Journal of Theoretical and Applied Information Technology
31st December 2011. Vol. 34 No.2
© 2005 - 2011 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EMPLOYING PERFORMANCE COUNTERS AND SOFTWARE WRAPPER FOR MEASURING QOS ATTRIBUTESOF WEB SERVICES

BAHAREH SADAT ARAB*, ABDUL AZIM ABD GHANI
Faculty of Computer Science and Info. Tech.,
University Putra Malaysia, 43400 UPM Serdang, Selangor D.E., Malaysia
E-mail: bahareh.arab@gmail.com , azim@fsktm.upm.edu.my

ABSTRACT

Web services have got popular for developing Service-Oriented Architectures recently. As several web services are available to execute the same function, Quality of Service (QoS) turns into a discriminative factor which is significantly considered in service selection and service composition approaches. In different approaches, monitoring of services is used for evaluating QoS attributes. Custom Windows Performance Counters (CWPC) is one of the approaches for monitoring performance of services at server-side. However, it has some limitations and it needs to access and change a service implementation which is not always possible in practice. In this paper, CWPC along with software wrapper is employed for measuring different QoS attributes such as response time, throughput and reliability in order to overcome current limitations. Additionally, it discusses how the proposed monitoring mechanism can be employed to optimize the service provider performance. The results show that the proposed monitoring approach is accurate in measuring QoS attributes.

Keywords: *SOA, Web Service; Monitoring; Quality Of Service; Qos Measurement; Performance Counter; Software Wrapper*

1. INTRODUCTION

Web services have got popular for developing Service-Oriented Architectures (SOA).Web services are located and invoked across the Web independent of platforms and programming languages. The current service oriented architecture contains three main roles: a service provider, a service consumer and the Universal Description, Discovery and Integration (UDDI) registry [9]. The service provider publishes web service description as well as detail information which are needed for invoking the service in the UDDI registry. The service consumer which refers to client user or client program can use the UDDI registry to discover a proper service which fulfills its requirements. Finally, the service consumer binds to the service provider to invoke its web service.

Quality of Service (QoS) is a combination of several quality attributes of a service that discriminate web services with same a functionality. It is a measure for how well a web service serves service consumers. QoS is a key factor for web service consumers to compare and select web services.

Windows Performance Counters (WPC) is one of monitoring approaches for measuring QoS attributes [6]. WPC provides predefined system counters that especially regards to the Windows Communication Foundation (WCF). However, WPC monitoring approach has some limitations and predefined system counters value do not map to QoS values properly and it can be employed just for WCF services. Custom Windows Performance Counters (CWPC) is used for monitoring of web services in order to overcome the current limitations of the WPC monitoring mechanism [1]. However, it needs to change the service code which may be impossible and sometimes there is not any authority to access the source code of each service and there is no guarantee that service providers will agree to change it. This paper has three main contributions as follows: Firstly, it proposes a web service architecture which employs CWPC along with software wrapper for measuring different QoS attributes. Secondly, it evaluates the accuracy of the proposed monitoring approach. Thirdly, it discusses how the proposed monitoring mechanism can be utilized to optimize the service provider performance.

The remainder of this paper is structured as follows. Section 2 presents related works for monitoring of web services. Section 3 describes the proposed architecture and Section 4 explains monitoring mechanism for measuring QoS of web services. Section 5 discusses about optimizing the service provider performance. Section6 reports the experiment results. Finally, Section 7 outlines the conclusions and the future work.

2. RELATED WORK

In this study, monitoring of web services is employed for measuring QoS attributes that are uncertain at invocation time and their value changes over time. Various researches describe the need for monitoring of web services [5], [7], [10], [12].They used different mechanisms for monitoring of services that are presented in this section.

Some of approaches are based on customizing and analyzing SOAP (Simple Object Access Protocol) massages [2], [4], [8], [11]. A Data Mart approach is presented for monitoring and evaluation of service provider performance [2]. In the proposed Web Services Log Architecture, SOAP intermediaries are used for implementing web services log. [4] describes an Automatic Web Services Testing Tool. The proposed automatic testing tool tracks and analysis extended SOAP messages. [8] presents automatic web service monitoring technique. In their approach, handlers are applied for processing and analyzing SOAP messages for measuring quality attributes of web services such as reliability, throughput and latency. [11] proposes a mechanism for automatic measurement of QoS attributes which set up on low level packet monitoring, proxy, and SOAP engine library modification. The low level packet monitoring is implemented by tracking SOAP packets. However, its implementation is hardware dependent. The proxy is a communication mediator which is responsible for measuring performance attributes. It is located between the service provider and the service consumer. As a result, the service consumer code should be configured and changed to use the proxy. In the SOAP engine library modification, the SOAP engine library should be modified for logging measured information which needs to distribute SOAP library modification on different implementations and platforms.

Some of approaches use software wrapper for monitoring of web services [3], [13]. In [3] the Parallel Performance Monitoring Service (PPMS) for monitoring the performance of media web services at runtime is proposed. The software wrapping technique is used for monitoring of response time of services at server-side. The wrapper custom messages exchanged between the consumer and the web service in order to calculate response time as a delay between service requests and the completion time of operation. [13] describes a wrapping-based monitoring at client-side. In their approach, software wrapping is applied for monitoring of web services during the service invocations. The wrapper is applied to evaluate a service for its response time. The main advantage of wrapping-based monitoring approaches is its easy implementation which can be custom based on consumer’s needs.

In [1], Custom Windows Performance Counters (CWPC) is defined for monitoring of web services. However, it needs to change the source code of each service. In this study, CWPC along with software wrapper is applied to overcome current limitations for measuring different type of QoS attributes such as response time, throughput and reliability. The proposed architecture and the monitoring mechanism are presented in next sections.

3. THE PROPOSED ARCHITECTURE

The proposed web service architecture enables monitoring and discovery of web services based on QoS requirements of the service consumer. Consequently, the traditional service-oriented architecture which composes the service provider, service consumer and UDDI registry is extended as shown in Figure 1.

In the proposed architecture, the service provider publishes their service information and QoS attributes via the Publish Manager. Web service information is published in the UDDI registry whereas QoS information is not supported by current UDDI registries. In this work, the QoSDB is used for storage of QoS values that were obtained by monitoring of web services. The monitoring is performed by the Monitoring Entity which applies performance counters. The result of monitoring would be stored as a log file. The performance counter log files should be gathered from the service provider for further analysis. The log information is transformed to XML format and scheduled for sending to the QoS Manager by the Log Convertor and Sender. The collected information from the service provider needs to be processed and then used to update QoS values in the QoSDB. The QoS Manager computes and updates values for each QoS attribute. In addition, it is responsible for sending notifications and reports to the service provider for its performance which is useful for determining the future trends such as taking QoS policy decisions. Ultimately, the service consumer sends its requirements to the Discovery Manager in order to find a service which meets its functional and QoS requirements. The Discovery Manager checks UDDI information and the QoSDB to discover a service which meets service consumer requirements and returns the results to the service consumer. Subsequently, the service consumer sends a service request to the service provider to invoke the selected service.

4. MONITORING MECHANISM

In this work, three most important QoS attributes such as reliability, response time and throughput are considered for monitoring of web services.

Response time: the required time for completing a service request. It is also related to the execution duration of a service.

Throughput: the number of service requests that a service provider can serve in a specified time interval.

Reliability: the ability of a service to execute its required functions under stated conditions for a specified time interval.

Performance counters along with software wrapper is utilized for monitoring and measuring the mentioned QoS attributes. The monitoring mechanism is presented in detail in next two subsections.

4.1. Performance Counters

CWPC must be defined to windows in order to measure QoS attributes of a web service at runtime.

The System Monitor utility provides set of predefined counters which can monitor system performance and track different processes in real time. A performance counter represents data for a particular component of the system or service. For our approach, performance counter framework is extended for custom counters and each service has a set of counters that track particular information for measuring its QoS attributes. Table 1 presents different performance counters and their types for evaluating related QoS criterion.

Table 1. Counters setting information

Counter Name	**Counter Type**	**QoS Criterion**
ResponseTime	AverageTimer32	Response time
Throughput	RateOfCounts PerSecond32	Throughput
SuccessfulExecution	NumberOfItems32	Reliability
FailedExecutions	NumberOfItems32	Reliability
TotalExecutions	NumberOfItems32	Reliability

Applications Performance log enables capturing counter data for later analyzing. Counter log files can be built on a regular schedule for automatic logging process. The log file contains measured data or counters values which would be used for calculating QoS values of a service as follows.

Response time presents the average required time that was taken to complete the service request at different successful invocation times.

$$\textbf{Response time} = \frac{\sum_{1}^{n} RT_i}{n} \qquad (1)$$

Where, RT_i is the historical *ResponseTime* counter data at the specific measurement time and n is the number of historical *ResponseTime* counter data for successful invocations.

Throughput is the average actual number of requests that the service provider served at different measurement times.

$$\textbf{Throughput} = \frac{\sum_{1}^{n} TP_i}{n} \qquad (2)$$

TP_i is the historical *Throughput* counter data at the particular measurement time and n is the number of historical *Throughput* counter data.

Reliability is the probability of a request is responded correctly. It is measured by considering the number of failures of a service in a time interval. It is also can be calculated as the ratio of successful executions and total executions during total measurement time.

Reliability =

$$1-\frac{\#FailedExecutions}{\#TotalExecutions}=\frac{\#SuccessfulExecution}{\#TotalExecutions} \quad (3)$$

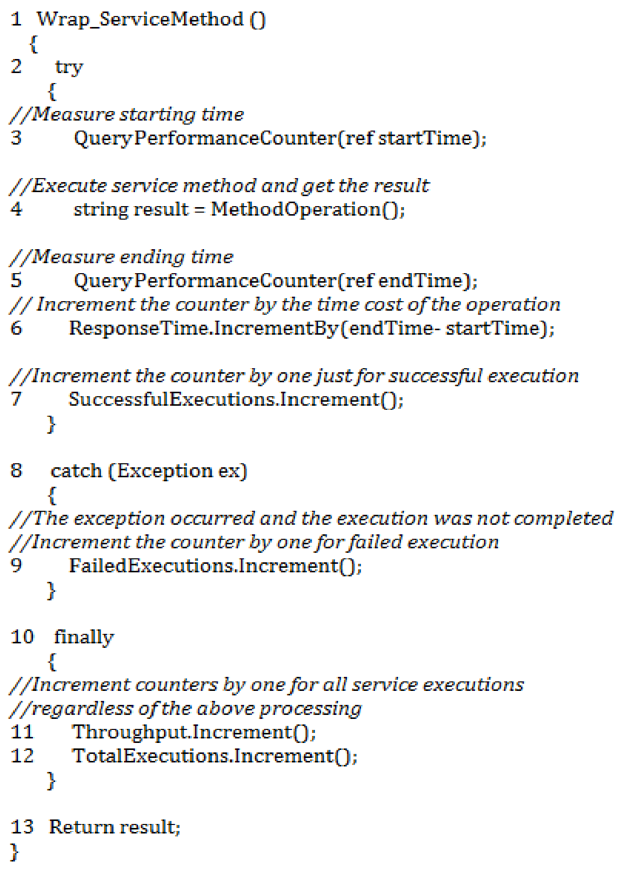
In this paper, three QoS attributes are considered in measurement process, although other QoS attributes can be calculated simply by further analyzing of the counters log file.

4.2. Software Wrapping

The benefit of using system performance counters is that, their value changes automatically whereas the value of CWPC should be set to change by an application.

There are two possible ways for setting counter's values. The first technique is to add increment methods in a service code directly which needs to access and change the web service implementation. The second technique is to apply software wrapping at server-side. The software wrapper can be applied in order to increment counters values and the service provider receives service requests through the wrapper. By contrast with the first technique, there is no need to change the service code and its implementation in the software wrapping technique. The software wrapping technique is used to wrap a web service with a supplementary software layer that hides the detail of service implementation and provides additional functions to adjust and increment counters values.

Figure 2 demonstrates a sample of employing software wrapper and custom performance counters for monitoring of a web service. The wrapper is applied between web service consumers and the web service for setting and incrementing of performance counters.



```
Wrap_ServiceMethod ()
 {
   try
    {
//Measure starting time
      QueryPerformanceCounter(ref startTime);

//Execute service method and get the result
      string result = MethodOperation();

//Measure ending time
      QueryPerformanceCounter(ref endTime);
// Increment the counter by the time cost of the operation
     ResponseTime.IncrementBy(endTime- startTime);

//Increment the counter by one just for successful execution
     SuccessfulExecutions.Increment();
    }

   catch (Exception ex)
    {
//The exception occurred and the execution was not completed
//Increment the counter by one for failed execution
     FailedExecutions.Increment();
    }

   finally
    {
//Increment counters by one for all service executions
//regardless of the above processing
      Throughput.Increment();
      TotalExecutions.Increment();
    }

  Return result;
}
```

Figure 3. Pseudo-code of counters setting

Figure 3 demonstrates the pseudo-code of a wrapper program and its descriptions. It shows the wrapper increments performance counters based on the QoS attribute definitions.

5. OPTIMIZING THE SERVICE PROVIDER PERFORMANCE

The proposed monitoring mechanism can be utilized to optimize and improve the service provider performance. One of the facilities that the system monitor provides is alerts. Alerts can be scheduled in monitoring process of web services to record an event or log other system performance counters when an especial event occurred. It is used for setting an action that will be performed when a specified counter reaches a given value. A threshold for a counter value is defined and the alert will trigger as the counter value exceeds or falls below the specified value so a cause of the change will be investigated by further analysis. Different actions can be set to perform when an alert triggers such as logging the event, sending a network message, starting performance data log or running an especial program. Alert is beneficial to recognize the system bottlenecks and it can be adjusted for performance counters by considering various scenarios that are important and crucial for the service provider. For instance, using alerts is helpful when a response time of a web service takes too long. In this case, there is a need for logging and checking other performance values of the system in order to find the reason of this event. Moreover, recorded events or collected data that was logged by the alert can be analyzed for generating reports. The analytical reports assist the service provider to determine the future trends and take management decisions for improving its performance.

Another important aspect that should be considered for server-side monitoring is the monitoring overhead. Monitoring impose overhead on the system as it consumes machine resources which degrade the service provider performance. Consequently, some factors should be considered in monitoring process in order to reduce the monitoring overhead on the system. One of the factors is decreasing the number of counters that are applied in monitoring process. For instance, there is no need to create a counter for counting the number of service execution failure if the reliability is calculated based on the number of successful service invocations divided by the total service invocations. Another significant factor is setting suitable monitoring interval. Choosing short monitoring interval leads to more data collection and more often sampling in compare of long monitoring interval. However, by reducing monitoring interval more machine resources would be consumed which may leads to degrade the service provider performance. Therefore, the monitoring interval should be adjusted properly for the system. The impacts of monitoring intervals on the service provider performance and QoS attributes will be discussed in the experimental results section.

6. EXPERIMENTAL RESULTS

The main purpose of the experiments is to assess and evaluate the proposed monitoring mechanisms. The implementation was done in a simulation environment. Simulations were performed in two phases, according to the goals of experiments. The next subsections explain the experiments, analyses and results.

6.1 Assessing the Accuracy of The Monitoring Approach

The goal of this phase of experiments is to evaluate the accuracy of the presented monitoring mechanism. To reach the goal, the accuracy of the proposed monitoring approach which employs performance counter along with software wrapper was compared with similar approaches in terms of response time, throughput and reliability. The accuracy of measuring response time was compared with applying software wrapper at server-side. Furthermore, the throughput and execution failure of a service were measured up to windows performance counters. In this experiment, a local WCF service was defined which can be monitored by the other compared monitoring approaches. In order to simulate a real service that is typically used, 1 second was set for the service method execution time and the probability of failure for service execution is 10%.A simulator program was run which used multi-threading to simulate different service consumers that sent service requests to the service at the same time. The service was monitored for 10,000 service requests with rate of 20 req/second the monitoring interval was set as 30 seconds.

Figure 4 presents the result of average response time by the CWPC monitoring and software wrapper in different monitoring intervals. The average response time for CWPC is 4.07 second and for the wrapper is 4.26 second. Additionally, the standard deviation for CWPC is 3 milliseconds whereas the standard deviation for software wrapper approach is 11 milliseconds. The result indicates that CWPC is more accurate for measuring response time of the service since the wrapper registers system time two times to compute and record response time to the performance report for each service request. Whereas, the average response time for different service requests computes automatically by the performance counters in CWPC approach and counter data was logged and recorded at each monitoring interval.

The result of Throughput for CWPC monitoring was evaluated with WPC monitoring which use predefined system counter as *CallsPerSecond* counter for measuring how often a service had been invoked.

Figure5provides the average throughput that was measured by both of monitoring approaches. The average measured values for CWPC is 17.86 while WPC is 17.66. The average of throughput by our approach is little bit higher however the results shows the average throughput or the number of service requests that can be processed in a second is approximately seventeen transactions per second by both monitoring approaches. The difference variation of values during monitoring time is because of the different incremental time of counters.

In WPC, *CallsFailedPerSecond* counter represent the number of calls that have unhandled exceptions, and are received by this service in a second. The counter can be used for counting unsuccessful service invocations. Also, *FailedExecution* counter is used in CWPC for counting failures of the service requests in order to compute the reliability of services.

Figure 6 illustrates the counters values in different time interval. As can be seen from the above figure, both of counters shows same values in different monitoring intervals which was expected. The reliability of the service can be measured by considering the number of service failures. As a result, the reliability of monitored service is 91% which is confirmed by both approaches. The results of CWPC monitoring in terms of throughput and failed executions are similar to WPC however the proposed CWPC can be used for monitoring different type of services as simple web services, XML web services or WCF services whereas the WPC can be applied just for WCF services.

6.2 Assessing the Performance Impact of Monitoring

This phase of simulations has the goal of evaluating the impact of CWPC monitoring on the service provider performance and provided QoS. For our approach, a local web service was defined. 1 second was set for its method execution and the probability of failure for web service's execution is 10% to simulate a real web service that is typically used. Three different service request rates as high, intermediate and low (high=20 req/sec, intermediate= 10 req/sec and low=5 req/sec) were adopted in this experiment. The web service was monitored for three request rates by considering different monitoring intervals (5, 15, 30, 45 and 60 seconds). The logging was performed for about 10,000 service requests.

Figure7 presents that average response time of the web service increases for higher number of service requests. As a result, high volume of service requests degrades the service provider performance. The result confirms the importance of load balancing which balances the workload among the similar services from different service providers. Additionally, it is suggested to prevent selecting a service provider in its peak time mode. The result expresses that the response time decreases for the high service requests rate when the monitoring interval is extended. Monitoring at server-side consumes system resources so longer monitoring interval leads to use less machine resources and the monitoring has lower overhead on the system. However, low monitoring interval does not impact the performance of the service for low service requests rate. As shown in the figure, the average response time values for the 10 request rate became steady for 30 seconds and higher monitoring intervals and it indicates monitoring does not have significant overhead in normal working state of the service provider.

Figure 8 shows throughput which is the number of completed requests per second. Throughput values significantly increase for high request rate when the monitoring interval is extended. As monitoring degrades the service provider performance, longer monitoring interval has less overhead and more services could be served at per unit of time.

The service is considered to perform well when its throughput is high and it has a faster response time. In low request rate, monitoring has not notable overhead on the service provider and values of response time and throughput do not change for different monitoring intervals. Accordingly, minimum monitoring interval is desirable for normal working time of the system as gathered information is more accurate when measurement interval is cut down. In high request rate, the monitoring overhead significantly impact QoS values. Consequently, long monitoring interval for high volume of service requests and peak working time mode of the system is recommended (in this case longer than 60 second).In intermediate request rate, average QoS values likely become steady after 30 second monitoring interval so longer than 30 second monitoring interval is preferable in this case. As a result, a compromise must be found between the performance of a service provider and freshness of measured data.

According to Figure 9, average reliability values slightly decrease by increasing of monitoring intervals. Reliability is related to the number of failed service executions. The failure probability of service invocations seems to exceed in extended time duration.

Results show that some nondeterministic QoS values depend on the measurement period. It indicates the importance of choosing proper monitoring interval. The effect of monitoring is more significant on QoS attributes such as response time and throughput.

7. CONCLUSION AND FUTURE WORK

The monitoring mechanism which is based on applying custom windows performance counters and software wrapper for measuring QoS attributes was described. The results indicate that the proposed monitoring mechanism is an accurate monitoring approach which supports different type of services. The monitoring mechanism can be utilized to improve the service provider performance. Additionally, the findings indicates that adjusting suitable monitoring interval is a critical factor for reducing monitoring overhead and improving performance of the service.

The work described in this paper can be extended for automatically predicting QoS based on the pattern of historical QoS values which measured by monitoring of services. Furthermore, more QoS attributes will be considered for monitoring and prediction process for future research.

REFRENCES:

[1] Arab BS, Abdul Ghani, A.A., "Performance Counter Monitoring Mechanism for Measuring QoS Attributes in SOA", *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 8, no. 2, 2010, pp.56-62.

[2] da Cruz SMS, Campos LM, Campos MLM, Pires PF., "A data mart approach for monitoring Web services usage and evaluating quality of services", *In Proceeding of the Twenty-Eighth Brazilian Symposium on Databases*, 2003, pp. 267-80.

[3] Kalavathy GM, Seethalakshmi P., "Parallel Performance Monitoring Service for Dynamically Composed Media Web Services", *Journal of Computer Science*, vol. 5, no. 7, 2009, pp.487-92.

[4] Li Y, Li M, Yu J., "Web Services Testing, the Methodology, and the Implementation of the Automation-Testing Tool", *Lecture Notes in Computer Science*, 2004, pp.940-947.

[5] Maximilien, E. M., & Singh, M. P., "A framework and ontology for dynamic web services selection", *Journal of IEEE Internet Computing*, 2004, pp.84-93.

[6] Michlmayr A, Rosenberg F, Leitner P, Dustdar S., "Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection", *In Proceeding of the International Middleware Conference*, USA, 2009, pp. 1-6.

[7] O'Brien L, Merson P, Bass L., "Quality attributes for service-oriented architectures", *In Proceeding of the International Workshop on Systems Development in SOA Environments IEEE Computer Society*,2007, pp. 20-26.

[8] Raimondi F, Emmerich W., "Efficient online monitoring of web-service SLAs", *In Proceeding of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering ACM New York*, NY, USA, 2008, pp. 170-80.

[9] Ran S., "A model for web services discovery with QoS", *Journal of ACM SIGecom Exchanges*, vol. 4, no. 1, 2003, pp.1-10.

[10] Saxena N, Goel A., "A Probe-based Observability Mechanism for Monitoring of Web Services" *Journal of Recent Trends in Engineering*, vol. 1, no.1, 2009, pp. 600-602.

[11] Thio N, Karunasekera S., "Automatic measurement of a QoS metric for Web service recommendation", *In Proceeding of the Australian Software Engineering Conference, Australia*, 2005,pp. 202-211.

[12] Yeom G, Tsai WT, Bai X, Min D., "Design of a Contract-Based Web Services QoS Management System", *In Proceeding of the 29th IEEE International Conference on Distributed Computing Systems Workshops*, Montreal, Québec, Canada, 2009,pp. 306-311.

[13] Yu L., "Applying software wrapping on performance monitoring of web services", *INFOCOMP Journal of Compute Science*, vol. 6, no. 3, 2007, pp. 1-6.

LIST OF IMAGES WHICH NEED TO BE PRINTED IN COLOR

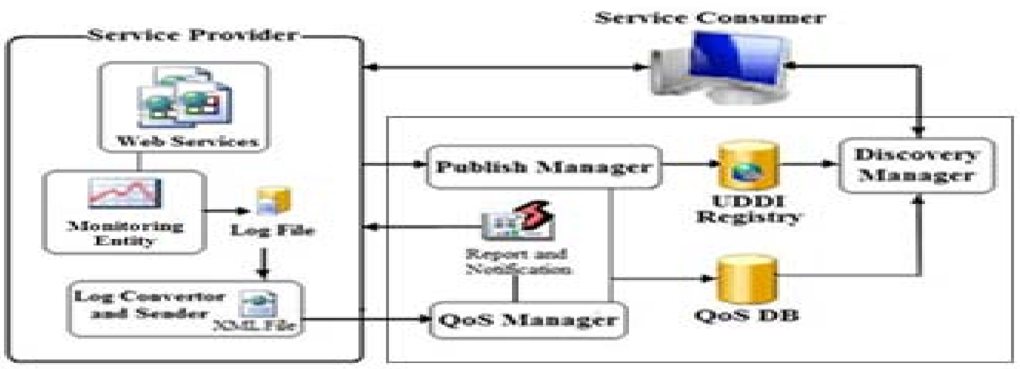


Figure 1. The proposed web service architecture

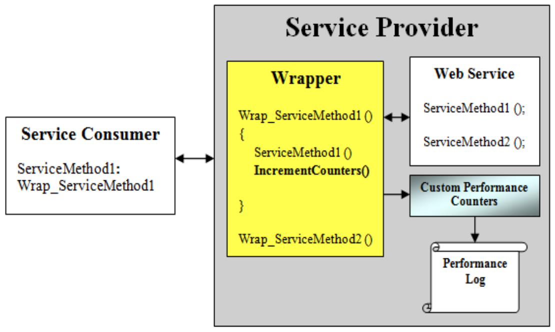


Figure 2. An example of using wrapper

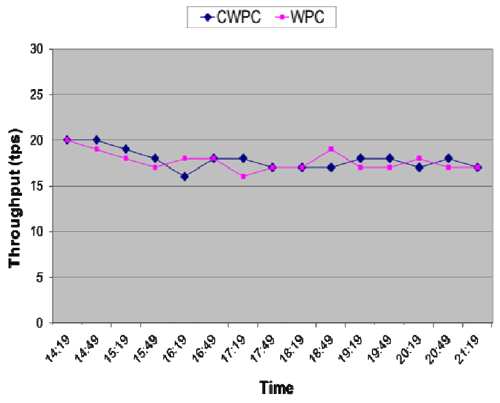


Figure 5: Average Throughput

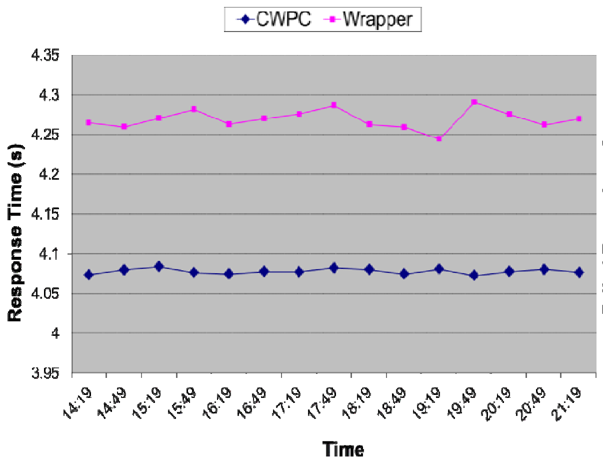


Figure 4: Average Response Time

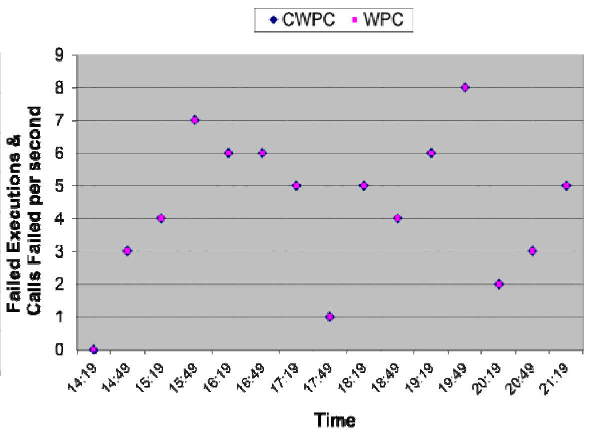


Figure 6: Failed Executions/Calls Failed Per Second

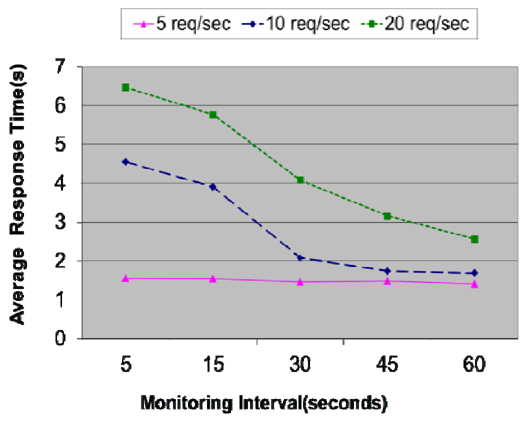


Figure 7: Average response time

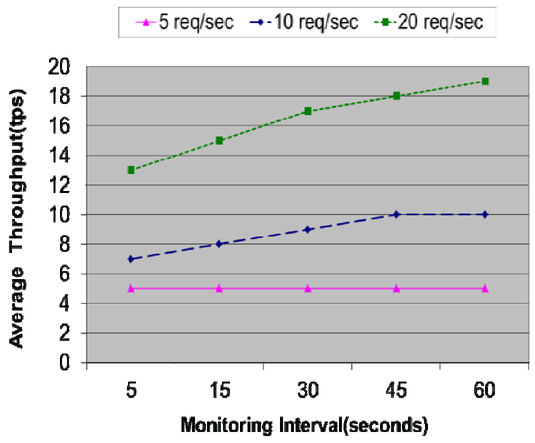


Figure 8: Average throughput

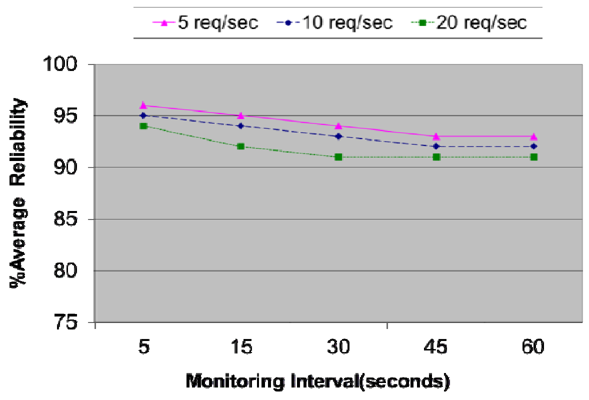


Figure 9: Average Reliability