International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)
ISSN: 0976-1353 Volume 22 Issue 1 – MAY 2016.

RESPONSE TIME BASED OPTIMAL WEB SERVICE SELECTION

Miss. A.Nandhini [1] Mr.P.V.Sankar Ganesh [2]

Abstract--**In this paper, we propose a novel method for Quos metrification based on Hidden Markov Models (HMM), which further suggests an optimal path for the execution of user requests. The technique we show can be used to measure and predict the behavior of Web Services in terms of response time, and can thus be used to rank services quantitatively rather than just qualitatively. We demonstrate the feasibility and usefulness of our methodology by drawing experiments on real world data. The results have shown how our proposed method can help the user to automatically select the most reliable Web Service taking into account several metrics, among them, system predictability and response time variability. For Internet services, the presence of low-performance servers, high latency or overall poor service quality can translate into lost sales, user frustration, and customers lost.**

I. INTRODUCTION

The Internet made the world a smaller place. Companies from all around the world may now compete over different service offerings not only with their local adversaries, but do now under a global scale. Escalating the competition and lead in industry segment can often be a matter of offering and, perhaps even most importantly, assuring the good quality of the services offered. In the Web this should be no different; controlling quality for Web Services (WS) is done by enforcing Quality of Service (QoS) policies and assuring needed quality conditions are always met. In this paper, we propose a novel method for Quos metrification based on Hidden Markov Models (HMM), which further suggests an optimal path for the execution of user requests. The technique we show can be used to measure and predict the behavior of Web Services in terms of response time, and can thus be used to rank services quantitatively rather than just qualitatively. We demonstrate the feasibility and usefulness of our methodology by drawing experiments on real world data. The results have shown how our proposed method can help the user to automatically select the most reliable Web Service taking into account several metrics, among them, system predictability and response time variability. For Internet services, the presence of low-performance servers, high latency or overall poor service quality can translate into lost sales, user frustration, and customers lost.

II. LITERATURE SURVEY

1. Architecture-based Dependability Prediction for Service-oriented Computing

In service-oriented computing, services are built as an assembly of pre-existing, independently developed services. Hence, predicting their dependability is important to appropriately drive the selection and assembly of services, to get some required dependability level. We present an approach to the dependability prediction of such services, exploiting ideas from the Software Architecture- and component-based approaches to software design.

2. QoS Analysis forWeb Service Composition

The quality of service (QoS) is a major concern in the design and management of Web service composition. Existing methods for QoS calculation either do not take the probability of path execution into consideration when QoSs are provided for different execution paths, or do not take different execution paths into consideration when a single integrated QoS is provided for the whole composition. In this paper, a comprehensive QoS analysis approach is proposed that calculates the QoS probability distribution by considering both the execution probability and execution conditions of each path in the service composition.

3. Run-Time Monitoring in Service-Oriented Architectures

Modern software architectures are increasingly dynamic. Among them, Service-Oriented Architectures (SOAs) are becoming a dominant paradigm. SOAs allow components to be exported as services for external use. Service descriptions (which include functional and non-functional properties) are published by service providers and are later discovered by potential users. Service discovery is based on matching the published service descriptions with the required service specifications provided by the user. Once an external service is discovered, it may be bound and invoked remotely. New services may also be created by composing existing services. To achieve full flexibility, the binding between a service request and a service provision may be set dynamically at run-time. In the new setting it extends to run-time and requires continuous monitoring of functional and non-functional attributes.

4. Architecture-Based Reliability Prediction for Service-Oriented Computing

In service-oriented computing, services are dynamically built as an assembly of pre-existing, independently developed, network accessible services. Hence, predicting as much as possible automatically their dependability is important to appropriately drive the selection and assembly of services, in order to get some required dependability level. We present an approach to the reliability prediction of such services, based on the partial information published with each service, and that lends itself to automatization. The proposed methodology

exploits ideas from the Software Architecture- and Component-based approaches to software design.

III. SYSTEM ANALYSIS

Existing system

In existing, they have focused on predicting reliability of various factors involved in building enterprise application, nonetheless, considered reliability of remote web service as constants. For remote web services the vender will provide probabilistic details about the flow of executing user requests.

Existing algorithm - CSPN model

Algorithm definition

This model deals more with design time problems and does not reflect the impact of problems that occur at runtime. They have studied in detail transactional dependency among different type of web services.

Disadvantages

- Quality of Service is not good.
- Response time calculation is not possible.

Proposed system

We propose a novel method for Quos metrification based on Hidden Markov Models (HMM), which further suggests an optimal path for the execution of user requests. The users can weigh their options directly and individually, for themselves.

Advantages

- Quality of Service is good.
- Response time calculation is possible.

Proposed algorithm

Hidden Markov Models.

Algorithm definition:

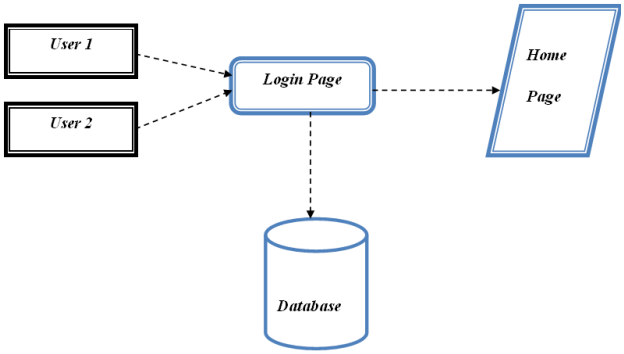
Building a directed graph among hidden states of component web services used in composition. Analyzing the current status of each vertex of directed graph i.e., underlying hidden states. Predicting hidden states' behavior in terms of response time during nth time interval t. Finally, selecting optimal web services used in composition based on hidden states' behavior.

Applications

- Weather Application
- Online shopping applications

IV. USER INTERFACE DESIGN

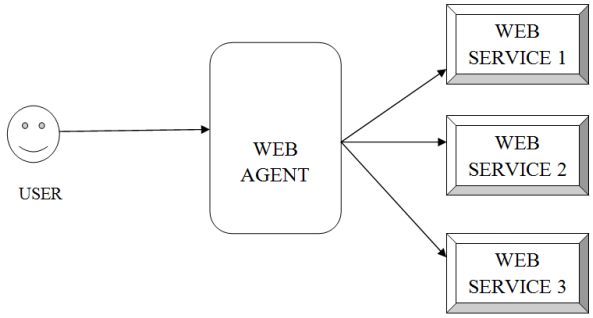
To connect with server user must give their username and password then only they can able to connect the server. If the user already exits directly can login into the server else user must register their details such as username, password, Email id, City and Country into the server. Database will create the account for the entire user to maintain upload and download rate. Name will be set as user id. Logging in is usually used to enter a specific page. It will search the query and display the query.



Fig; User interface design

V. WEBSITE VISITING

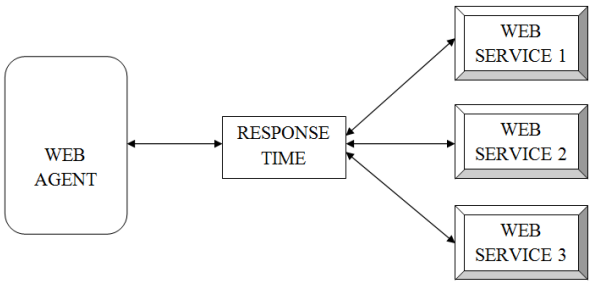
The Internet is supposed to be a global network that links the entire world, but many websites are confined to specific countries. Unsurprisingly, piracy is higher in countries where content isn't legally available. Some services work through some DNS wizardry. Web service selection is the action or fact of carefully choosing someone or something as being the best or most suitable. A process in which environmental or genetic influences determine which types of organism thrive better than others, regarded as a factor in evolution.



Fig; Website visiting

VI. RESPONSE TIME CALCULATION

Response time is the total amount of *time* it takes to respond to a request for service. That service can be anything from a memory fetch, to a disk IO, to a complex database query, or loading a full web page. Ignoring transmission *time* for a moment, the *response time* is the sum of the service *time* and wait *time*. *Response time* may refer to: The *time* lagged between the input and the output signal which depends upon the value of passive components used. *Response time* (technology), the *time* a generic system or functional unit takes to react to a given input. Responsiveness, how quickly an interactive system responds to user input.



Fig; Response Time Calculation

VII. ARCHITECTURE DIAGRAM

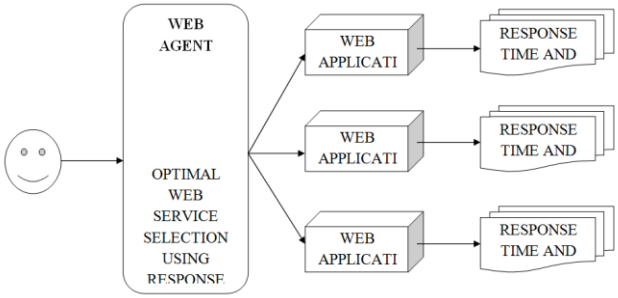


Fig: Architecture Diagram

Explanation

The systems architect establishes the basic structure of the system, defining the essential core design features and elements that provide the framework. The systems architect provides the architects view of the users' vision. Above diagram user first login to the account then he enter query and it search which are available in server and display query.

SOFTWARE DESCRIPTION

About JAVA

Java is a programming language originally developed by Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code that can run on any Java virtual machine (JVM) regardless of computer architecture.

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any supported hardware/operating-system platform. One should be able to write a program once, compile it once, and run it anywhere.

This is achieved by compiling the Java language code, not to machine code but to Java byte code – instructions analogous to machine code but intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a JRE installed on their own machine, or in a Web browser.

Standardized libraries provide a generic way to access host specific features such as graphics, threading and networking. In some JVM versions, byte code can be compiled to native code, either before or during program execution, resulting in faster execution.

A major benefit of using byte code is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executables would, and Java suffered a reputation for poor performance. This gap has been narrowed by a number of optimization techniques introduced in the more recent JVM implementations.

One such technique, known as (just-in-time compilation) JIT, translates Java byte code into native code the first time that code is executed, then caches it. This result in a program that starts and executes faster than pure interpreted code can, at the cost of introducing occasional compilation overhead during execution. More sophisticated VMs also use dynamic recompilation, in which the VM analyzes the behavior of the running program and selectively recompiles and optimizes parts of the program. Dynamic recompilation can achieve optimizations superior to static compilation because the dynamic compiler can base optimizations on knowledge about the runtime environment and the set of loaded classes, and can identify hot spots - parts of the program, often inner loops, that take up the most execution time. JIT compilation and dynamic recompilation allow Java programs to approach the speed of native code without losing portability.

PLATFORM INDEPENDENCE

Swing is platform independent both in terms of its expression (Java) and its implementation (non-native universal rendering of widgets).

EXTENSIBILITY

Swing is a highly partitioned architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations. In general, Swing users can extend the framework by extending existing (framework) classes and/or providing alternative implementations of core components.

COMPONENT-ORIENTED

Swing is a component-based framework. The distinction between objects and components is a fairly subtle point: concisely, a component is a well-behaved object with a known/specified characteristic pattern of behavior. Swing objects asynchronously fire events, have "bound" properties, and respond to a well-known set of commands (specific to the component.) Specifically, Swing components are Java Beans components, compliant with the Java Beans Component Architecture specifications.

RELATIONSHIP TO AWT

Since early versions of Java, a portion of the Abstract Window Toolkit (AWT) has provided platform-independent APIs for user interface components. In AWT, each component

is rendered and controlled by a native peer component specific to the underlying windowing system.

By contrast, Swing components are often described as *lightweight* because they do not require allocation of native resources in the operating system's windowing toolkit. The AWT components are referred to as *heavyweight components*.

Much of the Swing API is generally a complementary extension of the AWT rather than a direct replacement. In fact, every Swing lightweight interface ultimately exists within an AWT heavyweight component because all of the top-level components in Swing (JApplet, JDialog, JFrame, and JWindow) extend an AWT top-level container. However, the use of both lightweight and heavyweight components within the same window is generally discouraged due to Z-order incompatibilities.

The core rendering functionality used by Swing to draw its lightweight components is provided by Java 2D, another part of JFC.

VIII. SOCKET OVERVIEW

A *network socket* is a lot like an electrical socket. Various plugs around the network have a standard way of delivering their payload. Anything that understands the standard protocol can "plug in" to the socket and communicate.

Internet protocol (IP) is a low-level routing protocol that breaks data into small packets and sends them to an address across a network, which does not guarantee to deliver said packets to the destination.

Transmission Control Protocol (TCP) is a higher-level protocol that manages to reliably transmit data. A third protocol, *User Datagram Protocol (UDP)*, sits next to TCP and can be used directly to support fast, connectionless, unreliable transport of packets.

Client/Server

A *server* is anything that has some resource that can be shared. There are *compute servers*, which provide computing power; *print servers*, which manage a collection of printers; *disk servers*, which provide networked disk space; and *web servers*, which store web pages. A *client* is simply any other entity that wants to gain access to a particular server.

In Berkeley sockets, the notion of a socket allows as single computer to serve many different clients at once, as well as serving many different types of information. This feat is managed by the introduction of a *port*, which is a numbered socket on a particular machine. A server process is said to "listen" to a port until a client connects to it. A server is allowed to accept multiple clients connected to the same port number, although each session is unique. To manage multiple client connections, a server process must be multithreaded or have some other means of multiplexing the simultaneous I/O.

RESERVED SOCKETS

Once connected, a higher-level protocol ensues, which is dependent on which port you are using. TCP/IP reserves the lower, 1,024 ports for specific protocols. Port number 21 is for FTP, 23 is for Telnet, 25 is for e-mail, 79 is for finger, 80 is for HTTP, 119 is for Netnews-and the list goes on. It is up to each protocol to determine how a client should interact with the port.

Java and the Net

Java supports TCP/IP both by extending the already established stream I/O interface. Java supports both the TCP and UDP protocol families. TCP is used for reliable stream-based I/O across the network. UDP supports a simpler, hence faster, point-to-point datagram-oriented model.

InetAddress

The InetAddress class is used to encapsulate both the numerical IP address and the domain name for that address. We interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The InetAddress class hides the number inside. As of Java 2, version 1.4, InetAddress can handle both IPv4 and IPv6 addresses.

IX. FUTURE ENHANCEMENT

For future work, testing web services at the client side is not as straightforward as testing traditional software due to the complex nature of web services and the absence of source code. Surveys the previous work undertaken on web service testing, showing the strengths and weaknesses of current web service testing strategies and identifying issues for future work.

APPLICATIONS

- Weather Application
- Online shopping applications

X. CONCLUSION

We propose a probabilistic model for predicting response time of web service and then selected an optimal web service at runtime from the list of functionally equivalent web services. To know the probabilistic insight of WSs we have used HMM. In our model we have assumed that WS is deployed on a cluster of web servers and sometime the delay or crash during WS invocation is because the bad node in sever clustering responds to users' requests. With the help of HMM we have predicted the probabilistic behavior of these web servers and then selected the WS based on their probabilistic value.

REFERENCE

[1] V. Grassi, ''Architecture-Based Reliability Prediction for Service-Oriented Computing,'' in Architecting Dependable Systems III. Berlin, Germany: Springer-Verlag, 2005, pp. 279-299.

[2] V. Cortellessa and V. Grassi, ''Reliability Modeling and Analysis of Service-Oriented Architectures,'' in Test and Analysis of Web Services. Berlin, Germany: Springer-Verlag, 2007, pp. 339-362.

[3] G. Stefano, C. Ghezzi, R. Mirandola, and G. Tamburrelli, ‘‘Quality Prediction of Service Compositions through Probabilistic Model Checking,’’ in Proc. 4th Int’l Conf. Quality Software-Architect., Models Architect., 2008, pp. 119-134.
[4] D.A. Menasce, ‘‘Composing Web Services: A QoS View,’’ IEEE Internet Comput., vol. 8, no. 6, pp. 80-90, Nov. 2004.
[5] H. Zheng, J. Yang,W. Zhao, and A. Bouguettaya, ‘‘QoS Analysis for Web Service Compositions Based on Probabilistic QoS,’’ in Service-Oriented Computing. Berlin, Germany: Springer-Verlag, 2011, pp. 47-61.
[6] Z. Zibin and R.L. Michael, ‘‘Collaborative Reliability Prediction of Service-Oriented Systems,’’ in Proc. 32nd ACM/IEEE Int’l Conf. Softw. Eng., Cape Town, Africa, 2010, vol. 1, pp. 35-44.
[7] R. Perrone, R. Macedo, G. Lima, and V. Lima, ‘‘An Approach for Estimating Execution Time Probability Distributions of Component- Based Real-Time Systems,’’ J. Universal Comput. Sci., vol. 15, no. 11, pp. 2142-2165, 2009.
[8] M. Cristescu and L. Ciovica, ‘‘Estimation of the Reliability of Distributed Applications,’’ Inf. Econ., vol. 14, no. 4, pp. 19-29, 2010.
[9] D. Zhong, Z. Qi, and X. Xu, ‘‘Reliability Prediction and Sensitivity Analysis of WS Composition,’’ in Petri Net: Theory and Applications, V. Kordic, Ed. Rijeka, Croatia: Intech, 2008, pp. 459-470.
[10] J. El Haddad, M. Manouvrier, G. Ramirez, and M. Rukoz, ‘‘QoSDriven Selection of Web Services for Transactional Composition,’’ in Proc. IEEE ICWS, 2008, pp. 653-660.
[11] B. Sami, G. Claude, and P. Olivier, ‘‘Transactional Patterns for Reliable Web Services Compositions,’’ in Proc. 6th Int’l Conf. Web Eng., Palo Alto, CA, USA, 2006, pp. 137-144.
[12] L. Li, L. Chengfei, and W. Junhu, ‘‘Deriving Transactional Properties of Composite Web Services,’’ in Proc. IEEE ICWS, 2007, pp. 631-638.
[13] K. Boumhamdi and Z. Jarir, ‘‘A Flexible Approach to Compose Web Services in Dynamic Environment,’’ Int’l J. Digit. Soc., vol. 1, no. 2, pp. 157-163, 2010.
[14] Y. Tao, Z. Yue, and L. Kwei-Jay, ‘‘Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints,’’ ACM Trans. Web, vol. 1, no. 1, p. 6, May 2007.
[15] Z. Yilei, Z. Zibin, and M.R. Lyu, ‘‘WSPred: A Time-Aware Personalized QoS Prediction Framework for Web Services,’’ in Proc. IEEE 22nd ISSRE, 2011, pp. 210-219.

A.NANDHINI at Studying IIYear M.E In Computer Science and Engineering, Sapthagiri College of Engineering, Dharmapuri.

P.V.SANKAR GANESH M.E., Assistant Professor, Department of Computer Science and Engineering, Sapthagiri College of Engineering, Dharmapuri

.