Measurement–based Performance Analysis of E–commerce Applications with Web Services Components

Venu Datla and Katerina Goševa–Popstojanova
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26506-6109
{venud, katerina}@csee.wvu.edu

Abstract

Web services are increasingly used to enable interoperability and flexible integration of e-business systems. In this paper 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 the TPC-W specification. The empirical results are obtained for two different user profiles, Browsing and Ordering, under different workload intensities. 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 e-commerce applications.

1. Introduction

Modern e-commerce 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 facilitate interoperability and flexible integration of systems developed on heterogeneous environments. The interface of a Web service is described in a machine processable format. Other 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 business-to-business applications. A brief description of the core standards used in Web services architecture is given next. For more detailed overview the reader is referred to [6], [19].

- *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. 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 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.
- *Universal Discovery, Description, and Integration* (UDDI) provides a standard way for businesses to publish and discover Web services. The UDDI specification consists of an XML schema for UDDI data structures and description of UDDI APIs specifications.

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

Proceedings of the 2005 IEEE International Conference on e-Business Engineering (ICEBE'05)
0-7695-2430-3/05 $20.00 © 2005 IEEE

incorporated support for Web service standards in their specifications. Facilities for Web services development in .NET and J2EE platforms are compared in [14]. 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 [24]) and integrity and confidentiality (WS-Security [31]), 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 paper we present a measurement–based study of performance of e-commerce applications that use 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. The paper is organized as follows. In section 2 we discuss the related work on performance evaluation of Web services and emphasize our contributions. The description of the prototype, including the software architecture, implementation, and deployment details, is given in section 3. The workload used in our experiments and the measurement methodology are described in sections 4 and 5, respectively. Section 6 presents the experimental results. Finally, the concluding remarks are given in section 7.

2. Related work and our contributions

Performance is an important quality aspect of Web services because of their distributed nature. Surprisingly, only a few papers 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 [29], one implemented using Java Messaging Service (JMS) and the other using Web services, were studied in [9]. In this work the workload was generated using the Siege tool [23] and it was shown that the JMS version has better performance than the Web services version of the application. A similar study was presented in [18]. The authors empirically compared two versions of an electronic book inventory system implemented using Active Server Pages (ASP) and Web services. The workload generator used in this study was S-client [2]. 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 [11]. Layered Queuing Network (LQN) model is used in [4] to calculate response times of a Web service based clinical decision support system. The LQN model is built based on the software architecture. The model is not validated with actual measurements. Queuing network model for performance evaluation of an e-commerce application was proposed in [11]. 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 modelling 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.

In this paper 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, 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 [5], [18] are not suitable for our application. Therefore, we have developed a session-based workload generation tool based on TPC-W benchmark specification [30]. TPC-W is oriented toward business-to-consumer e-commerce interactions and tests many important elements of most e-commerce applications [13]. 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 [7]. 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], [9], and [18] which analyzed the overall throughput and response times of Web service based applications, we measure the performance at ar-

chitectural 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 [22] 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.

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 [9] and [18], different workload profiles were not considered. The empirical results presented in this paper 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 [12] which was focused on priority-based resource management policies aimed at increasing the business-oriented metrics such as revenue per second.

3. Prototype description

In this section 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.

3.1. 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 [20] 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 [21].
- `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`, and `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.

3.2. Implementation details

Our online travel agency application is implemented using J2EE [27], 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). We use J2EE 1.4 Application Server which supports Web services in the form of JAX-RPC API 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.

3.3. Deployment details

The UML deployment diagram of our prototype application is shown in Figure 1. 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 3.1, 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 [17] it was reported that in 2001 48% of the production UDDI registry had links that were unusable. A more recent study [10] 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.

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.

4. Workload description

A key issue in performance evaluation of software systems is the workload characterization which should closely represent the behavior of real users. Our previous work was focused on detailed characterization of the Web workload in terms of user sessions based on data extracted from actual Web logs of ten Web servers [8]. In this paper we take a different approach - we analyze the performance of e-commerce applications using synthetically generated workload which allows us to run controlled experiments. An excellent survey presented in [1] analyzes in details popular Web workload benchmarking tools such as httperf [15], SPECweb99 [25], Surge [16], S-Client [2], TPC-W [30], and Web Stone [32].

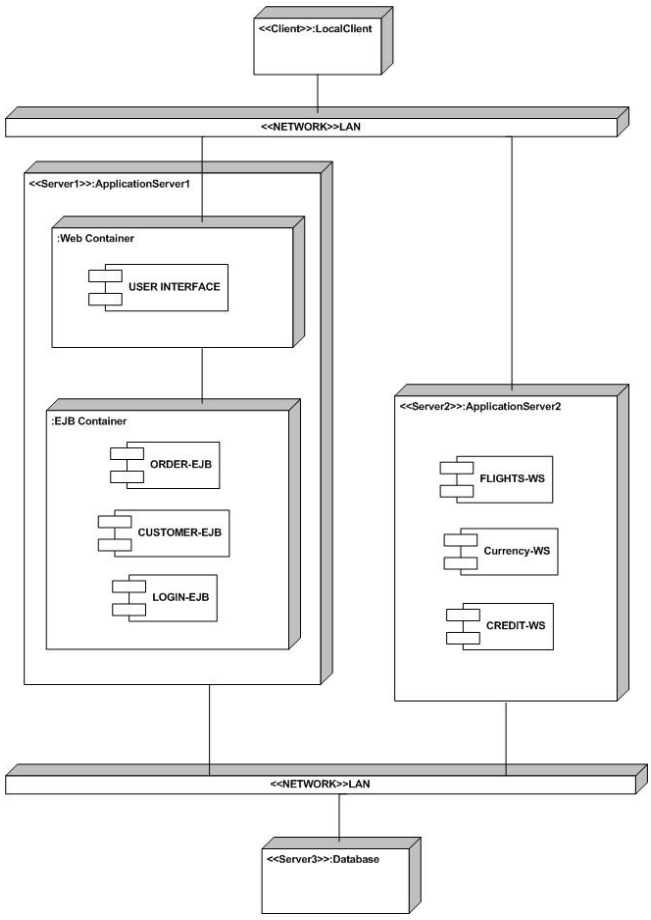

Figure 1. UML deployment diagram of the travel agency application

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 [30], 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 [7], [13]. Its

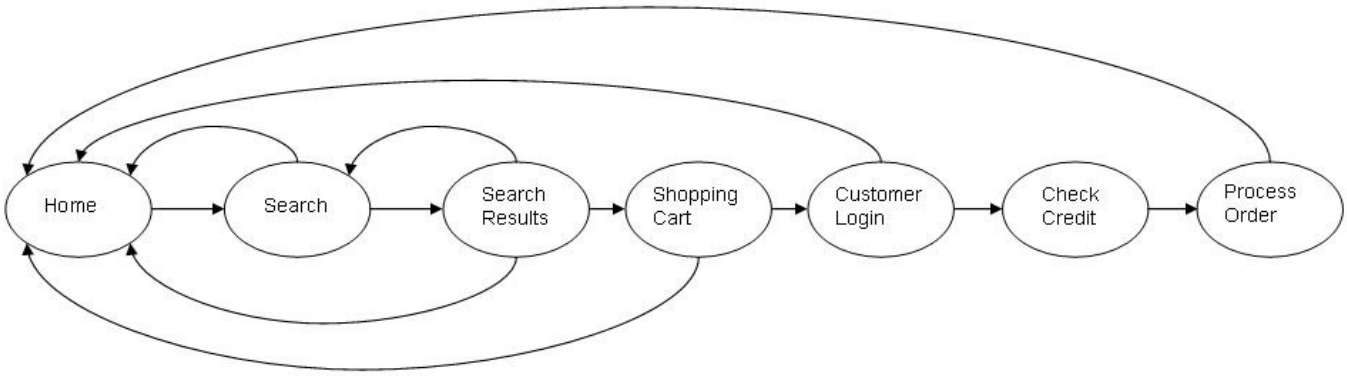

Figure 2. DTMC for the travel agency application

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 [7]. 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 [13] 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 2. 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.

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 paper we present 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 1.

The workload generated accordingly to TPC-W specification consists of three phases: ramp–up interval, steady–state interval, and ramp–down interval [30]. 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

	Browsing profile (79-21)	Ordering profile (50-50)
Browse	79%	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 1. Mixes of Web interactions for Browsing and Ordering profiles

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 2 and requires reporting of the 90th percentile response time during the steady–state measurement interval.

5. 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 2 involve executing from one to three different software components (see Section 3.1) 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 [28]. 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 3 [26]. 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.

```
[#|yyyy-mm-ddThh:mm:ss.SSS- Z | LogLevel| ProductName_Version |LoggerName
|Key Value Pairs |MessageId: Message|#]
```

Figure 3. 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 [22] 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 1), 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*).

6. 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 4, 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.

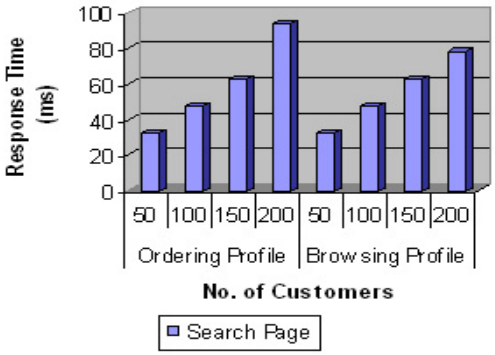

Figure 4. Response times for Search Web interaction

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 5. It is obvious that the response times increase almost linearly with the increase of the number of clients for both profiles. The response times in `Ordering` profile, however, are approximately 10% higher than in `Browsing` profile. It should be noted that in our implementation, the `Customer-EJB` component retrieves customer information from the database only during the 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.

Finally, we consider the remaining three interactions, `Search Results`, `Credit Check`, and `Process Order`, which involve calls to Web services components.

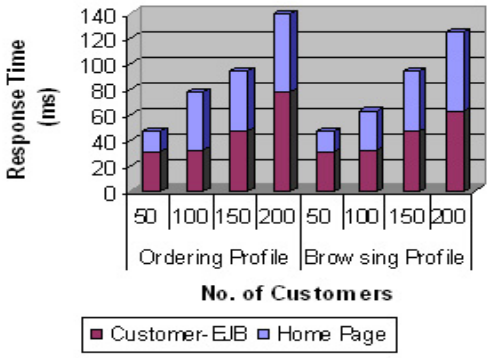

Figure 5. Response times for Home Web interaction

Figures 6, 7, and 8 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 6, 7, and 8. First, for each interaction the overall interaction response time, 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 150 clients. 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.

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 1). As expected, we observe from Figure 9 that the CPU utilization increases with the number of clients for both `Ordering` and

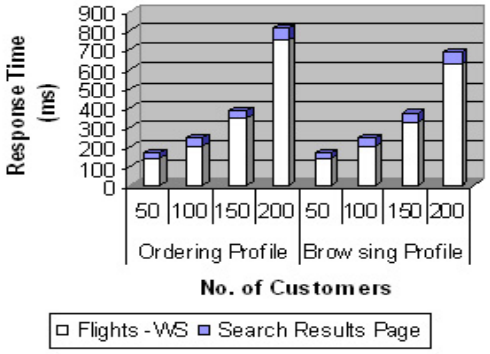

Figure 6. Response times for Search Results Web interaction

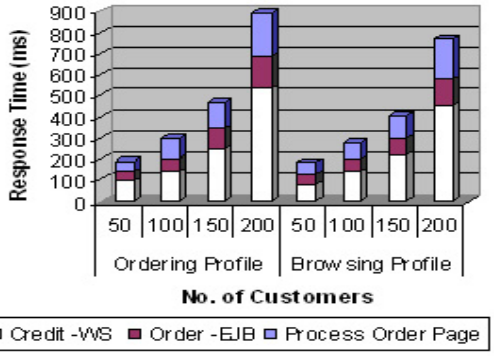

Figure 8. Response times for Process Order Web interaction

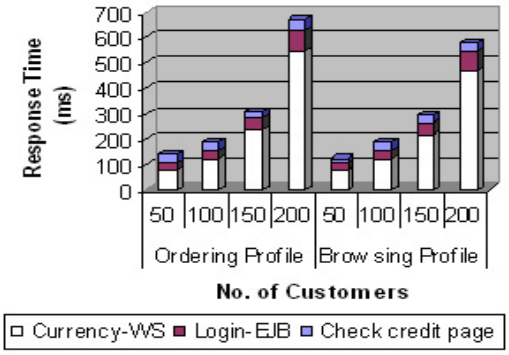

Figure 7. Response times for Credit Check Web interaction

`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.

We also study the disk activity on the Database server (i.e., Server 3 in Figure 1) 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 10 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.

7. Conclusion

In this paper 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 the 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,

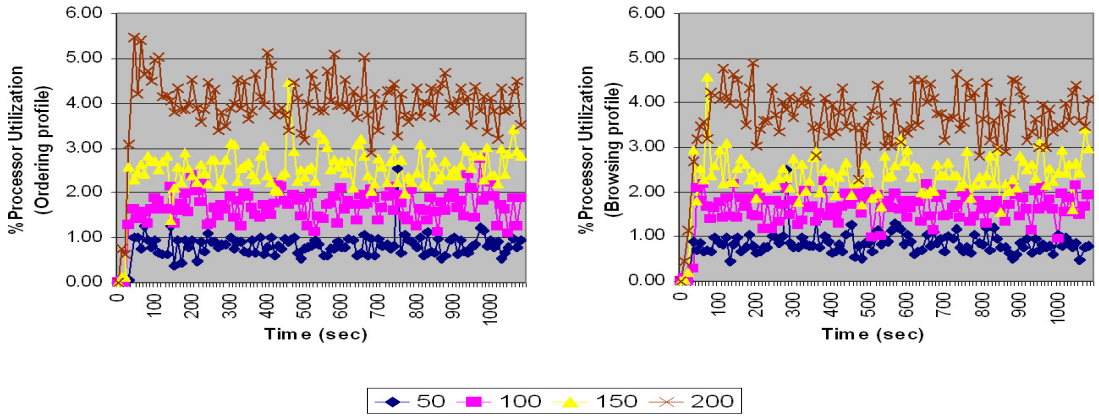

Figure 9. CPU utilization in Ordering and Browsing Profiles at Application server 2

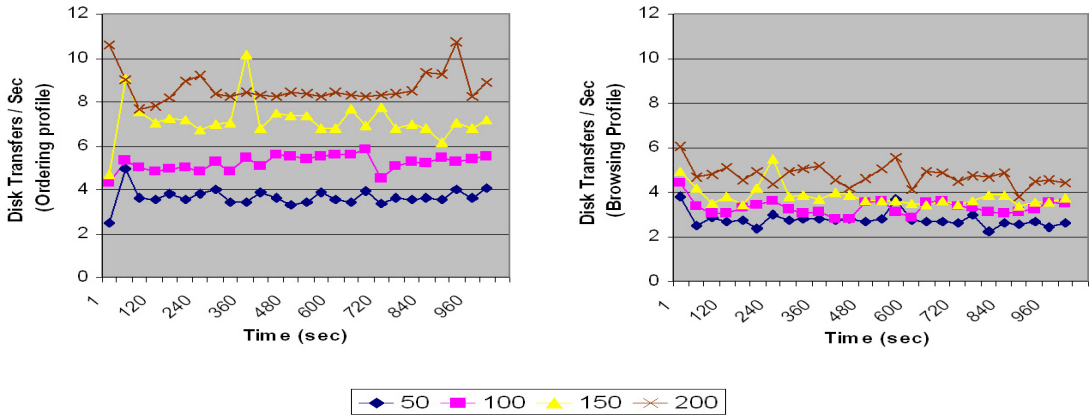

Figure 10. Database Disk activity in Ordering and Browsing Profiles at Server 3

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 paper is a step towards this goal.

8. Acknowledgement

This work is funded in part by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) under grant managed through the NASA IV&V Facility, Fairmont, West Virginia and by the National Science Foundation under CAREER grant number CNS-0447715.

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] 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.

[6] 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.

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

[8] K. Goševa–Popstojanova, S. Mazimdar, and A. Singh, "Empirical Study of Session-based Workload and Reliability for Web Servers", *15th IEEE International Symposium on Software Reliability*, Saint-Malo, France, November 2004, pp. 403–414.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] D. Mosberger and T. Jin, "httperf – A tool for measuring Web server performance", *ACM Performance Evaluation Review*, Dec. 1998, pp. 31–37.

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

[17] S. Run, "A Model for Web Services Discovery with QoS", *ACM SIGecom Exchanges*, Vol.1, Issue 1, March 2003, pp. 1–10.

[18] 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.

[19] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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