Applying Software Wrapping on Performance Monitoring of Web Services

Liguo Yu
Computer Science and Informatics
Indiana University South Bend
South Bend, IN 46634, USA
ligyu@iusb.edu

ABSTRACT. Commercial, scientific, and social activities are increasingly becoming dependent on service-based web applications. Web-services are influenced by many uncontrollable factors. It is difficult to predict their performance only based on the pre-deployment testing. The performance characteristics, such as response time and failure rate, are volatile and therefore crucial in web service-based application. This paper states that it is important to continually monitor the performance of web-services during the process of their invocation on the client site. In this study, a wrapping-based approach to monitor the performance of web-services is presented and the results of a preliminary experiment that utilizes this approach are reported.

Keywords: Web service, performance monitoring, software wrapping.

(Received November 13, 2006 / Accepted January 3, 2007)

1. Introduction

Service-oriented applications refer to the software programs that use network services provided by third parties. They are emerging technologies useful for integrating enterprise applications and providing business solutions, such as electronic B2B and B2C. The most commonly used network service is web service that interacts with the clients over an interface described in WSDL and follows the SOAP standard [1]. The messages passed between the clients and the web services are in XML format and are transmitted using HTTP protocol.

Web service belongs to the component-based software development technology, in which, software products are no longer built from scratch; instead, they are built by integrating the existing pre-built components, commercial off-the-shelf (COTS) software, and legacy components [2].

The idea of web service-based application is to build an application by using web services provided by third-parties [3]. Figure 1 illustrates an example of web service-based application. A travel agent web application is built by integrating three third-party web services: air ticket booking service, car rental booking service, and hotel booking service. The travel agent application developers do not need to implement these complicated services themselves. Instead, they can focus on the user interaction and the internal logic of the application.

As with general COTS software, the web service must be tested before it can be integrated into the target application. This includes testing of both the functional requirement and the nonfunctional requirement. Usually, COTS software bought from a third party and integrated into a client application is managed by the client on the application site. Because the operation environment of COTS software is known to and controllable by the client, after it is tested before the deployment, its future behavior is largely predicable and manageable.

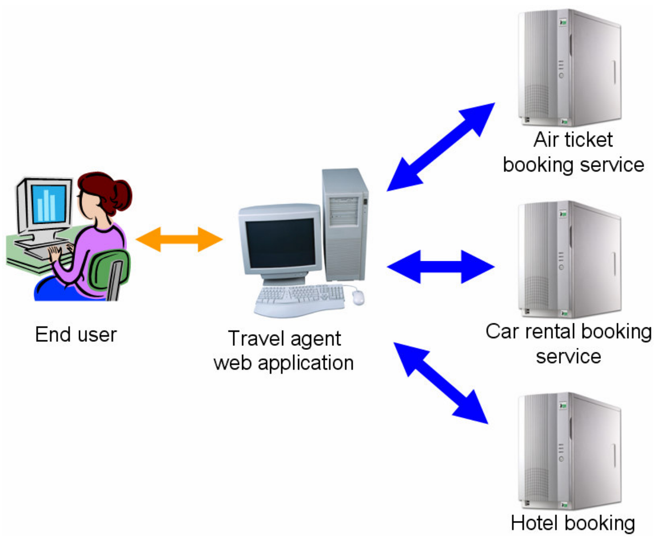

Figure 1: A service-based travel agent web application

However, web services are different from general COTS software. They are highly dynamic and interactive and accordingly pose new challenges to software quality assurance. Web services are managed by the third parties; the client can only buy the license to use it. Although the service could be tested by both the developer on the service site and by the buyer on the client site, its behavior is not guaranteed as it is tested. First, the web service is influenced by many factors that are not controllable or even aware of by the client. The examples of these factors are network traffics, host workloads, host running environment changes, and so on. Second, the clients are not guaranteed to know when the service is going to be updated, removed, or unavailable. Third, the errors detected for the services are reported to the service provider, the clients have no ideas about the reported errors and accordingly can not adjust their own request strategies.

The web service quality, especially nonfunctional requirement, such as performance, is an important issue in web service-based application. Web applications are easily to fail than conventional applications. A study showed that out of 41 sites managed by the U.S. Government, 28 sites contained web application failures [4].

Therefore, to predict and control the behavior of web service-based application, it is crucial for the client to continually observe and monitor the performance of imported web services. The monitoring results should be analyzed and provided to the client application administrator promptly to help the client decide whether the service has changed its behavior and whether these changes have effects on the client application. The clients can accordingly adjust their request strategies based on the monitoring results.

In this paper, a wrapping-based approach to monitoring the performance of web service is presented and the preliminary results of monitoring Google web service using this approach is analyzed and reported. The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes software wrapping. Section 4 describes the wrapping-based performance monitoring approach. Section 5 presents the experiment on Google web service. The conclusions and future research are presented and outlined in Section 6.

2. Related work

Performance is one of the most important nonfunctional requirements of software products. However, because service-based software development is an emerging new technology, there have been few reported performance assurance studies on web service-based applications. In this section, the work that is related to web service and performance assurance is briefly reviewed.

Fu et al. [5] used compile-time analysis techniques to perform the white-box coverage testing of exception handlers in Java web services. Their techniques applied the compiler-directed fault injection method. Offutt and Xu [6] presented an approach to testing web services based on data perturbation: data value perturbation, which modifies parameter values according to the data type, and interaction perturbation, which uses two types of communication mechanism, RPC communication and data communication.

Huang et al. [7] implemented *Waves*, a software tool for assessing web application security. The tool was based on a number of software-testing techniques including dynamic analysis, black-box testing, fault injection, and behavior monitoring. Liu et al. [8] proposed a web test model, which considers each web application component as an object and generates test cases based on data flow between those objects. Ricca and Tonella [9] presented a model based on the Unified Modeling Language (UML) to generate test cases and analyze web application evolution.

Weyuker and Vokolos [10] reported an industrial experience of testing the performance of a distributed telecommunication application at AT&T. They concluded that, given the lack of historical data on the usage of the target system, the architecture is the key to identify software processes and input parameters that influence the performance of distributed applications. Cai et al. [11] proposed a quality assurance model for Component-Based Software Development (CBSD). Using different quality prediction techniques, the model were applied to a number of programs to predict their quality.

Gorton and Liu [12] used a benchmark that includes the middleware infrastructure, the transaction and directory services, and the load balancing, to compare the performance of six different J2EE-based distributed applications. Avritzer et al. [13] compared the performance of different Object Request Broker (ORB) implementations that adhere to the CORBA Component Model. Liu et al. [14] investigated the suitability of light-weight test cases on distributed applications.

It worth noting here that the performance assurance testing described above are all performed before the deployment of the services. To our knowledge, there is no reported research on performance monitoring of web services after it is deployed.

3. Software wrapping

Software wrapping refers to a reengineering technique that surrounds a software component or system with a new software layer to hide the internal code and the logic of the component or system and to supply modern interfaces. One example of software wrapping is to reuse a legacy system in modern applications, where the unwanted complexity of the old system is hidden to the applications. Software wrapping removes the mismatch between the interfaces exported by a software artifact and the interfaces required by the current software program [15].

Software wrapping has been widely used to ensure the quality of third party software [16]. For example, Ghosh and Schmid [17] presented an approach and tool for assessing the robustness of COTS applications to failures from operating system functions or other third-party COTS software. Their approach consists of wrapping executable application software with an instrumentation layer that can captures, records, and perturbs the interactions between the target software and the third-party COTS software.

Guerra et al. [18] proposed an architectural solution to turning COTS components into idealized fault-tolerant COTS components by adding protective wrappers to them. Dean and Li [19] described a security wrapper technology that was implemented for COTS software products. The technology focuses on interchangeability for COTS software components, portability for the wrapper, and security for communications between applications via the wrapper.

In this research, software wrapping is applied on the performance monitoring of web services.

4. Wrapping-based web service performance monitoring

Figure 2 illustrates the proposed approach to monitoring the performance of web services. The software wrapping technique is utilized at the client site. The client interacts with the service through the wrapper. In this approach, basically, the wrapper provides two functions: (1) to customize the message exchanged between the client and the service, and (2) to monitor the performance of the service. Other functions could be added to the wrapper if needed, such as security checking, logging, and so on.

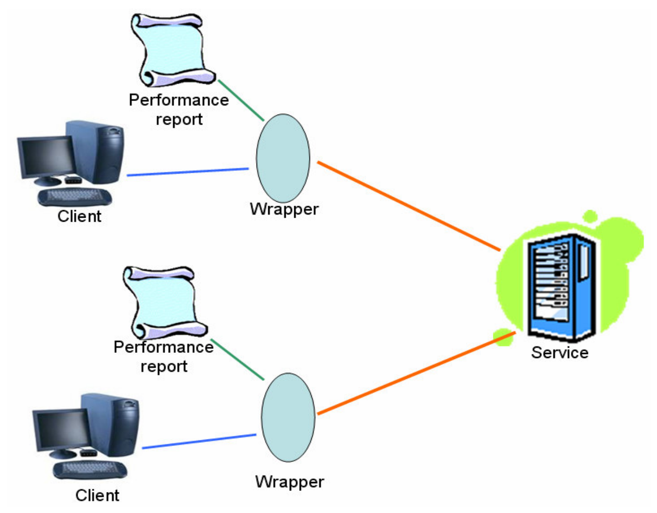

Figure 2: The wrapping of web service at client site

Figure 3 shows an example of a wrapper program that monitors the response time of the service. Besides the normal functionality, the specified performance parameter—response time—is recoded in the performance report. Because different clients are interested in different performance characteristics, in practice, the wrapper can be tailored according to the client's needs.

We remark here that the service wrapping is implemented at the client site, because in this paper, we are interested in the performance of web services from the client point of view. If the web service performance is considered by the service provider at the service site, different approaches should be used.

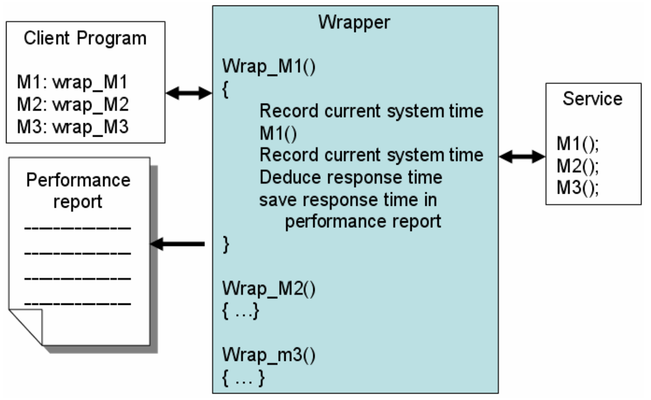

Figure 3: An example of wrapping a service

5. Experiment on Good web service

5.1. Experiment setting

Google is a tool for finding resources on the World Wide Web. It not only can enable the user to directly use the search engine via its web site, Google SOAP Search API service also allows software developers to query web pages directly from their own programs. In other words, Google search API is a web service that uses the SOAP and WSDL standards to allow other developers to import the service to their own applications under various environments, such as Java, Perl, or Visual Studio .NET.

As described before, to use a third party service, such as Google search API, it is important to observe and monitor its performance. In this experiment, we use C# under Visual studio .NET environment to implement an application that uses the Google search API. The application provides additional flexible search options, such as specifying the document type, the document language, and the last updated date. Moreover, the application is used as a benchmark to monitor the performance of Google search API from the client site.

Similar to other client site invocation, a software wrapper is implemented to parameterize the message that is sent between Google search API and the client application. More specifically, the wrapper is added with performance monitoring capabilities using the method shown in Figure 3.

Software performance can be characterized in several different ways. For instance, response time describes the delay between a request and the completion of an operation. Throughput denotes the number of operations that can be completed in a given period of time. Failure rate identifies the dependability of the service. The major factors that can affect the performance of a web service and the client can not control are network traffic conditions and server workloads. Therefore, in this experiment, we study response time and failure rate, which are direct representations of network traffic conditions and server workloads from the client point of view.

The experiment contains three steps. First, a list of 20 similar requests is defined. Next, the application is started and for every two seconds, it randomly selects one request from the request list and submits it to Google search API. If the client application successfully receives a response from the Google search API service, the response time is calculated otherwise, this is considered as a failure. Both the response time and the failure request are recorded in the performance report. Finally, the results are analyzed.

5.2. Experiment result

Figure 4 shows the response time of Google search API in a typical day. It does not represent its behavior in other days. However, it shows that the response time differs from request to request. The shortest response time is 0.02 seconds while the longest response time is 1.27 seconds, which differs about 60 times.

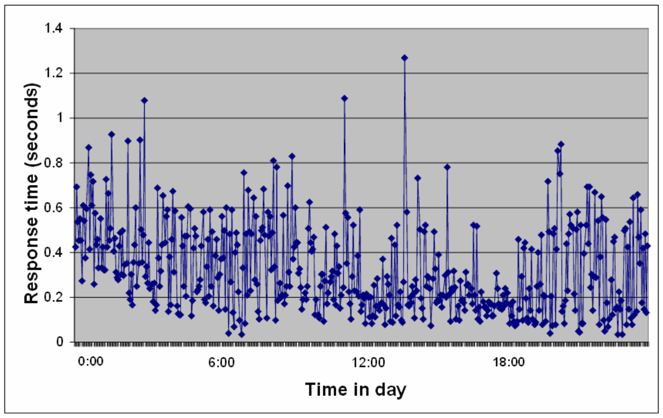

Figure 4: The response time of Google search API web service in a day

Figure 5 shows the failure rate of Google search API in a typical day. It is measured as the ratio of the number of failure requests over the total number of requests in one hour period. It shows that the failure rate in the morning (North American Eastern Time) is higher than that in the afternoon. It further suggests that it is better to run the application in the afternoon to avoid high failure rate. For example, at 6:00, the failure rate is about 60%; while at 18:00, the failure rate is about 0%.

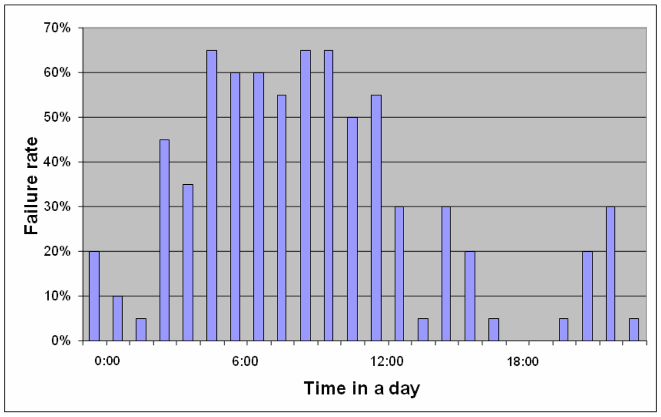

Figure 5: The failure rate of Google search API web service in a day

Figures 6 and Figure 7 show the boxplot of the average response time and failure rate of Google search API in different days of a week respectively. Both the response time and the failure rate are calculated in the unit of a day. The data is gathered in a four-week study period.

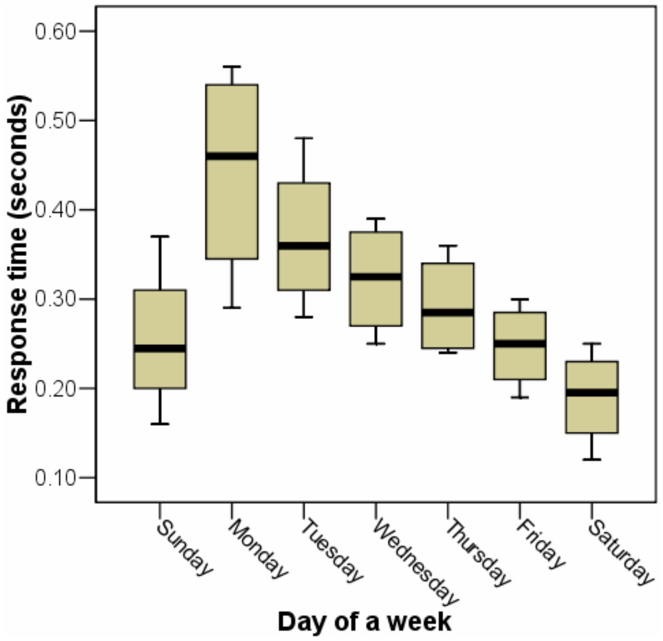

Figure 6: The average response time of Google search API web service in different days of a week

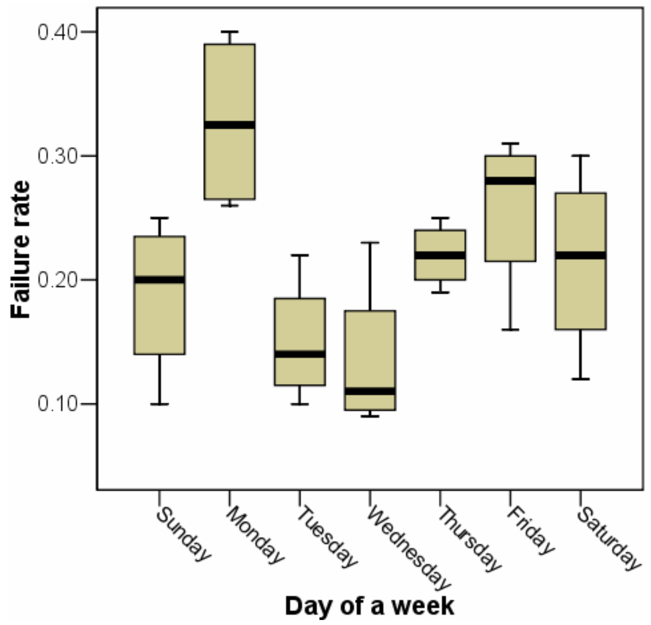

Figure 7: The average failure rate of Google search API web service in different days of a week

In a boxplot, the bold line within the box indicates the median. The box spans the central 50 percent of the data. The lines attached to the box denote the standard range. Figure 6 and Figure 7 show that on average, Monday and Tuesday have the two largest average response times and Monday and Friday have the two highest failure rates.

To study whether the response time and the failure rate are different in different days of a week statistically, we tested the following two null hypotheses:

- H_{01}: *There is no significant difference between the means of the response time of Google search API in different days of a week.*
- H_{02}: *There is no significant difference between the means of the failure rate of Google search API in different days of a week.*

To test these hypotheses, we performed two one-way ANOVA tests. The results are shown in Table 1. In both of the two tests, the *p*-values are less than 0.01. Therefore, we reject these null hypotheses and accept the corresponding alternative hypotheses. We conclude that there is significant difference between the means of the response time of Google search API in different days of a week; there is significant difference between the means of the failure rate of Google search API in different days of a week.

The experiment on Good search API web service indicates that its performance (response time and failure rate) differs hour to hour in a day and day to day in a week. Therefore, the performance of the client application is dependent on and sensitive to its subscribed web services. It is crucial for the client to monitor, analyze, and predict the performance of web services.

Table 1: The ANOVA test results

Hypothesis	DF	F value	*P*-value
H_{01}	6	4.710	0.003
H_{02}	6	4.444	0.005

6. Conclusions

Web service-based computing allows clients to dynamically bind services, and providers to modify the service implementation independently. It is becoming the solution for rapid and seamless integration of enterprise applications both and outside the enterprise boundaries. However, currently, one of the major concerns in web based application is the performance assurance.

Software performance assurance of web service has not been thoroughly investigated so far. However, with the widely use of web services in research and industry, this issue will become more and more critical. Our research tackles this problem and states that the performance of web service must be continually monitored after it is deployed; the pre-deployment testing is not enough and can not guarantee the future behavior of the web service.

In this paper, a software wrapping-based approach to monitoring the web service performance is presented. This approach can be easily implemented on the client site. The results of a preliminary experiment on Google search API web service using this approach indicated that the response time and the failure rate of web services differ from hour to hour in a day and day to day in a week. This findings support our statement of the importance of motoring the web service on the client site.

The long-term goal of this research project is to provide an automated software environment that can predict the future performance of web services based on the historical performance data gathered from the wrapper. More specifically, data mining techniques will be used to generate prediction rules and apply them on real time data.

Another future research is to combine the statistical approach with the modeling techniques to study web service performance. In this research, both the empirical study and the theoretical model analysis will be used to predict the performance of web services.

References

[1] Takahashi, K., Emmerich, W., Finkelstein, A., and Guerra, S. System development using application services over the Net, *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, p. 830, 2000.

[2] Emmerich, W. Distributed component technologies and their software engineering implications, *Proceedings of the 24th International Conference on Software Engineering*, Orlando, Florida, p. 537–546, 2002.

[3] Naik, V.K., Sivasubramanian, S., and Krishnan, S. Adaptive resource sharing in a web services environment, *Proceedings of the 5th ACM/IFIP/USENIX international Conference on Middleware*, Toronto, Canada, p 311–330, 2004.

[4] BIGSF. Government Web Application Integrity. The Business Internet Group of San Francisco, 2003.

[5] Fu, C., Ryder, B.G., Milanova, A., and Wonnacott, D., Testing of java web services for robustness, *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing*, p.23–34, 2004.

[6] Offutt, J. and Xu, W, Generating test cases for web services using data perturbation, *ACM SIGSOFT Software Engineering Notes*, 29(5), p. 1–10, 2004.

[7] Huang, Y., Huang, S., Lin, T., and Tsai, C. Web application security assessment by fault injection and behavior monitoring, *Proceedings of 12th International World Wide Web Conference*, Budapest, Hungary, p. 148–159, 2003.

[8] Liu, C., Kung, D., Hsia, P., and Hsu, C., Structural testing of web applications, *Proceedings of 11th International Symposium on Software Reliability Engineering*, p.84, 2000.

[9] Ricca, F., and Tonella, P. Analysis and testing of web applications, *Proceedings of the 23rd International Conference on Software Engineering*, Toronto, Canada, p. 25–34, 2001.

[10] Weyuker, E. and Vokolos, F. Experience with performance testing of software systems: issues, an approach, and case study, *IEEE Transactions on Software Engineering* 26(12), p.1147–1156, 2000.

[11] Cai, X., Lyu, M.R., and Wong, K, A generic environment for COTS testing and quality prediction, *Testing Commercial-off-the-shelf Components and Systems*, Sami Beydeda and Volker Gruhn (eds.), Springer-Verlag, Berlin, p.315–347, 2005.

[12] Gorton, I. and Liu, A. Software component quality assessment in practice: successes and practical impediments, *Proceedings of the 24th International Conference on Software Engineering*, Orlando, Florida, p.555–558, 2002.

[13] Lin, C., Avritzer, A., Weyuker, E., and Sai-Lai, L. Issues in interoperability and performance verification in a multi-ORB telecommunications environment, *Proceedings of the International Conference on Dependable Systems and Networks*, New York, NY, p. 567–575, 2000.

[14] Liu, Y., Gorton, I., Liu, A., Jiang, N., and Chen, S. Designing a test suite for empirically-based middleware performance prediction, *Proceedings of the 14th International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*, Sydney, Australia, p.123–130, 2002.

[15] Vigder, M.R. and Dean, J. An architectural approach to building systems from COTS software components, *Proceedings of 1997 Center for Advanced Studies Conference*, Toronto, Ontario, Canada, p. 131–143, 1997.

[16] Mingins, C. and Chan, C. Building trust in third-party components using component wrappers in the .net frameworks, *Proceedings of the 14th International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*, Sydney, Australia, p.153–157, 2002.

[17] Ghosh, A.K. and Schmid, M. An approach to testing COTS software for robustness to operating system exceptions and errors, *Proceedings 10th International Symposium on Software Reliability Engineering,* Boca Raton, Florida, p. 166, 1999.

[18] Asterio de C, P., Romanovsky, A., and de Lemos, R. Integrating COTS software components into dependable software architectures, *Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Hokaido, Japan, p. 139, 2003.

[19] Dean, J. and Li, L. Issues in developing security wrapper technology for COTS software products, *Proceedings of the 1st International Conference on COTS-Based Software Systems*, Orlando, Florida, p. 76–85, 2002.