# An Objective and Automatic Feedback Model for QoS Evaluation

Cheng Zhou
School of Software
Shanghai Jiao Tong University
Shanghai P.R.China
86-13764475246
zhoucheng32@gmail.com

Haopeng Chen
School of Software
Shanghai Jiao Tong University
Shanghai P.R.China
86- 021-34204124
chen-hp@sjtu.edu.cn

## ABSTRACT

Quality of Service (QoS) is an important factor during service composition and recommendation. 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 use an efficient and accurate feedback method to calculate and send those dynamic QoS data to the service registry.

In this paper, we propose a QoS feedback model based on objective QoS metrics using some simple statistical theories and a dynamic queue as a data pool to cache all runtime status. Moreover, error determination and sampling feedback have been taken into consideration so that service provider assigns less hardware resource and avoids disturbing feedback result from unfriendly exception. By carrying out experiments, it demonstrates that this feedback model evaluates the WS performance better than other common methods. This model provides QoS metrics that are easy to rank and sensitive to the status change.

## Keywords

QoS Feedback, Web Service discover

## 1. INTRODUCTION

One key issue in the web service discovery and composition area is to estimate each service of all visible providers' nodes, whether their services meet the functional requirements of consumers. Equally important, consumers also would like to know which best meet their non-functional requirement, among these optional providers. Quality of services (QoS) is the majority part of these non-functional requirements, i.e., performance, reliability, availability, security, transaction, etc.

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

During the first selection, searching in service registry by semantic technique returns lots of choices. QoS-based selection ranks these services and makes recommendation according to their QoS attributes. Service registry collects these attributes from feedback of both providers and consumers. Consequently, feedbacks become determinant of ranking.

In this paper we present a model for improving feedback efficiency and accuracy. Before sending out feedbacks from original data, they are processed. So our model guarantee these feedbacks with distortions and the disturb elements are eliminated. That means single feedback message stands for a statistical result of long term or short term runtime status collection. It discriminates between impulsive noises and normal error behaviors. And also cheat in invocations is detected and excluded from calculation resource, in order to improve their performance expression.

## 2. RELATED WORK

Feedback mechanism involves three aspects: QoS metrics definition, data collection and calculation.

### 2.1 A. QoS metrics Classic Definition

A wide spectrum of metrics which attribute to quality of service has been put forth by the research community with often varying interpretations. Presented here is some metrics with classic definitions where applicable. We primary focus on these metrics listed below in this paper.

Throughput: Throughput is the number of web service requests served in a given period of time [1]. The response time of a system increases as the throughput increases and an important policy decision is to make a compromise between maximizing the throughput and minimizing the response time [2].

Response time: The amount of time between sending the request and receiving a response [2] or the guaranteed average time required to complete a service request [1]. Also referred to as execution duration, it is computed using the processing time and the transmission time.

### 2.2 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[11] to model different QoS metrics flexibility. Hence, we need a feedback source to support this type of service registry.

## 2.3 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, our previous research has presented three ways of implementing a QoS interceptor, which can be used according to different environments. The first one involves AOP method [8]: setting an aspect to record the performance of each invocation before and after message-sending-out action executed. 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, monitoring all requests and response going through.

## 2.4 Other QoS feedback mechanism

However, previous research has failed to consider about precomputing before feedback. Most statistical jobs are executed when original data received in service registry [6]. It brings more computing pressure and takes up many other hardware resources of service registry. Meanwhile, it is not so easy for designers to determine how many history records should be recorded in registry. As our implementation always provides transient performance status, evaluation depends on average performance value of several invocations recently. On one hand, if we use a fixed formula to handle this, it works reasonable only when invocation frequency, average message size, network lendcy and etc are all the same for different service provider. It sounds a little unpractical. On the other hand, if we never add historic data while calculation, evaluation of service fluctuates according to the feedback up-to-date. Unstable behave makes selection and sorting more difficult and incorrect.

## 3. QOS METRICS CALCULATION

In this paper we only take some performance attributes to introduce the feedback model.

Metrics for performance are Service Response Time (SRT) and Throughput of a Service (TP (SRV)). SRT is an elapsed time between the end of a request to a service and the beginning of the service's response. Here, the service indicates both the atomic and composite service.

SRT = Time when Service Consumer finishes sending request to the service – Time when Service Consumer starts receiving response from the service[3] (1)

SRT includes transmission time, XML parsing time, and actual processing time. The range of SRT is SRT >0, where the lower value indicates higher response time. TP (SRV) represents the number of requests served at a given period of time.

TP (SRV) =Number of Completed Service Requests / Unit of Time [3] (2)

The numerator is the number of successfully completed requests to a service exposed by WSDL. The service can indicate both the atomic and composite service. The denominator is a unit of time such as second, minute, or hour. The value range of the metric is TP (SRV) > 0. Maximum number of requests that can be processed indicates how many users can be processed concurrently in a service.

## 4. DYNAMIC FEEDBACK QUEUE

### 4.1 Queue Construction

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 dynamic feedback queue to path this problem.

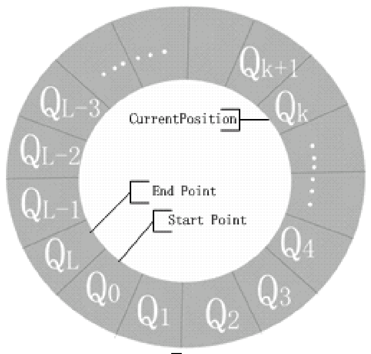

**Figure 1. Queue structure**

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

Feedback frequency depends on the number of invocation in unit time. Also, there is a maximum distance between two feedbacks, if certain service is rarely invoked recently. We suppose a data structure to store all QoS elements of each invocation. It's a circular queue or ring. A ring showing, conceptually, a circular buffer. This visually shows that the buffer has no real end and it can loop around the buffer [9]. All elements in the buffer 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:

$$f_{feedback} = k \frac{f_{invoke}}{l_{queue}} \tag{3}$$

$f_{feedback}$ 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. Service registry wants feedback represent how the service works now. Outdated elements may stand for historical status, they are including in feedback before, but not this time. 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" minutes. Each element in queue will be removed no matter it is in the end of queue or not if its count value become zero.

## 4.2 Feedback Result Calculation

Now we have a queue full of runtime status data. The following step is getting a value of single QoS metric.

Firstly, we need to eliminate noises in the queue. When we invoke the service first time, the local DNS cache happens to miss the destination address. While the request or response message transferring in the network, some IO exception occur in Tcp package. They all bring some extraordinary element into queue (extraordinary element means these performances will not stand for the real status of service, they are considered to be noises). The final evaluation value will not including these noises, because they don't stand for the real performance status of the service provider. At the same time, we have to different noises from bad performance; make sure the queue is sensitive to QoS changes.

We suppose $D_{k\sim k+1}$ stands for distance of $E_k$ and $E_{k-1}$ in certain QoS dimensions. $D_N$ Stands for the longest distance between normal elements. If both $D_{k\sim k+1}$ and $D_{k-1\sim k}$ bigger then $D_N$, we consider $E_k$ is noisy node.

Otherwise, some other element exists nearby this element. it probably stands for bad performance.

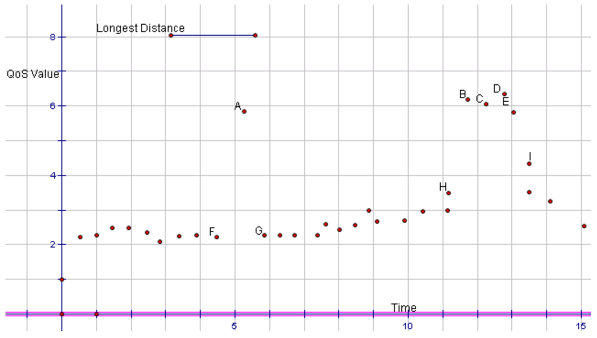

**Figure 2. Performance Example**

$D_{A\sim F} > D_N$ and $D_{A\sim G} > D_N$, so element A is noise.

$D_{B\sim C} < D_N$, $D_{C\sim D} < D_N$, $D_{D\sim E} < D_N$, so elements B, C, D, E are effective value.

Second, using following formula, we can get feedback values.

$$Q_{response} = \frac{\sum_{i=1}^{n} l_i Q_i}{\sum_{i=1}^{n} l_i} \tag{4}$$

Take response time for example, every element's life value will be added together to make a weighted average. That means element stays in the queue longer, the less effective it makes to the evaluation result.

## 4.3 Sampling Method

The model introduced before, goes through every invocation of the service. When the server is quite busy, it brings process load and affects the normal function of itself. Consequently, we use sampling approach to solve it. In this case, not all the invocation performance is added into feedback queue.

The sampling theorem describes two processes in signal processing [4]: a sampling process, in which a continuous time signal is converted to a discrete time signal, and a reconstruction process, in which the original continuous signal is recovered from the discrete time signal. The continuous signal varies over time (or space in a digitized image, or another independent variable in some other application) and the sampling process is performed by measuring the continuous signal's value every T units of time (or space) [4], which is called the sampling interval. In practice, for signals that are a function of time, the sampling interval is typically quite small, on the order of milliseconds, microseconds, or less. This results in a sequence of numbers, called samples, to represent the original signal. Each sample value is associated with the instant in time when it was measured.

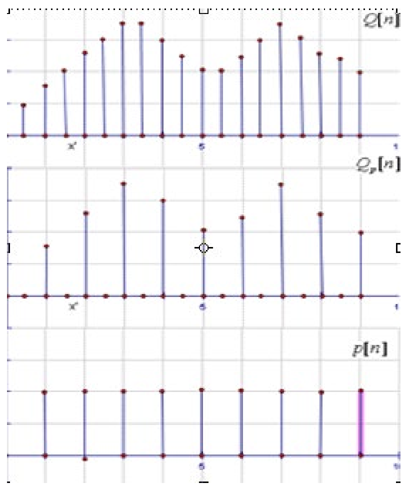

**Figure 3. Discrete-time sampling**

The original sequence Q[n] at integer multiples of the sampling periods N and is zero at the intermediate samples [5], that is,

$$Q_p[n]\begin{cases} Q[n] & if \ n = an \ \text{int} eger \ multiple \ of \ N \\ 0 & otherwise \end{cases} \quad (5)$$

The effect in the frequency domain of discrete-time sampling is seen by using the modulation property[5].

$$Q_p[n] = Q[n]p[n] = \sum_{k=-\infty}^{+\infty} Q[kN]\delta[n-kN] \quad (6)$$

We have in the frequency domain that

$$Q_p(\Omega) = \frac{1}{2\pi}\int_{2\pi} P(\theta)Q(\Omega-\theta)d\theta \quad (7)$$

The Fourier transform of the sampling sequence p[n] is[5]

$$p(\Omega) = \frac{2\pi}{N}\sum_{k=-\infty}^{+\infty}\delta(\Omega - k\Omega_s) \quad (8)$$

Where Ωs, the sampling frequency, is 2π/N. Combining 2 formulas before, we have

$$Q_p(\Omega) = \frac{1}{N}\sum_{k=0}^{N-1} Q(\Omega - k\Omega_s) \quad (9)$$

Now, we can get original performance metrics recovered from the sampling signal.

It's still a problem: what determines the sampling frequency. We can't tell 1/2 of invocation frequency batter than its 1/4. The Nyquist–Shannon sampling theorem states that perfect reconstruction of a signal is possible when the sampling frequency is greater than twice the maximum frequency of the signal being sampled,[4] or equivalently, when the Nyquist frequency (half the sample rate) exceeds the highest frequency of the signal being sampled. If lower sampling rates are used, the original signal's information may not be completely recoverable from the sampled signal.

We suppose the performance status Q(t) follows linear time-invariant system (LTI) theory. Linearity means that the relationship between the input and the output of the system is a linear map [4]: If input x1(t), produces response y1(t), and input x2(t), produces response y2(t), then the scaled and summed input a1 x1(t)+ a2 x2(t). Time invariance [4] means that whether we apply an input to the system now or *T* seconds from now, the output will be identical except for a time delay of the *T* seconds. That is, if the output due to input *x*(*t*) is *y*(*t*), then the output due to input *x*(*t* − *T*) is *y*(*t* − *T*). The reason why the performance status is time-invariant is quite clear. These performance statuses only depend on the network traffic situation and how many request the server received. Both of two impact factor is independent of time. We suppose service performance is linear because concurrent operation in computer over limited numbers of CPU is serial in fact. Consequently, if response one request needs T milliseconds, two responses cost 2T milliseconds probably in my treatment.

$$Q(n) = \frac{1}{2\pi}\int_{2\pi} Q(e^{j\omega})e^{j\omega n}d\omega \quad (10)$$

$$Q(e^{j\omega}) = \sum_{n=-\infty}^{+\infty} Q[n]e^{-j\omega n} \quad (11)$$

$Q[n]$ is a sequence of all frequency elements. $Q(e^{j\omega})$ is the frequency spectrum of Q. It seems sampling frequency has nothing to do with invocation frequency. According to Nyquist–Shannon sampling theorem[5], the sufficient condition for exact reconstruct ability from samples at a uniform sampling rate $f_s$ (in samples per unit time) is: $f_s \geq 2B$, 2Bis called the Nyquist rate[5].B is the bandwidth, the biggest frequency of all base signals. $f_s / 2$ is called the Nyquist frequency and is a property of this sampling system.

Here is an example:

$$Q[n] = \cos\omega_0 n = \frac{1}{2}e^{j\omega_0 n} + \frac{1}{2}e^{-j\omega_0 n}, \omega_0 = \frac{2}{5}\pi \quad (12)$$

$$Q(e^{j\omega}) = \pi\delta(\omega - \frac{2\pi}{5}) + \pi\delta(\omega + \frac{2\pi}{5}), -\pi \leq \omega < \pi \quad (13)$$

We suppose the feedback attributes express like this:

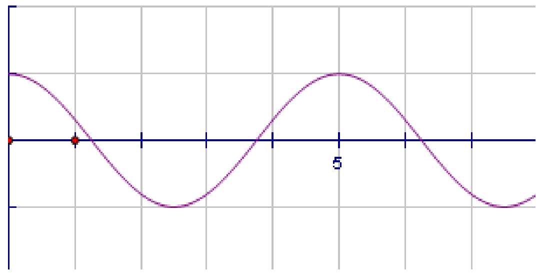

**Figure 4. Hypothetical Performance Example**

The sampling frequency needs to be more than '2/5'.

## 5. CASE EXAMPLE

In this section we apply the feedback model to process a list of runtime performance data, comparing with two different methods. All ways introduced here works on QoS data after sampling. It shows practicability and usefulness of the model introduced in this paper. We suppose a web service processing request represent as Figure 5. And we invoke the same operation of this service with a random delay.

Figure 5 shows original performance data without any process. Each point in the diagram stands for a feedback value. We take response time for example. Most response time is near 200 milliseconds.

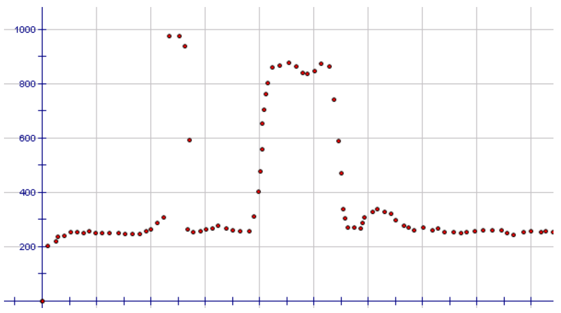

**Figure 5. Runtime QoS value**

Figure 6 shows original performance data have been processed by certain sampling frequency. The fourth point is considered to be invocation noise. And three big response time elements follow after.

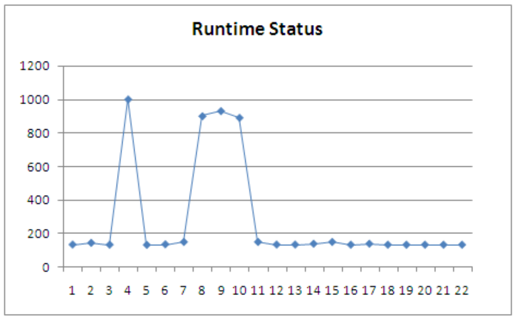

**Figure 6. Runtime QoS value after sampling**

Firstly, we using a method to feedback every invocation performance. Here we get result from service registry as Figure 7.Sampling helps to reduce pressure by 1/ $f_s$ , the more service performance smoothly, $f_s$ will be larger.

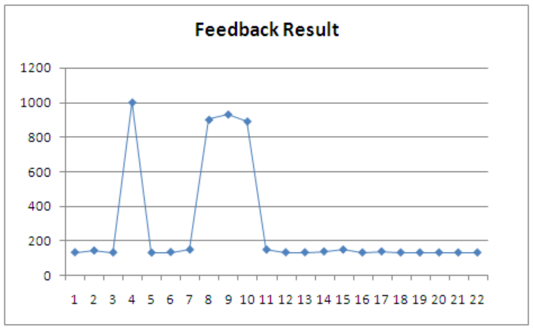

**Figure 7. Feedback result of every invocation**

Secondly, we add element together, and get their average value.

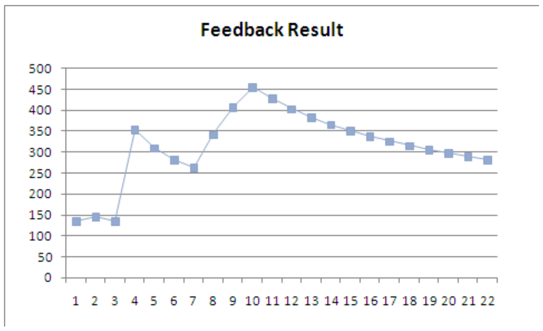

**Figure 8. Average of all invocation**

Finally, feedback using model introduced in this paper with sampling. Length of queue is 5.We can see from Figure 9 noise has nothing to do with the result. Change of original data effect result smoothly. When the data element out of the queue, it never effect any more.

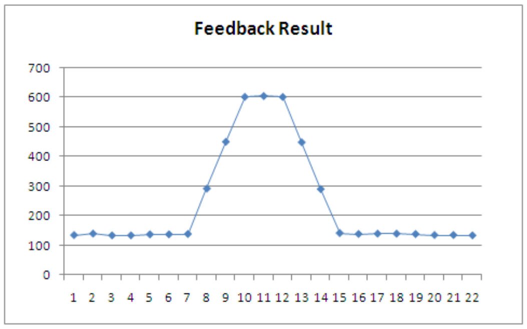

**Figure 9. Average value of invocation in queue**

## 6. CONCLUSION AND FUTURE WORK

Most of the existing ways for evaluation WS are all based on WS feedback [7]. Basically those feedbacks bring service registry lots of statistical job, because they are all original data stand for instantaneous status. In order to solve the problem, we proposed a computing model based on objective QoS metrics using some simple statistical theories and a dynamic queue used as a data pool to store all runtime status. Moreover, sampling feedback has been taken into consideration so that service provider with high invocation frequency will assign less hardware resource.

This paper is supposed to be a research on automatic feedbacks for service registry. The computing model proposed here needs improvement as well as extension in order to support other measurable objective QoS attributes and subjective attributes, such as user satisfaction, reputation. Invocation fraud also needs further study, which helps to avoid disturbing feedback result from unfriendly invocation. Besides, feedbacks to service registry is a multidimensional value. WS selection over multitude attributes will be studied in future.

## 7. REFERENCES

[1] Shuping Ran, “A Model for Web Services Discovery with QoS”, ACM SIGecom Exchange, Volume 4 Issue 1(March 2003).

[2] Gregor V. Bochmann, Brigitte Kerherve, Hanan Lutfiyya, Mohammed-Vall M. Salem and Haiwei Ye, “Introducing QoS to Electronic Commerce Applications”, Springer-Verlag Berlin Heidelberg 2001 pp. 138-147.

[3] Si Won Choi, Jin Sun Her, and Soo Dong Kim, “QoS Metrics for Evaluating Services from the Perspective of Service Providers”, 2007 IEEE International Conference on e-Business Engineering

[4] http://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem (2009-7-20)

[5] Alan V.Oppenheim; Alan S.Willsky; with S.Hamid, “Signals and Systems (2nd Edition), Pearson Education”, p250-p257,p470-p476.

[6] Niko Thio and Shanika Karunasekera, Automatic Measurement of a QoS Metric for Web Service Recommendation, the 2005 Australian Software Engineering Conference

[7] 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.

[8] Zhang, Jingjun; Meng, Fanxin; Liu, Guangyuan ,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

[9] http://en.wikipedia.org/wiki/Ring_Buffer (2009-7-23)..

[10] 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