly available Web service [22] which has the same functionality. However, this service had poor availability. Furthermore, when it was available the service responded with server error whenever more than five simultaneous

search requests were generated. Due to these reasons we decided to implement the Flights-WS and host it locally.

Credit-WS is a Web service which validates customer's credit card information. This Web service is hosted locally.

Currency-WS is a locally hosted Web service which calculates the exchange rates between two currencies. The WSDL of a similar but publicly hosted Web service is located at [23].

Customer-EJB component stores the customer information such as name and ID for the duration of the customer session.

Login-EJB component performs the login function by validating customer's username and password.

Order-EJB component is responsible for maintaining the persistence of customer orders. Persistence is an important aspect since the order information should be preserved even after the customer logs out of the system.

It should be noted that components that require interoperability in order to interact with other (possibly heterogeneous) systems are implemented as Web services Flights-WS, Credit-WS, Currency-WS.

The data layer of our application consists of a backend relational database management system that stores persistent information in the form of tables. The components of the business logic layer, Flights-WS, Credit-WS, Order-EJB, Customer-EJB, and Login-EJB manipulate the data in the corresponding database tables to process requests from the user interface layer.

4.2. Implementation details

Our online travel agency application is implemented using J2EE [32], a widely used standard which facilitates development of scalable, robust, multi-tiered enterprise systems. The user interface layer is written in Java Server Pages (JSP) which is a J2EE technology for creating dynamic Web content. We use Tomcat v5.0 as a Web server.

The business logic layer components are implemented using Web services and Enterprise Java Beans (EJB). Starting from version 1.4, J2EE has added support for Web services in the form of JAX-RPC API which we use to create the Web service components Credit-WS, Flights-WS, and Currency-WS. The other business logic layer components, Order-EJB, Customer-EJB, and Login-EJB, are implemented as EJB which is a J2EE standard for developing server side components.

Finally, we use Oracle 9i Release 2 as a database server.

4.3. Deployment details

The UML deployment diagram of our prototype application is shown in Figure 2. The Web server and EJB components run on the same machine with a 3 GHz Pentium 4 processor and 1 GB RAM. The application server which hosts the Web services components Flights-WS, Currency-WS, and Credit-WS runs on another system with a 3GHz Pentium 4 processor and 1GB RAM. The database server runs on a different machine with the same configuration and 120 GB disk drive. We use a 1.2 GHz Pentium M processor with 512 MB RAM system to run the workload generator. All these machines run Windows 2000 operating system and are connected through Ethernet LAN with 100 Mbps speed.

We decided to develop all Web services and host them locally due to two main reasons. First, as explained in Section 5.1, during our initial experiments we found that some of the public Web services have low availability and reliability. This does not seem

to be an isolated incident. In [19] it was reported that in 2001 48 % of the production UDDI registry had links that were unusable. A more recent study [12] reported similar findings - during six months period (August 2003 - January 2004) 67 % of the public Web services registered in the UDDI registry were invalid (i.e., their WSDL files were either inaccessible or not registered. This state of the practice prevents integration of public Web services in any application which relies on them to achieve high dependability.

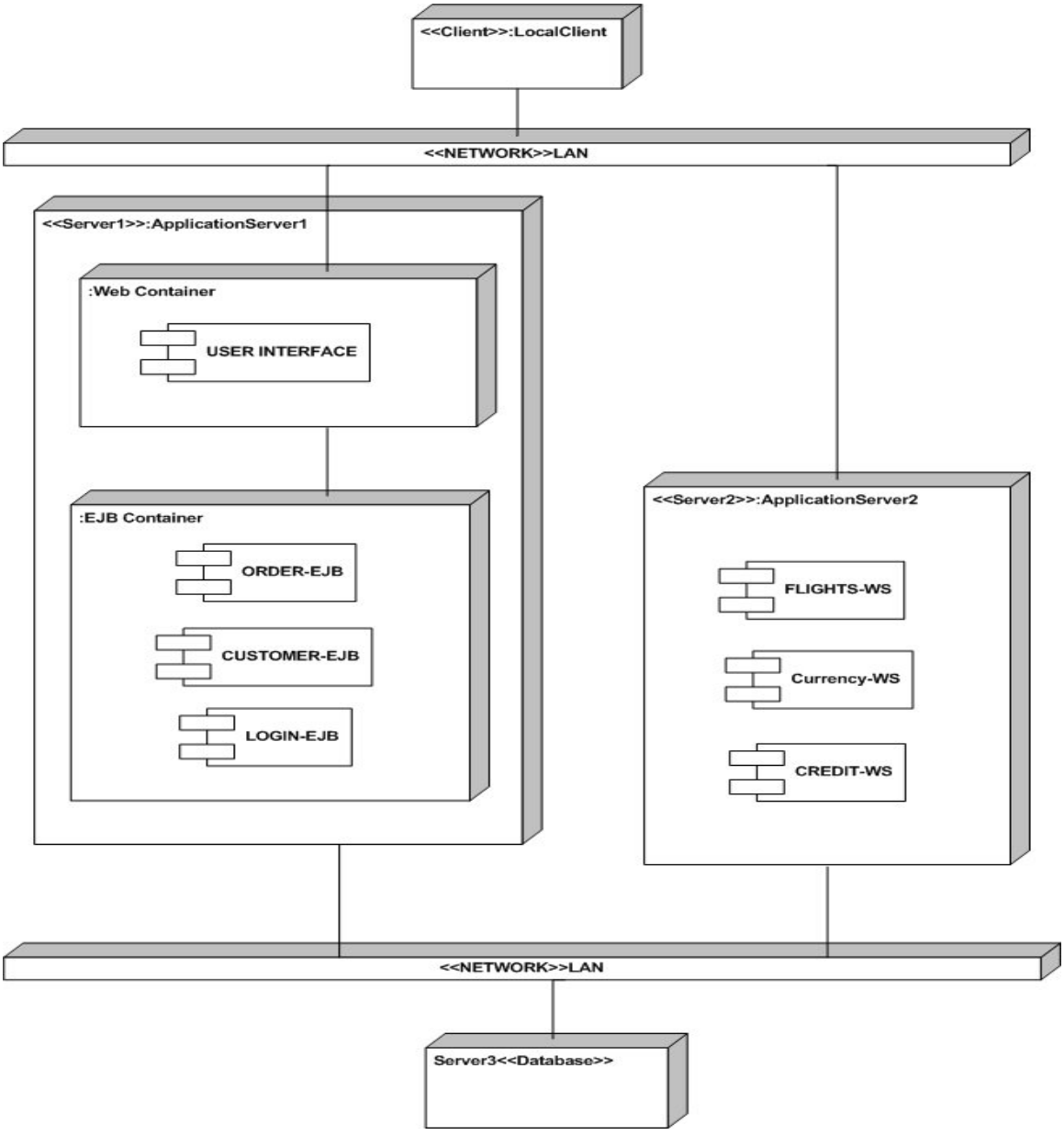

Figure 2: UML deployment diagram of the travel agency application

Second, by hosting all software components locally we avoid accounting for network latency which is beyond our control. This, however, does not limit the scope of our research since our goal is to study the contribution of software components to the response time of e-commerce interactions at the server-side rather than to study end-to-end response time as perceived by the user. Even more, hosting all components locally supports experiments with higher workload which may not be possible with publicly hosted Web services components.

Chapter 5: Workload Description

A key issue in performance evaluation of software systems is the workload characterization which should closely represent the behavior of real users. The design of synthetic workload is affected by several factors like request rate, request size, session length, think time, open or closed loop model and so on. An excellent survey presented in [1] analyzes in details popular Web workload benchmarking tools such as httperf [17], SPECweb99 [30], Surge [18], S-Client [2], TPC-W [35], and Web Stone[38]. Table 1 compares the characteristics Web benchmarks and workload models. Next we explain in detail benchmarking tools and specifications for Web workload.

- *Httperf* [17] is an open loop Web benchmarking tool developed by researchers at HP labs. The characteristic feature of open loop models is that clients make requests independent of the server responses. This model is more appropriate for performance evaluation of Websites as it closely follows real Web traffic patterns. The tool generates either a request based or session based workload. It is capable of generating workloads based on traces from Web server logs. Httperf supports HTTP 1.0 and HTTP 1.1 protocols. Cookies and basic SSL requests are also handled by the tool.

- *SURGE* [18] generates Web workloads using various analytical distributions. It is capable of generating self similar workload and mainly useful for dealing with static requests. The main limitation of this benchmark is that it does not account for dynamic requests. This benchmark follows a closed loop model. In a closed loop model clients generate requests only after receiving response from previous requests. The workload also addresses the burstiness feature exhibited by real Web traffic. The SURGE benchmark supports both HTTP 1.0 and HTTP 1.1 protocols.

- S-Clients [2] workload is designed to measure Web server capacity and performance. The workload is particularly useful for stress testing a Web server system. It does not exercise other tiers of the Web system like backend databases. Hence the workload generated by S-Clients is not realistic. Also the tool does not support HTTP 1.1 protocol.

- WebStone [30] workload includes facilities for generation of static and dynamic services. The benchmark is request based and it is not useful for session-oriented workloads. The only protocol supported by Webstone is HTTP 1.0. It does not handle encryption and authentication. The workload follows a closed loop model.

- SPECweb99 [38] is a benchmark from SPEC organization for evaluating the performance of Web servers. It is a successor of an earlier Web benchmark SPECweb96. It can generate both static and dynamic Web requests and supports secure request generation using SSL. The workload generated by this tool has fixed number of clients per experiment. Hence it represents a closed loop model. The tool supports generation of workload to test commercial Web server features like advertising and user registration.

- TPC-W [35] is a benchmark aimed at evaluating the performance of websites which communicate with backend database for serving requests. The workload generated by TPC-W is closed loop and is session-oriented. The workload intensity in TPC-W depends on the size of database tables. Since TPC-W is only a specification benchmark developers can customize their implementations according to their program environments. TPC-W introduces cost based metrics which can be used for comparing performance of different Web server systems.

	SPECweb99 [30]	*WebStone[38]*	*SURGE[18]*	*Httperf[17]*	*TPC-W[35]*
Organization	Standard Performance Evaluation Corporation	SGI	Bardford & Crovella. Boston University, CS Dept	HP	Transaction Processing Performance Council

	SPECweb99 [30]	*WebStone[38]*	*SURGE[18]*	*Httperf[17]*	*TPC-W[35]*
Type of request supported	Static/Dynamic	Static	Static	Static/Dynamic	Static/Dynamic
Metrics measured	(Primary Metric)Maximum number of simultaneous connections under specific error rate and throughput requirements	(Primary Metric)Maximum server throughput and average response time	Session-oriented metrics	Session and request oriented statistics	(Primary Metric) WIPS- web interactions per second $/WIPS- cost metric
Protocols supported	HTTP/1.0 HTTP/1.1	HTTP/1.0	HTTP/1.0 HTTP/1.1	HTTP/1.0 HTTP/1.1	HTTP/1.0 HTTP/1.1
Secure (SSL) Connections	Yes	Not officially supported .Patched version has support for SSL	No	Yes	Yes
Wokload Generation mechanism	POSIX threads or processes	Preconfigured number of user processes acting as clients	User Equivalent represented by a thread	A process implementing an event-driven approach with non-blocking I/O	As threads knows as 'emulated browsers' making requests to SUT

	SPECweb99 [30]	*WebStone[38]*	*SURGE[18]*	*Httperf[17]*	*TPC-W[35]*
Goal of benchmark	Comparing Web server performance	Measure performance of Web servers	Generating representative workloads based on analytical models of Web use. To show self similarity	A robust tool for measuring Web server performance (ability to generate and sustain server overload)	Evaluating performance of e-commerce web sites.
Session oriented workload	No	No	Yes	Yes	Yes
Scalability	Has certain degree of scalability but cannot sustain when the distributed system is under stress	Not able to sustain high loads	Not scalable. Clients have to be distributed on different client nodes for scalability	Must be run of distinct nodes to achieve scalability	Depends on database table size and scaling factors in TPC-W specification

Table 1: Comparison of Benchmarks and workload models

In this work we analyze the performance of e-commerce applications using synthetically generated workload which allows us to run controlled experiments. For our application the workload should emulate the activity of online customers interacting with the e-commerce Web site through a browser. The customer behavior under these conditions is session oriented. Benchmarking tools such as SPECweb99 and S-Client are request-based and do not capture the concept of customer sessions. We decided to use the TPC-W [35], a benchmark from Transaction Processing Performance Council (TPC), which specifies a session-based workload for simulating customer activities for an online bookstore application. TPC-W is a well designed benchmark oriented toward business-to-

customer e-commerce applications which was studied and evaluated in [9], [15]. Its main features include generation of multiple online browser sessions, dynamic page generation with database access and update, authentication through secure socket layer (SSL) or transport layer security (TSL), and enforcement of ACID properties on database transactions. Another advantage of TCP-W benchmark is the capability of generating different Web interaction mixes which consists of different percentages of browse and ordering operations.

It is important to emphasize that TPC-W benchmark is a specification, not a tool that can readily be used for workload generation. As a part of this research effort, we have developed a workload generation tool accordingly to TPC-W specification. This is a complex task that requires knowledge of wide spectrum of software and communication technologies [9]. It should be noted that our implementation adapts the workload designed originally in TPC-W specification for an online bookstore to suit the requirements of an online travel agency.

Workload characterization in TPC-W is based on the customer's view of the system and it can be described with a Discrete Time Markov Chain (DTMC) which characterizes the customers request patterns. DTMC consists of a set of user states; each request is represented as a transition from one state to another. Accordingly to the Markov property, the transition to the next state is a function of the current state and the transition probability. The probabilities associated with transitions are determined from the workload profiles (i.e., Web interaction mixes). Note that in [15] the DTMC model is called a Customer Behavior Model Graph (CBMG).

The DTMC which defines the user sessions for our application is show in Figure 3. Each customer session starts in the Home state and navigates through the states of the DTMC. For each user session the emulated browser (client) in TPC-W generates a random number from a negative exponential distribution which represents the User Session Minimum Duration (USMD). The user session ends when the USMD has elapsed and the next Web interaction is Home Web interaction. Because there will be on average

a non-zero time between the USMD elapsing and the next selection of Home Web interaction, the actual average duration of user sessions will be somewhat greater than USMD. The user session does not end until the next Home Web interaction in order to maintain the required mix of Web interactions (i.e., workload profile). A new customer session is started as soon as the workload generator terminates the current session. The clients in TPC-W workload follow the closed loop model. In this model the workload consists of a fixed number of clients which generate new request only after the response on the previously submitted request is received from the server.

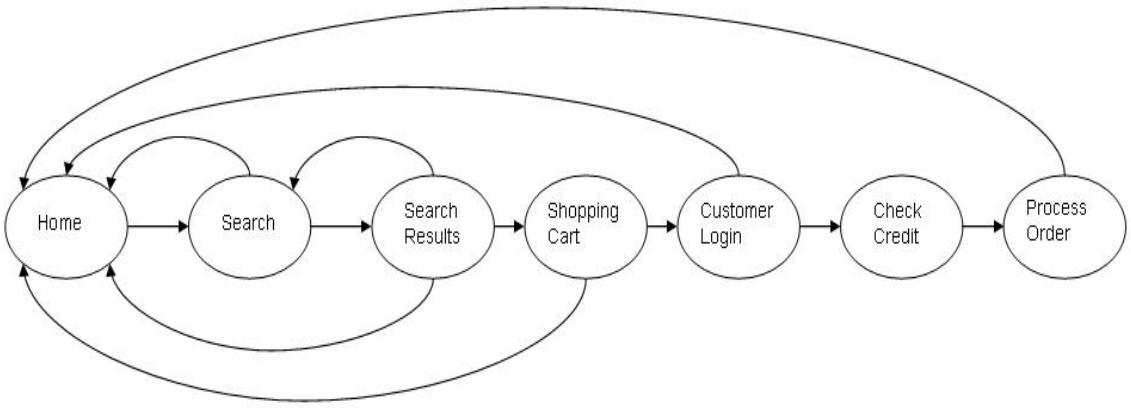

Figure 3: DTMC for the travel agency application

The TPC-W workload is made up of a set of Web interactions which can be classified as either Browse or Order depending on whether they involve browsing and searching on the site or whether they play an explicit role in the ordering process. In our case the browsing category consists of Home, Search, and Search Results interactions, while the ordering category consists of Shopping Cart, Customer Login, Check Credit, and Process Order interactions. In this thesis we run experiments with two different workload profiles. The Browsing profile describes the behavior of customers who spend most of their time browsing and searching and rarely place orders for tickets. In this profile 79% of requests are for interactions in browsing group and only 21% are for

interactions in the ordering group. In the Ordering profile customers tend to have more ordering activity, that is, 50% of requests are for browsing interactions and 50% for ordering interactions. The detailed mixes of Web interactions for these two profiles are shown in Table 2.

	Browsing profile (79-21)	Ordering profile (50-50)
Browse	71%	50%
Home	21.0%	17.0%
Search	30.0%	17.5%
Search results	28.0%	15.5%
Order	21%	50%
Shopping cart	12.0%	14.0%
Customer login	3.2%	13.0%
Check credit	2.9%	11.5%
Process order	2.9%	11.5%

Table 2: Mix of Web interactions for Browsing and Ordering profiles

The workload generated accordingly to TPC-W specification consists of three phases: ramp-up interval, steady-state interval, and ramp-down interval [35]. During the ramp-up interval the system initializes its components and reaches a steady-state. The data must be collected over a measurement interval during which the throughput level is in a steady-state condition that represents the true sustainable performance of the application. In our experiments the duration of the ramp-up, steady-state, and ramp-down intervals are 5, 30, and 1 minute, respectively.

Another important requirement imposed by the TPC-W specification is that the size of the database tables must be scaled accordingly to the number of clients. For both Ordering and Browsing profiles we run experiments with 50, 100, 150, and 200 clients. Therefore, following the TPC-W specification [30], we populate the database with a customer table of size 576,000 rows. TPC-W specification also requires average think time and average user session duration to be reported, which in our case are 7 seconds and 11 minutes, respectively. Finally, TPC-W imposes restrictions on the response times

for each type of Web interaction shown in Figure 3 and requires reporting of the 90th percentile response time during the steady-state measurement interval.

Chapter 6: Measurement methodology

For each Web interaction the TPC-W benchmark measures at the client-side (i.e., Emulated Browser) the Web Interaction Response Time (WIRT) which is defined as the difference between the time measured after the last byte of the last HTTP response that completes the Web interaction is received by the Emulated Browser (EB) from the System Under Test (SUT) and the time measured before the first byte of the first HTTP request of the Web interaction is sent by the EB to the SUT.

Our goal is to measure the response time at software architectural level which will allow us to study how each software component contributes towards server--side response time for each Web interaction. The Web interactions presented in Figure 3 involve executing from one to three different software components (see Section 5.1) as listed below.

- Home interaction: Home page and Customer-EJB
- Search interaction: Search page
- Search Results interaction: Search Results page and Flights-WS
- Shopping Cart interaction: Shopping Cart page
- Customer Login interaction: Customer Login page
- Check Credit interaction: Check Credit page, Login-EJB, and Currency-WS
- Process Order interaction: Process Order page, Credit-WS, and Order-EJB

We extract information about the response times of components participating in each Web interaction from the Application server logs. J2EE Application servers record application events in ASCII log files using the java.util.logging API [33]. An application event may be a request for Web page, execution of an EJB method, a request for a Web service, error, exception and so on. The format of the records in the application server logs is shown in Figure 4. It contains the time stamp of the event, log level that identifies priority of the message, name of the application server, component that logs this message, key value pairs containing thread ID, message ID, and the message.

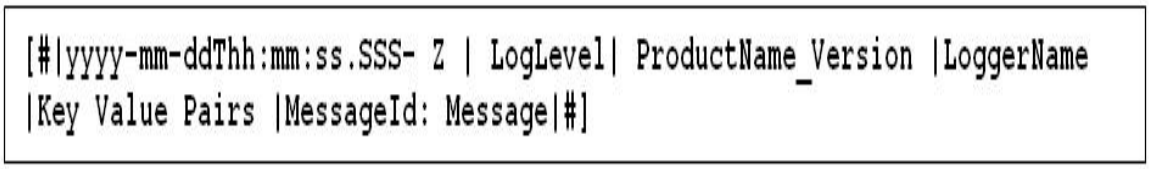

Figure 4: Format of a record from the Application server log

In default server settings only critical events such as errors and exceptions are logged. We modified the application server settings to enable the Web container and EJB container to log time stamps of all relevant events in our application. Then, the application components were instrumented by adding statements which call the java.util.logging API. This API persists components response times in the application server logs. Since during our experiments many events were recorded in the application server logs, their size was in range of hundreds of Mega bytes. Of course, extracting the response times for each execution of each component cannot be done manually. Therefore, we wrote scripts in AWK scripting language [25] which parse the application server logs and automatically extract component level response times.

In addition to software architecture level measurements, we also study the hardware resource usage of Web services based e-commerce application. For hardware resource level measurements we use Windows 2000 performance monitoring tool. In particular, we use the Performance Logs and Alerts utility to create counter logs which record data about hardware usage and activity of system services. Since the components of our e-commerce application are deployed across several machines (see Figure 2), on each machine we record the percentage of non-idle processor time spent in user mode (*%User Time*) and the rate of read and write operations on the disk (*Disk Transfers/sec*).

In this thesis we measure and compare the performance of Web services implemented using .NET and J2EE. For these experiments we use the same methodology and workload as described above. But in this case we run the experiments by deploying .NET implementation of the Web services on server2 in Figure 2.

Chapter 7: Experimental results

First, we analyze the response times of the Search, Shopping Cart, and Customer Login interactions which serve only static html content. Since response times of these interactions are similar, we discuss only the results of the Search interaction. As it can be seen from Figure 5, which shows the 90th percentile response times for Search interaction, the response times for Ordering and Browsing profiles are approximately the same for 50, 100, 150 customers. For 200 customers the response time in Ordering profile is higher than in Browsing profile. This is due to the fact that the CPU utilization of the machine hosting the Web server has slightly higher utilization for Ordering than for Browsing profile. The response times of Shopping Cart and Customer interactions as shown in Figures 6 and 7 show similar behavior.

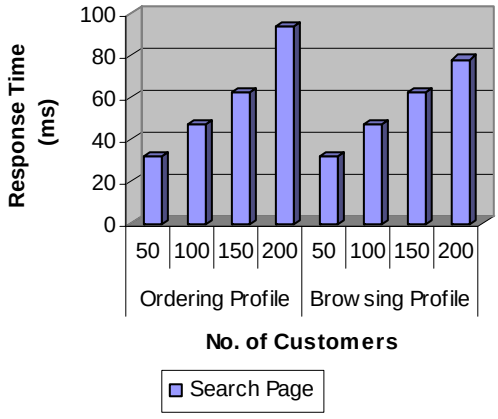

Figure 5: Response time for Search Web interaction

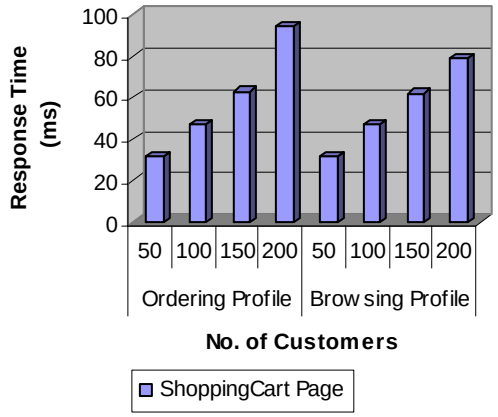

Figure 6: Response time for Shopping Cart interaction

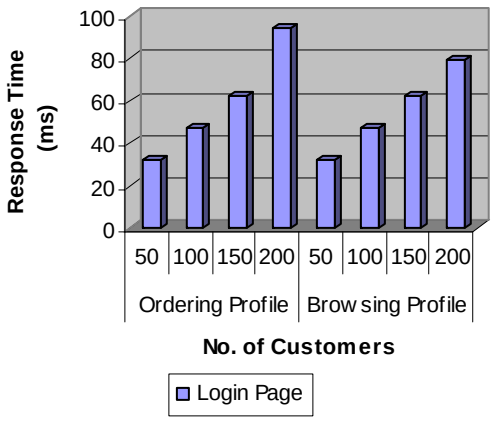

Figure 7: Response times for Login interaction

Next, we discuss the performance of the Home interaction which involves processing of Customer-EJB component and the html content of the Home page. The contributions of each component to the overall response times of Home interactions for both profiles and different number of customers are shown in Figure 8. It is obvious that the response times increase almost linearly with the increase of the number of clients for both profiles. The response times for Ordering profile, however, are approximately 10 % higher than for Browsing profile. It should be noted that in our implementation, the Customer-EJB component retrieves customer information from the database only during first visit to the Home page. In all subsequent requests, the Customer-EJB does not make database calls; hence the calls to this component are relatively inexpensive.

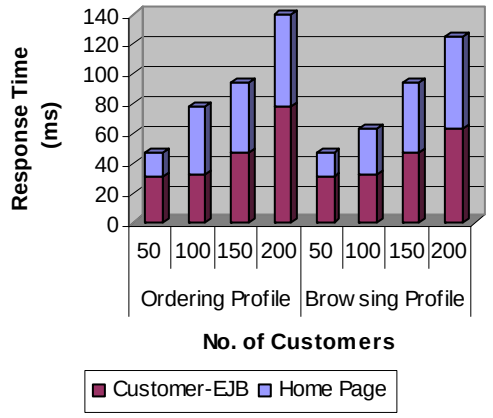

Figure 8: Response times for Home Web interaction

Finally, we consider the remaining three interactions, Search Results, Credit Check, and Process Order, which involve calls to Web services components. Figures 9, 10 and 11 show the distribution of the response times of these interactions across different components used to process the corresponding interaction, for the two workload profiles under different workload intensities. Several common observations can be drawn from Figures 9, 10 and 11.

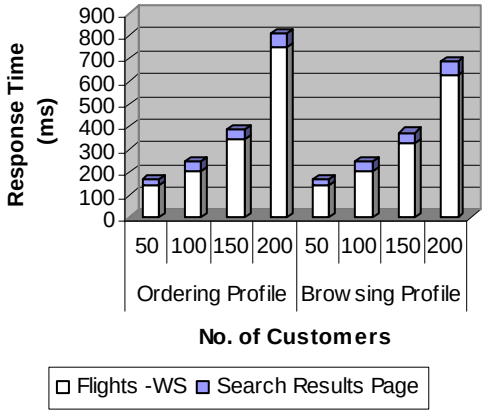

Figure 9: Response times for Search Results Web interaction

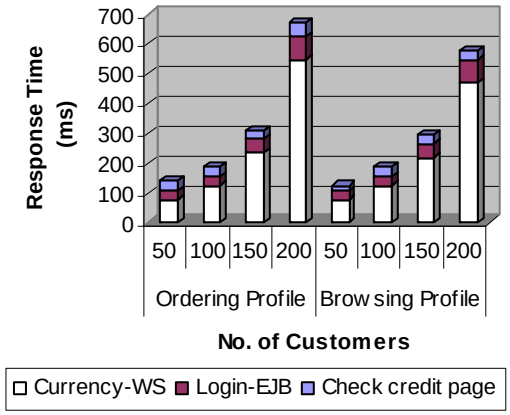

Figure 10: Response times for Credit Check Web interaction

First, for each interaction the overall interaction response times, as well as the corresponding components' response times are approximately the same for the both profiles when the workload consists of 50, 100 and 150 clients. For each profile, the response time in case of 200 clients is significantly higher than the response time for 150clients. This increase is mainly due to the increase of the response time of Web

services components. Furthermore, the response times of Web services components in the Ordering profile are nearly 20% higher than in the Browsing profile when the workload intensity is 200 clients. Next, for all workload intensities and both profiles 60-80% of the overall response times of Search Results, Credit Check, and Process Order interactions is spent in executing Web services components. It is interesting to notice that the values of the response times of Web services components are much higher than the response times of any other component in any interaction. Thus, it follows that the Web service components are the performance bottlenecks not only in Search, Credit Check, and Process order interactions, but in the whole system as well.

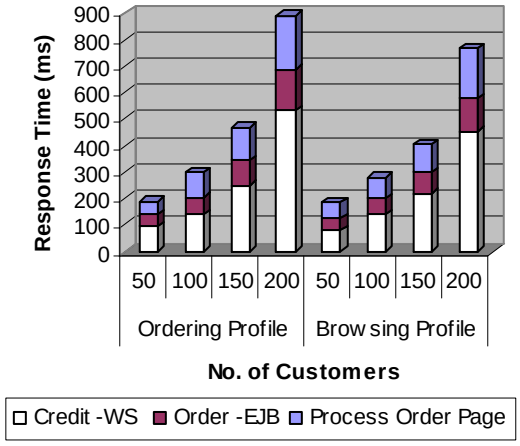

Figure 11: Response times for Process Order Web interaction

Web service calls are expensive because they communicate by XML based protocols such as SOAP. This type of communication requires Web service endpoint to convert the SOAP request messages into method calls to local objects, as well as to encode the results into SOAP messages before they can be transmitted to the Web service client. These parsing and encoding activities incur additional overhead on the performance of the system. Since parsing and encoding of XML messages are CPU intensive activities we analyze the CPU utilization on the machine where Web services components are deployed (i.e., Application Server 2 in Figure 2).

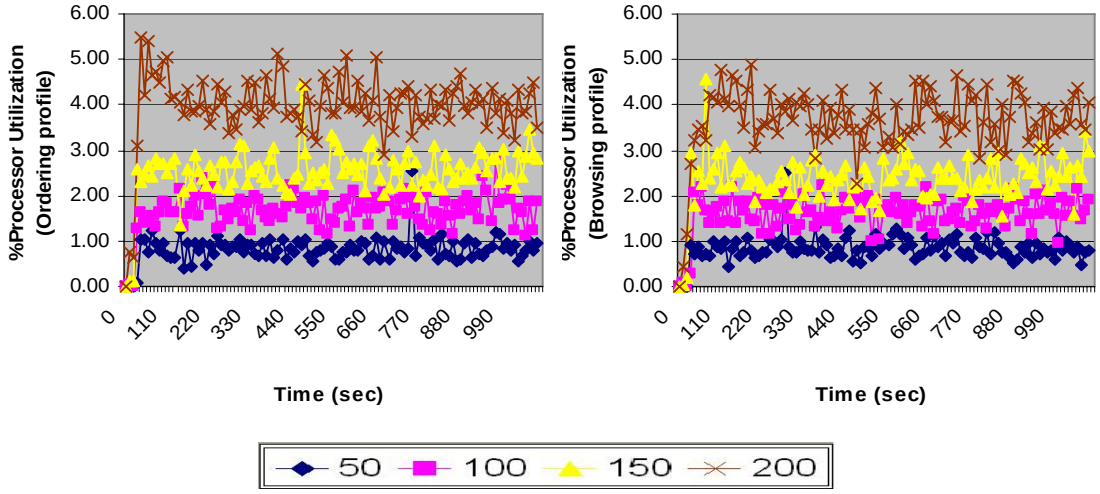

Figure 12: CPU Utilization in Ordering and Browsing Profiles at Application Server 2

As expected, we observe from Figure 12 that the CPU utilization increases with the number of clients for both Ordering and Browsing profiles. More interesting observation, however, is that the increase in CPU utilization for 200 clients with respect to 150 clients is significantly higher than between other workload intensities (i.e., 150 and 100 clients, or 100 and 50 clients). Obviously, one of the reasons for increased response time of Web services components under higher workload is the overhead due to parsing and encoding the XML messages which leads to increased CPU utilization.

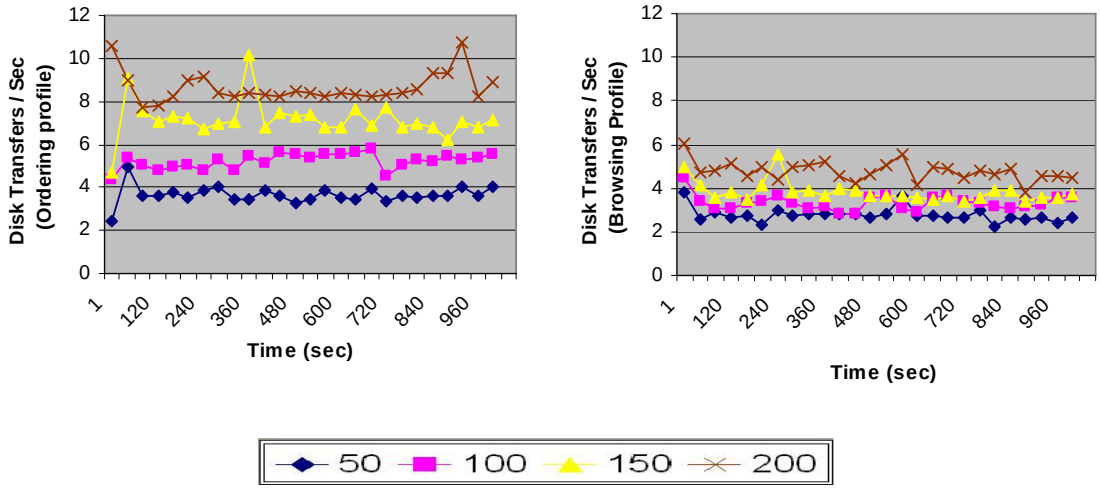

Figure 13: Database Disk activity in Ordering and Browsing Profile at Server 3

We also study the disk activity on the Database server (i.e., Server 3 in Figure2) because the Web services and EJB components in our application perform operations on the backend database to serve user requests. It can be observed from Figure 13 that the number of disk transfers per second for Ordering profile are higher than for Browsing profile regardless of the number of customers. Furthermore, the difference increases with the workload intensity, which clearly explains the increase in response times of Web services and EJB components.

Chapter 8: Performance of .NET and J2EE Web services

8.1 Prototype Description

For comparing the performance of J2EE and .NET Web services, we developed two versions of the e-commerce application described in Chapter 4. In the first version the Web services were implemented in Java using JAX–RPC API and deployed in Tomcat 5.0 Web container. In the second version we used ASP.NET to implement the Web services components. The .NET Web services were deployed on IIS 5.0 Web server. Since our intention is to compare the performance of Web services we keep the rest of the application architecture same in our experiments.

8.2 Experimental Results

Performance of .NET and J2EE versions of the application was analyzed by exercising the components with TPC-W based workload generator which was described in chapter 5. Next, we compare performance of Web services implemented using .NET and J2EE. Figures 14, 15 and 16 show the performance of .NET and J2EE Web services in Ordering and Browsing profiles. From the results we observe that in both the profiles our J2EE and .NET Web services shows similar performance characteristics. For .NET version of Flights-WS, Currency-WS and Credit-WS components the difference of response times in Ordering and Browsing profiles is less than 5%. In case of J2EE version the response times of these Web services in Ordering profile are approximately 10% higher than in Browsing profile. The results shows that .NET Web services perform slightly better in Ordering profile especially for higher workload intensities (150, 200 clients). One of the reasons for performance improvements in .NET is that we run all our experiments in Windows environment and the IIS Web server which processes requests for .NET Web services is tightly integrated with the Windows operating system.

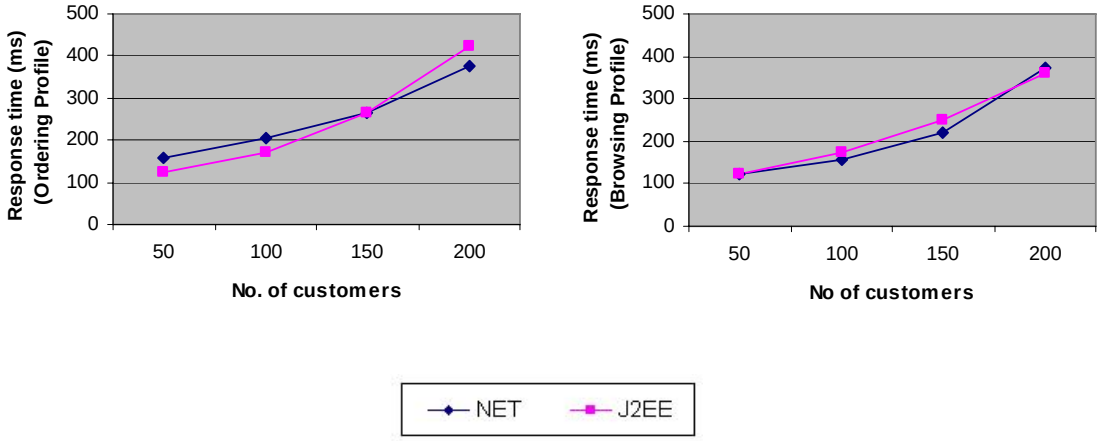

Figure 14: Performance of Flights Web service in J2EE and .NET

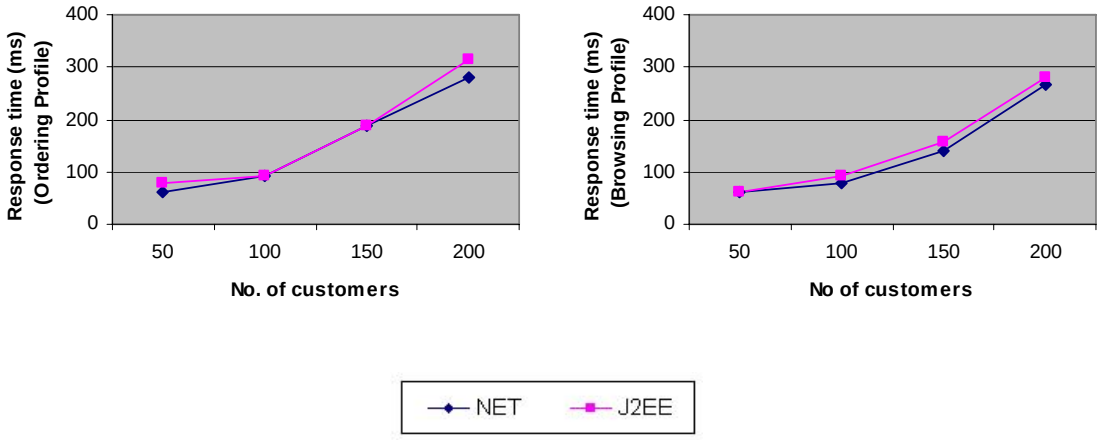

Figure 15: Performance of Currency Web service in J2EE and .NET

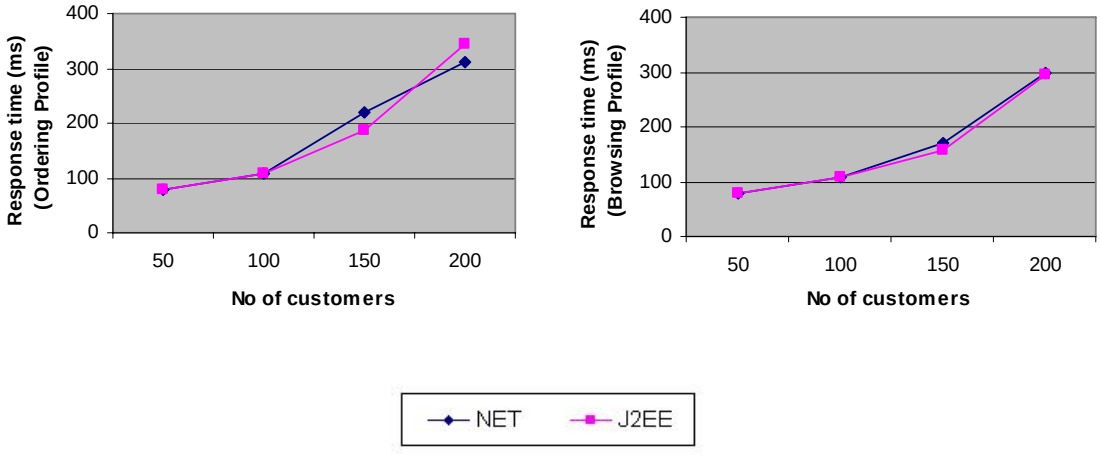

Figure 16: Performance of Credit Web service in J2EE and .NET

Chapter 9: Conclusion

In this thesis we present a measurement-based performance analysis of an e-commerce application which includes Web services components in the business logic layer. The experimental setup includes a prototype of an online travel agency with a three tier architecture deployed on several machines and a workload generator developed accordingly to the TPC-W specification. The empirical results are obtained for two different workload profiles, Ordering and Browsing, under different workload intensities of 50, 100, 150, and 200 clients.

In contrast to the related work which evaluated the overall application response time, our study includes measurements and analysis of server-side performance at different levels.

- Software architectural level allows us to study the distribution of the Web interactions response time among different components used to process the interaction.

- Hardware resource level provides additional insights and helps explaining the observed phenomena.

The results show that Web services components tend to become bottlenecks in the system, particularly in heavy load conditions. This phenomenon is attributed to the overhead introduced by the additional processing of the XML messages and, basically, is the price paid for the interoperability and flexibility of integration. One of the solutions to this problem is to develop more efficient XML parsers. Also, the application server vendors should incorporate better mechanisms to perform encoding and decoding of SOAP messages.

Another interesting observation is that under higher workload the response time for the Ordering profile becomes significantly worse than the response time for the Browsing profile. This is an important observation due to the fact that the customers in the Ordering profile generate more revenue to the organization as they have higher

purchasing activity. The main reasons for worse response time in Ordering profile are the higher database activity and contention for database resources which affect the performance of the EJB components and even more the performance of Web services components. To improve the performance of components that access the backend databases, application developers can use techniques such as database connection pooling.

In summary, analyzing the performance of e-commerce applications at different levels (i.e., Web interaction, software architecture, and hardware resource levels) provides insightful information about potential bottlenecks (i.e., software components and hardware resources) and enables system designers and application developers to improve performance in a cost effective manner. The wide adoption of new technologies such as Web services, to large extent, will depend on the capability to assess and even more to provide guarantees for their QoS. We believe that the research work presented in this thesis is a step towards this goal.

References

[1] M. Andreolini, V. Cardellini and M. Colajanni, “Benchmarking Models and Tools for Distributed Web-server systems”, *Performance Evaluation of Complex Systems: Techniques and Tools, Lecture Notes in Computer science*, *Springer-Verlag*, Sep. 2002, Vol.2459, pp. 208-235.

[2] G. Banga and P. Druschel, “Measuring the Capacity of a Web Server under Realistic Loads”, *World Wide Web*, May 1999, pp. 69-89.

[3] J. Cardoso, J. Miller, A. Sheth, and J. Arnold, “Modeling Quality of Service for Workflows and Web Service Processes”', Technical Report, LSDIS Lab, Computer Science Department, The University of Georgia.

[4] C. Catley, D. Petriu and M. Frize, “Software Performance Engineering of a Web Service-based Clinical Decision Support Infrastructure”', *4th International Workshop on Software and Performance*, Redwood Shores, California, 2004, pp. 130-138.

[5] E. Cecchet, J. Marguerite, W. Zwaenepoel , “Performance and scalability of EJB applications”, *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Seattle, 2002.

[6] S. Chandrasekaran, J. Miller, G. Silver, B. Arpinar, and A. Sheth, “Performance Analysis and Simulation of Composite Web Services”, *International Journal of Electronic Commerce & Business Media* Vol.13, No.2, 2003, pp.18-30.

[7] M. Chen, E. Kiciman, E. Fratkin, E.Brewer and A. Fox. “Pinpoint: Problem Determination in Large, Dynamic, Internet Services”, *Proceedings of the International Conference on Dependable Systems and Networks (IPDS Track)*, Washington D.C., 2002.

[8] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web Services Web", *IEEE Internet Computing*, March-April 2002, pp. 86-93.

[9] D. Garcia and J. Garcia, "TPC-W E-commerce Banchmark Evaluation", *IEEE Computer*, February 2003, pp. 42-48.

[10] I.Gorton, "Evaluating the Performance of EJB Components", *IEEE Internet Computing*, May/June 2003, pp.18-23.

[11] K. Juse, S. Kounev, and A. Buchmann, "PetStore-WS: Measuring the Performance Implications of Web Services", *29th International Conference of the Computer Measurement Group (CMG) on Resource Management and Performance Evaluation of Enterprise Computing Systems*, December 2003.

[12] S. M. Kim and M. Rosu, "A Survey of Public Web Services", *13th International World Wide Web Conference*, New York, USA, May 2004, pp. 312-313.

[13] S. Kounev and A. Buchmann, "Performance Modeling and Evaluation of Large-Scale J2EE Applications", *29th International Conference of the Computer Measurement Group (CMG) on Resource Management and Performance Evaluation of Enterprise Computing Systems*, December 2003.

[14] D. Menasce, V. A. F. Almeida, R. Foneca, and M. A. Mendes, "Business-oriented Resource Management Policies for E-commerce Servers", *Performance Evaluation*, Vol.42, No.2-3, 2000, pp. 223-239.

[15] D. Menasce, "TPC-W, A Benchmark for E-Commerce", *IEEE Internet Computing*, May-June 2002, pp. 83-87.

[16] G. Miller, "NET vs. J2EE", *Communications of the ACM*, June 2003, pp. 64-67.

[17] D. Mosberger and T. Jin, “httperf -- A tool for measuring Web server performance”, *ACM Performance Evaluation Review*, Dec. 1998, pp. 31-37.

[18] B. Paul and C. Mark, “Generating Representative Web Workloads for Network and Server Performance Evaluation”, *ACM SIGMETRICS*, June 1998, Madison, WI, pp. 151-160.

[19] S. Run, “A Model for Web Services Discovery with QoS”, *ACM SIGecom Exchanges*, Vol.1, Issue 1, March 2003, pp.1-10.

[20] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller, “Performance Impact of Web Services on Internet Servers”, *International Conference on Parallel and Distributed Computing and Systems*, Marina Del Rey, USA, Nov. 2003.

[21] A. Tsalgatidou and T. Pilioura, “An Overview of Standards and Related Technology in Web Services”, *Distributed and Parallel Databases*, Vol.12, 2002, pp.135-162.

[22] Airfares Web service endpoint: http://ws.netviagens.com/webservices/AirFares.asmx

[23] CurrencyConverter service: http://www.webservicex.net/CurrencyConvertor.asmx

[24] Frank Cohen, “Discover SOAP encodings impact on Web service performance”, http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc/

[25] GNU AWK utility, http://www.gnu.org/software/gawk/gawk.html.

[26] Jeffrey Fulmer, Siege -- An Open Source Stress Tester, 2002. http://www.joedo.org/siege/index.html.

[27] OASIS, Organization for the Advancement of Structured Information Standards.

http://www.oasis-open.org

[28] OASIS Web Services Reliable Messaging, http://docs.oasis-open.org/wsrm/2004/06/WS-Reliability-CD1.086.pdf.

[29] Sean MacRoibeaird, Universal Description, Discovery and Integration, http://wwws.sun.com/software/xml/developers/uddi/

[30] Standard Performance Evaluation Corp, SPECweb99. http://www.spec.org/osg/web99.

[31] Sun Java™ System Application Server Platform Edition 8 Administration Guide, Logging. http://docs.sun.com/source/817-6088/logging.html.

[32] Sun Microsystems -- Java 2 Platform Enterprise Edition Specification v1.4, http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf.

[33] Sun Microsystems, Java Logging API's, http://java.sun.com/j2se/1.4.2/docs/guide/util/logging.

[34] Sun Microsystems, Inc. Java Petstore Application. Documentation. http://java.sun.com/blueprints/code/\\jps131/docs/index.html.

[35] TPC-W Transactional Web Commerce Benchmark, Transaction Processing Performance Council. http://www.tpc.org/tpcw.

[36] Universal Description, Discovery and Integration (UDDI), OASIS Technical Report, http://xml.coverpages.org/uddi.html

[37] Web Services Security Specification, http://www-106.ibm.com/developerworks/webservices/library/ws-secure.

[38] WebStone. http://mindcraft.com/webstone

[39] D Venu, K Goseva-Popstojanova, “Measurement based Performance Analysis of E-commerce Applications with Web services components”, *IEEE International Conference on E-business Engineering*, 2005.

Appendix 1

Application Server log format for SUN J2EE 1.4 Application Server:
The J2EE1.4 application server stores log information in the file 'server.log'. The file is located in the 'logs' directory. The application server uses the java.util.logging API to log messages. Each log record has the following format:

[#|yyyy-mm-dd*T*hh*:*mm*:*ss.SSS-Z|LogLevel|ProductName_Version|LoggerName|*K*eyValuePairs |MessageId *:*Message|*#]*

1) Each record is delimited by the characters [# and #].

2) The attributes of the record are separated by '|' character.

3) The first field of the record contains timestamp in the format '*yyyy-mm-ddThh:mm:ss.SSS-z* . SSS denotes the millisecond and z denotes the time zone.

4) The *Log Level* indicates the priority or importance of the message. This application server identifies seven log levels- FINEST, FINER, FINE, CONFIG, INFO, WARNING, SEVERE. The default log level is the INFO level.

5) The *productName_Version* for this application server is 'j2ee-appserver1.4'.

6) *Logger name* is the name of the logger object a j2ee component uses to log the message.

7) The *Key value pairs* are key names and values, typically a thread ID such as _Thread=14.

8) Each message is identified by a unique *Message ID*. The message ID has the format <Subsystem><4CharacterIntegerID>. The subsystem is a module that generates the log messages.

The subsystems are:

ADM	–	Admin
ACC	–	Application client container
CORE	-	Core
DPL	–	Deployment
DTX	–	Java transactions API
EJB	–	Enterprise java bean
Install	–	Installer
IOP	–	Internet Inter-ORB protocol
JMS	–	Java messaging service
JTS	–	Java transaction services
LCM	–	Life cycle module
LDR	–	Class loader
MDB	–	Message driven bean container
RAR	–	Resource Adapter
SEC	–	Security services
VRFY	–	Verifier tool
UTIL	–	Utility services
WEB	–	Web container

The log settings for the application server can be modified from the admin console or by making changes to server.xml file in the config directory of the domain. The users can change log levels for each subsystem. An application can customize the log messages by adding custom log handlers.

Sample server log entries:

[#|2004-03-18T20:00:13.812-0500|INFO|j2ee-appserver1.4|javax.enterprise.system.tools.admin|_ThreadID=10;|ADM1041:Sent the event to instance:[ApplicationDeployEvent -- deploy __ejb_container_timer_app]|#]

Appendix 2

1. WSDL for Flights Web service

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="AirFaresService" targetNamespace="http://157.182.194.109:18080/Airfares"
    xmlns:tns="http://157.182.194.109:18080/Airfares" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
- <types>
- <schema targetNamespace="http://157.182.194.109:18080/Airfares"
    xmlns:tns="http://157.182.194.109:18080/Airfares" xmlns:soap11-
    enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
- <complexType name="ArrayOfstring">
- <complexContent>
- <restriction base="soap11-enc:Array">
  <attribute ref="soap11-enc:arrayType" wsdl:arrayType="string[]" />
    </restriction>
    </complexContent>
    </complexType>
    </schema>
    </types>
- <message name="SearchFlights_getFares">
  <part name="String_1" type="xsd:string" />
  <part name="String_2" type="xsd:string" />
  <part name="String_3" type="xsd:string" />
  <part name="String_4" type="xsd:string" />
  <part name="String_5" type="xsd:string" />
    </message>
- <message name="SearchFlights_getFaresResponse">
  <part name="result" type="tns:ArrayOfstring" />
    </message>
- <portType name="SearchFlights">
- <operation name="getFares" parameterOrder="String_1 String_2 String_3 String_4 String_5">
  <input message="tns:SearchFlights_getFares" />
  <output message="tns:SearchFlights_getFaresResponse" />
    </operation>
    </portType>
- <binding name="SearchFlightsBinding" type="tns:SearchFlights">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
- <operation name="getFares">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
      namespace="http://157.182.194.109:18080/Airfares" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
      namespace="http://157.182.194.109:18080/Airfares" />
    </output>
    </operation>
```

```
    </binding>
- <service name="AirFaresService">
- <port name="SearchFlightsPort" binding="tns:SearchFlightsBinding">
  <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
    </service>
    </definitions>
```

2. WSDL for Currency Web service

```
    <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="cservice" targetNamespace="http://157.182.194.109:18080/currency"
      xmlns:tns="http://157.182.194.109:18080/currency" xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types />
- <message name="CurrencyService_conversionRate">
  <part name="String_1" type="xsd:string" />
  <part name="String_2" type="xsd:string" />
    </message>
- <message name="CurrencyService_conversionRateResponse">
  <part name="result" type="xsd:double" />
    </message>
- <portType name="CurrencyService">
- <operation name="conversionRate" parameterOrder="String_1 String_2">
  <input message="tns:CurrencyService_conversionRate" />
  <output message="tns:CurrencyService_conversionRateResponse" />
    </operation>
    </portType>
- <binding name="CurrencyServiceBinding" type="tns:CurrencyService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
- <operation name="conversionRate">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
      namespace="http://157.182.194.109:18080/currency" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
      namespace="http://157.182.194.109:18080/currency" />
    </output>
    </operation>
    </binding>
- <service name="Cservice">
- <port name="CurrencyServicePort" binding="tns:CurrencyServiceBinding">
  <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
    </service>
    </definitions>
```

3. WSDL for Credit Web service

```
   <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="creditservice" targetNamespace="http://157.182.194.109:18080/credit"
      xmlns:tns="http://157.182.194.109:18080/credit" xmlns="http://schemas.xmlsoap.org/wsdl/"
```

```
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types />
- <message name="ccheck_check">
  <part name="String_1" type="xsd:string" />
  <part name="String_2" type="xsd:string" />
  <part name="String_3" type="xsd:string" />
      </message>
- <message name="ccheck_checkResponse">
  <part name="result" type="xsd:string" />
      </message>
- <portType name="ccheck">
- <operation name="check" parameterOrder="String_1 String_2 String_3">
  <input message="tns:ccheck_check" />
  <output message="tns:ccheck_checkResponse" />
      </operation>
      </portType>
- <binding name="ccheckBinding" type="tns:ccheck">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
- <operation name="check">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
        namespace="http://157.182.194.109:18080/credit" />
      </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
        namespace="http://157.182.194.109:18080/credit" />
      </output>
      </operation>
      </binding>
- <service name="Creditservice">
- <port name="ccheckPort" binding="tns:ccheckBinding">
  <soap:address location="REPLACE_WITH_ACTUAL_URL" />
      </port>
      </service>
      </definitions>
```