A Mechanism for Collecting and Feedbacking the Real-time Quality of Web Service

Cheng Zhou
School of Software
Shanghai Jiao Tong University
Shanghai P.R.China
zhoucheng32@gmail.com

Haopeng Chen
School of Software
Shanghai Jiao Tong University
Shanghai P.R.China
chen-hp@sjtu.edu.cn

Quality of Service (QoS) is an important factor during service composition and replacement. Since static QoS data cannot reflect the real-time performance of Web Service (WS), a mechanism is needed to collect the dynamic QoS data. Also, QoS query result is determined by the QoS data stored by service registry. Consequently, it is also necessary to enable a mechanism to feedback those dynamic QoS data to the service registry from both consumer-side and provider-side.

In this paper, we propose three methods to collect the real-time QoS data, namely AOP-based, Proxy-based, and Port-based. The application of the three methods will be discussed by comparing their advantages and disadvantages. Since the feedback conforms to the OWL-Q language, which is a rich and extensible modular ontology language, programs can understand the semantic feedback automatically by parsing the feedback messages.

QoS feedback; WS monitor; OWL-Q; AOP intercept

I. INTRODUCTION

In this paper, we first introduce several methods to get QoS feedback. Then, we investigate the performance issue, and put forward a way to improve feedback's efficiency.

Current WS standards, such as WSDL and UDDI, mainly focus on the aspect of functionality rather than the description of QoS. The feedback also needs a universal description language, which can be understood by machine.

OWL-S[3] is an ontology, within the OWL-based framework of the Semantic Web, for describing Semantic Web Services. It will enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints. However, it does not describe any QoS concept.

OWL-Q[1] is a rich, extensible and modular ontology language that complements the WS functional description language OWL-S.(OWL-Q is a language and ontology for stating conjunctive queries against OWL data. It provides ontology of queries.) In addition, the author of OWL-Q have extended the most prominent CSP-based QoS-based WS discovery approach. OWL-Q is an OWL-S extension (syntactical separation), for QoS-based WS description of both requests and offers. OWL-S ontological description is extended for two reasons: to comply with Semantic WS description standards (standards compliance) and to use the OWL ontology formalism (extensible and formal semantic QoS model).

If we want to evaluate a certain Web service, the easiest way is to test it, and record its performance using statistical methods. SoapUI[4], which is a useful tool for WS development. For example, the average response time of a request can be asserted to not going under a specified value for a longer period of time. Other assertions available including max-response time, TPS, etc. All of these tests are executed in local environment. We suppose each invoke can be feed back its quality to a control center so that others can query all kinds of services by their functional requirements and quality of services.

We leverage AOP-based, proxy-based and package-block-based ways to monitor the WSs. All QoS information in OWL-Q is sent to the service registry. QoS information in OWL-Q will be much easier to be understood and processed. Thus, service consumers can select the WS using both functional requirement and non-functional requirement much easier.

II. RELATED WORK

The QoS maintenance mechanism involves two different aspects: QoS monitoring and QoS aware registry mechanism.

A. Other QoS monitoring mechanism

However, previous research has failed to consider about both consumer-side and provider-side situation. Some paper introduces an automatic measurement of a QoS metric for web service, but not all the metric can monitor from provider. [2] For example, response time stands for the amount of time between sending the request and receiving a response or the guaranteed average time required to complete a service request. Also referred to as execution duration, it is computed using the processing time and the transmission time [8]. So provider side will never know how long it takes this request message transferring on network. As we see, it is necessary to set up a monitor in client.

This paper is supported by the National High-Tech Research Development Program of China (863 program) under Grant No. 2007AA01Z139.

B. A registry set up to provide QoS based search

Most of the existing service registries either do not consider QoS or provide insufficient QoS support. Nevertheless, we have proposed a P2P service registry extension named QMC to enhance their ability to manage and query QoS. Similar to other P2P system, this system scales well in terms of number of peers, search efficiency, and number of users. We were leveraging OWL-Q to model different QoS metrics flexibility. Hence, we need a feedback source to support this type of service registry.

C. Extended WSDL for QoS Attribute

Some of my mates introduce a lightweight WSDL extension for the description of QoS characteristics of a web service, such as performance, reliability, availability, security, etc. The proposed extension is defined by first introducing the WSDL metamodel, derived from the WSDL XML Schema, and then transforming it into a QoS-enabled WSDL (Q-WSDL) metamodel, from which the Q-WSDL XML Schema is derived. Indeed, an XML Schema defines the WSDL language in the same respect as a metamodel is used to define. Representing the WSDL grammar in terms of a metamodel allows something to enhance its comprehensibility and facilitate its extension.

III. REAL-TIME QOS DATA COLLECTION

The QoS data are collected from both the provider-side and the consumer-side. Different operating systems, programming languages and transfer protocols may affect the way of collecting QoS data. Therefore, in this section, we propose three ways of implementing a QoS interceptor, which can be used according to different environments. The first one involves AOP method to record the performance of each message. The second one relates to building an http package filter to reassemble the incoming request messages and outgoing response messages. The third one is based on a proxy settled between client and service.

A. AOP-based Way

Aspect-oriented programming (AOP) is a programming paradigm that increases modularity by enabling improved separation of concerns [6].

AOP is the natural choice of separating cross-cutting issues from core business logic. We proposed a QoS language to separate QoS aspects from application by AOP approach. Dynamical weaving is also extensively explored in AOP literature to build reconfigurable systems. We employ AOP to ensure providers to monitor and report their service reports obligatorily and accurately by separating authoritative monitoring aspects from susceptible service providers.

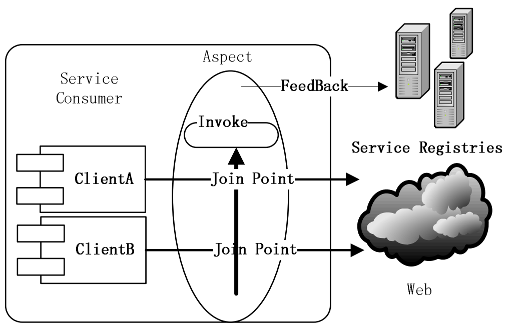


Figure 1. Aspect component--service register interactions

In this method, we analyze how a JAX-WS complied framework invokes a service, Take JAX-WS RI as an example. Programmers need to declare the types of parameters when invocating service functions. Then the web services framework can help them to send request and return the result. There must be one method that the web service framework used to send the request of service consumer to service provider and get return message. So if we intercept this method, we would get the feedback information, and add them to QoS collocation.

The primary difficulty of monitoring service invoking by AOP method is to determine which method to intercept. The request will go through a dozen layers of the stack, from initial declaration to output finally. If we choose a low level method to intercept (By low level, we mean the method is a super class, which will be extended by its subclasses), it can lose generality, and bring more errors to QoS attribute computing. On the other hand, if we intercept too deeply, it blocks a special function belong to a certain framework of web service. So it makes the service consumer depend on that framework. Ff JAX-WS standard has already defined the method to send requests in last step, every kind of WS frameworks, which follows the JAX-WS standards, would be monitoring the same invoke method in consumer-side.

Unfortunately, JAX-WS standards have no such definition of those interfaces. It only contains upper interfaces and interactive mode between service consumer and service provider. Secondly we choose JAX-WS reference implementation framework, JAX-WS RI the block for experiment [9].

Following up the course of sending call request, we found the final method is 'process' in class 'com.sun.xml.ws.client.Stub'. So an aspect 'FeedbackAspect' is defined, which including pointcut for ' process' method and two advices: before, after.

Two advices are used for running additional code before and after invoking 'process' method.

This aspect will create a new Feedback instance before 'process' execution and invoke relevant method in relevant instance to do computing before calling service. Then, another method, which will calculate the QoS feedback information and send to service registry, will execute before response reach. It should be noted that, method "calculateBeforeServiceInvocation" and "calculateAfterServiceInvocation" of class-"Feedback" are cycle called by the method with the same name, in each FeedbackItem instance. Meanwhile, each attribute of QoS is calculated. The whole collocation action and feedback process will be shown as following figure.

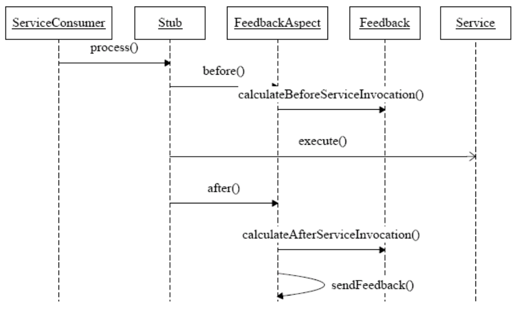


Figure 2. Sequence Diagram of Collecting and Sending Feedback

In the process, the main task is to select which kind of methods to send feedback to service registry. In order to receive feedback from service consumer, a service registry must provide certain interface for invoker to call automatically. By all appearances, it is best to implement this interface as another web service. By this way, it breaks dependence on programming language and other technical detail, in service registry's view.

B. Http-package-based Method

If we want a third part system separated from service and client application scope, An AOP way may not meet requirement. Without any effect to both sides, we need a more abstract method to monitor the web service invoking.

There are tools available to do low level packet monitoring such as libpcap and winpcap utility, which is widely used for network traffic monitoring and packet filtering. The main idea of this approach is to capture SOAP incoming and outgoing packets, exchanged between a client and the server by installing a monitor or sniffer application to capture the HTTP traffic between the two points. The main advantage of this approach is that the monitoring program can be completely independent from client code, and as a result can be done without modifications to the client code. However, this approach has some limitations.

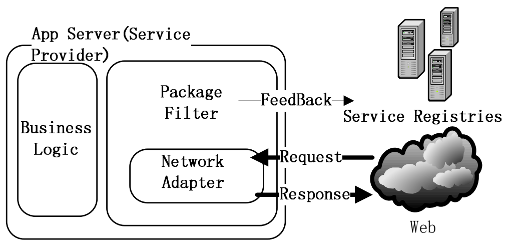


Figure 3. Package block component--service register interactions

Firstly, the libpcap utility that captures the low level packets is hardware dependent. Therefore, the monitoring program needs to be built to recognize a lot of hardware related protocols (such as Ethernet, PPPoE, PPPoA, and so on). This means more CPU resources are needed to decode, filter and reconstruct the raw packets. Secondly, if the SOAP messages are encrypted or compressed, the monitoring program will be ineffective since it needs to (due to the cost to) decrypt and decompress.

Network protocols often need to transport large chunks of data, e.g. when transferring a file. The bottom layer protocol might not be able to handle that chunk size (e.g. limitation of the network packet size), or is stream-based like TCP, which doesn't know data chunks at all.

In that case the network protocol has to handle the chunk boundaries itself and (if required) spread the data over multiple packets. It obviously also needs a mechanism to determine the chunk boundaries on the receiving side.

It is the same when we get SOAP Message over network. One SOAP request and response will be divided into many Tcp packages. Meanwhile HTTP/GET method and HTTP/POST method request will be divided. So the packet reassembling will be very important when we monitor the web services invoking events over low level protocol.

Here is the solution.

- Two caches are initialized before we start. One is used for storage of tcp request which including an http header in its tcp data segment. The other records each request after meeting its first response package.
- Each coming package is examined. If its data segment contain any string match the http request header format, such as, this package is considered to be the first part of invoke. Using its acknowledged number as index, its data is put into the first cache.
- If acknowledged number of coming package is already existed in first cache, The data of this package is add after the existed one.
- If coming package's serial number equals one of the indexes in the first cache, it is considered to be response message. Then, the request in the first cache is removed, and is added into the second cache with response message, using response package's acknowledged number as index as index.
- If coming package's acknowledged number is equals an existed one in the second cache. Its tcp data segment is added to the response message of that index.

Finally, all packages in the first cache will be the request message without response coming yet. All the messages in the second cache will be the whole request message with the whole response or partly of it. Briefly speaking, the reorganization of TCP package is only concerned about the serial number, response number, and data.

If we implement TCP reassembly strictly, each data has to be confirmed (Acknowledged), before pop up. However, in the packet data processing is similar this as long as the Serial Number directly deals with continuous data, rather than cached.

This is based on the following considerations:

- It needs to be cached while waiting for confirmation. The majority of TCP packets are continuous without requirement to sort. Direct treatment can greatly improve efficiency.
- Data from later response will certainly be confirmed. Only when TCP transformation interrupted, the confirmation would lose.
- Ignore the one way packet data flow analysis. Without that case we need a large number of package caches, because no confirmation number can determine to pop-up package from cache.

Proxy-based Method

In this approach, a proxy is used as a communication mediator between the client and the provider [5]. Messages sent between the client and the providers are visible to the proxy, and the performance attributes will be measured by the proxy.

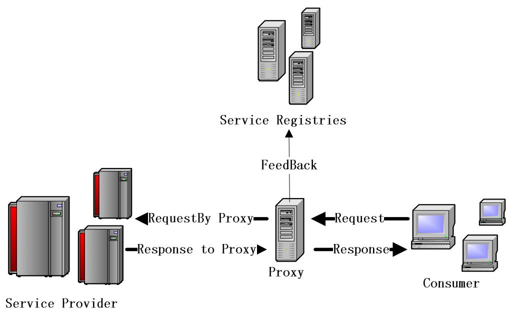


Figure 4. Proxy component--service register interactions

Every connection will be received and sent forward by proxy process. A thread gets all request messages, while another will start a socket connection to for deliver the request body to the real destination. After the response reaches, proxy will replay original invoker with response message. Each communication between service provider and service consumer will be visible to the proxy. It is the easiest way to monitor web service without considering the performance cost.

C. *Summary*

It should be noted that this study has only propose (provide those optional) different approaches to monitor quality. No one is considered to be the best or the worst in this paper.

TABLE I. TABLE TYPE STYLES

Item	*AOP*	*Package block*	*Proxy*
high degree of accuracy	Y	N	Y
easy to monitor	Y	N	Y
loosely-coupled with WS development language	N	Y	Y
distributed calculate	Y	Y	N
loosely-coupled with WS Operation system platform	Y	N	Y
less cup and other resource	Y	N	N
taken			

IV. RUNTIME STATUS PROCESSING

A. *Data initialization (Feedback Preparation)*

After we collected enough information about runtime status, it's time to send them to services registries. Neither, the feedback frequency is high or low will increase the load or reduce the nicety of query result in registries. On the one hand, if registry records every invoking by feedback unit, it takes much resource to compute statistical result from original data. On the other hand, if remote client only communicates with registry monthly or weekly, the latest situation can't reflect the query result at once. Hence, we design a strategy to path this problem.

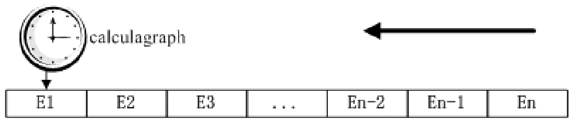


Figure 5. Structure of feedback queue

Result of each feedback will be:

$$QoS_{\text{attribute}} = \frac{\sum_{i=1}^{n} Q_i}{L_{queue}} \tag{1}$$

Q_i represents the element in the queue. L_{queue} represents the length of the queue.

Feedback frequency depends on the invoking times in unit time. Also, there is a maximum distance between two feedbacks, if certain service is hardly invoked recently. We suppose a data structure to store all QoS elements of each invoking. It's a circular queue. This is equivalent to the requirement that whenever an element is added, all elements that were added before have to be removed before the new element can be removed. All elements in the queue will affect the performance of the service. In this way, single feedback is a statistical result based on all data in queue. Consequently, the feedback frequency is to determine like this:

$$f_{feedback} = k\frac{f_{invoke}}{l_{queue}} \tag{2}$$

$f_{feedback}$ represents the frequency of feedback to service registry. f_{invoke} represents the frequency of the web service is invoked. l_{queue} represents the length of local QoS queue. k represents a constant for regulating.

There are two probabilities that the feedback of single invoking is removed from the queue. The first one is this element meets the end of the queue, and is removed because of the incoming of new element. The other condition is, this element has stayed in the queue for long enough to lose statistical value. For example, let's consider an example one service hasn't been invoked for weeks because of network error or others. Some elements in the queue are recorded weeks ago.

they have no relationship with performance behaviors of the service at present. They all need to be removed. So we need a counter, and give all elements in the queue a count value. We suppose the initial count value is set to “n”. Count value of all elements will be decreased by 1, in every “T” hours. Each element in queue will be removed no matter it is in the end of queue or not if its count value become zero.

B. Data Structure

As we describe in beginning of this paper, using OWL-Q as our feedback message format is batter for computer to understand QoS information and easier for query. For example, if we organize it into a simple format xml document, using some kinds of tag according to existed QoS attributes, it actually can work as well. Unfortunately, we may have hard code in program, in another meaning, that is, we need know every tag, while decoding the xml document using xpath. In this way, the number of QoS attributes is fixed, unextensibly.

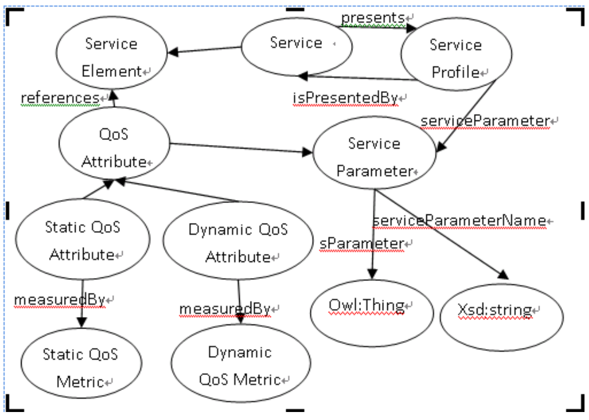


Figure 6. Main facet of OWL-Q[10]

serviceParameters[2] - An expandable list of properties that may accompany a profile description.

ServiceParameter - describes service parameters. In general we can think this class as the root of an ontology of Service Parameters of different types. Other types of ServiceParameters may expand this definition by adding other properties.

ServiceAttribute[2] - a subclass of OWL-S ServiceParameter and references a ServiceElement.A Service Attribute contains two subclasses: QoSAttribute is a subclass of the general class Attribute. An attribute can be separated into a) physical or service attributes, b) measurable or immeasurable attributes and c) unique or derived attributes.

Physical attributes [1] like Time, Temperature and Location characterize environmental (contextual) factors of a WS or its requester while service attributes like Availability or NumOf Interfaces are functional or non-functional characteristics of a WS. Measurable attributes like Time are measured by specific metrics while immeasurable attributes like manageability cannot be measured. Unique attributes like Time are not derived by other attributes and are measured by resource metrics while derived attributes like Throughput are produced by complex metrics computed by functions using metrics of other attributes. The Domain class represents the domain of knowledge that a service applies to and is separated into two sub classes[2]:

a)GeneralDomain and b) Specific Domain. The GeneralDomain stands for every possible WS. Specific Domain can be further specialized/subsumed.

Supposing we follow some way to get service performance data as list below:

Here we monitor 2 different operations from different web services, using single thread.

GetWeather-
http://www.h2bbs.com/Weather/Weather.asmx?WSDL

GetVersion- http://localhost:8080/axis2/services/Version?wsdl

TABLE II. TABLE TYPE STYLES

Operation	*Mi n*	*Max*	*Avg*	*Cnt*	*Tps*	*Bytes*	*err*
GetWeather	52	1029	134. 46	73	7.43	125122	0
getVersion	4	16	5.32	62	187. 87	20026	0

min: the shortest time the step has taken;

max: the longest time the step has taken;

avg: the average time for the test step;

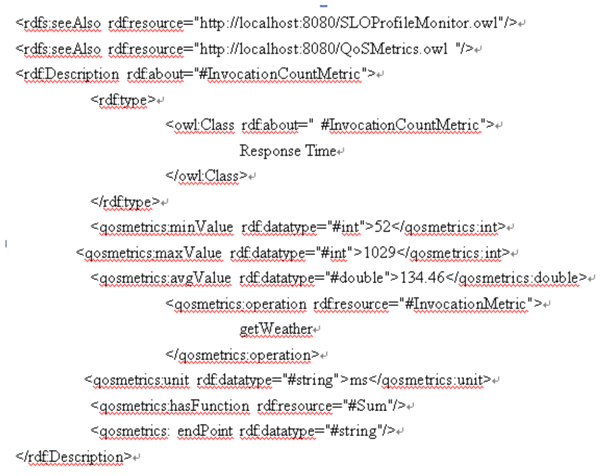
cnt: the number of times the test step has been executed;

tps: the number of transactions per second for the test step, see Calculation of TPS/BPS below;

bytes: the number of bytes processed by the test step;

err: the number of assertion errors for the test step[3].

My feedback of QoS metrics message to service registry will be as following script. We only take one attribute for example:



```
<rdfs:seeAlso rdfresource="http://localhost:8080/SLOProfileMonitor.owl"/>
<rdfs:seeAlso rdfresource="http://localhost:8080/QoSMetrics.owl "/>
<rdfDescription rdfabout="#InvocationCountMetric">
          <rdftype>
                    <owl:Class rdfabout=" #InvocationCountMetric">
                              Response Time
                    </owl:Class>
          </rdftype>
          <qosmetrics:minValue rdfdatatype="#int">52</qosmetrics:int>
        <qosmetrics:maxValue rdfdatatype="#int">1029</qosmetrics:int>
          <qosmetrics:avgValue rdfdatatype="#double">134.46</qosmetrics:double>
                    <qosmetrics:operation rdfresource="#InvocationMetric">
                              getWeather
                    </qosmetrics:operation>
         <qosmetrics:unit rdfdatatype="#string">ms</qosmetrics:unit>
          <qosmetrics:hasFunction rdfresource="#Sum"/>
          <qosmetrics: endPoint rdfdatatype="#string"/>
</rdfDescription>
```

Figure 7. Feedback Message

V. CONCLUSION AND FUTURE WORK

As the QoS from both the provider and consumer has an important role for the client to choose a provider, a mechanism to maintain the QoS measure of the service invocations is needed. Throughout this paper we have presented the mechanism of fetch runtime QoS attributes. Three different monitoring methods are introduced. For some attributes are meaningless to get from provider, such as response time, the client want to know exactly performance in certain network location. While, some other such as throughput can only be get from provider. So it is necessary to design those method based on different mechanism, just make sure automatically monitoring QoS attributes will feedback well and truly. The rest of QoS attributes such as security and transaction support needs to be measured manually, may not include in this paper's scope.

We aim suppose to build a plug-in for IDE, such as eclipse. So, it will be much easier to find out QoS metric for each service, when we construct our application. If there are a set of optional services to select, this plug-in help you to find out which service is more suitable, for your network environment and personal preference.

Furthermore we want to define more QoS metrics. At this moment we only get serial ones such as response time and package size. More metrics perfect the attribute of each service, based on which the service resister will give a much exacter search result.

REFERENCES

[1] Kritikos Kyriakos, Plexousakis Dimitris "A semantic QoS-based web service discovery algorithm for over-constrained demands" NWeSP 2007 3rd International Conference on Next Generation Web Services Practices, p 49-54, 2007

[2] Niko Thio and Shanika Karunasekera "Automatic Measurement of a QoS Metric for Web Service Recommendation" the 2005 Australian Software Engineering Conference (ASWEC'05)

[3] OWL-QoS-ontology http://www3.ntu.edu.sg/home5/PG04878518/OWLQoSOntology.html 2009-04-20

[4] http://www.soapui.org/ 2009-03-25

[5] SiMing Li, Chi-Hung Chi, Chen Ding, Shuo Chen, Ying Huang, Automatic Recommendation of Quality Requirements for Software Services, 2007 IEEE International Conference on e-Business Engineering

[6] Research on SOA-based applications based on AOP and web services. Proceedings of the 2008 International Conference on Computer and Electrical Engineering, ICCEE 2008, p 753-757, 2008

[7] Restoration and Audit of Internet E-mail Based on TCP Stream Reassembling. International Conference on Communication Technology Proceedings, ICCT, v 1, p 368-371, 2003

[8] Sravanthi Kalepu, Shonali Krishnaswamy, Seng Wai Loke, Verity: A QoS Metric for Selecting Web Services and Providers. The Fourth International Conference on Web Information Systems Engineering Workshops

[9] Guang Yang, Hao-peng Chen, An Extensible Computing Model for Reputation Evaluation Based on Objective and Automatic Feedbacks, Proceedings - ALPIT 2008, 7th International Conference on Advanced Language Processing and Web Information Technology

[10] Kyriakos Kritikos and Dimitris Plexousakis OWL-Q for Semantic QoS-based Web Service Description and Discovery. First International Joint Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web