Understanding quality of service for Web services

Improving the performance of your Web services

Anbazhagan Mani (manbazha@in.ibm.com), Software engineer, IBM Software Labs, India
Arun Nagarajan (anagaraj@in.ibm.com), Software engineer, IBM Global Services India

Summary: With the widespread proliferation of Web services, quality of service (QoS) will become a significant factor in distinguishing the success of service providers. QoS determines the service usability and utility, both of which influence the popularity of the service. In this article, we look at the various Web service QoS requirements, bottlenecks affecting performance of Web services, approaches of providing service quality, transactional services, and a simple method of measuring response time of your Web services using the service proxy.

Date: 01 Jan 2002
Level: Introductory

The dynamic e-business vision calls for a seamless integration of business processes, applications, and Web services over the Internet. Delivering QoS on the Internet is a critical and significant challenge because of its dynamic and unpredictable nature. Applications with very different characteristics and requirements compete for scarce network resources. Changes in traffic patterns, denial-of-service attacks and the effects of infrastructure failures, low performance of Web protocols, and security issues over the Web create a need for Internet QoS standards. Often, unresolved QoS issues cause critical transactional applications to suffer from unacceptable levels of performance degradation.

With standards like SOAP, UDDI, and WSDL being adopted by all major Web service players, a whole range of Web services -- covering the financial services, high-tech, and media and entertainment -- are being currently developed. As most of the Web services are going to need to establish and adhere to standards, QoS will become an important selling and differentiating point of these services.

QoS covers a whole range of techniques that match the needs of service requestors with those of the service provider's based on the network resources available. By QoS, we refer to non-functional properties of Web services such as performance, reliability, availability, and security.

Web service QoS requirements

The major requirements for supporting QoS in Web services are as follows:

- **Availability**: Availability is the quality aspect of whether the Web service is present or ready for immediate use. Availability represents the probability that a service is available. Larger values represent that the service is always ready to use while smaller values indicate unpredictability of whether the service will be available at a particular time. Also associated with availability is time-to-repair (TTR). *TTR* represents the time it takes to repair a service that has failed. Ideally smaller values of TTR are desirable.
- **Accessibility**: Accessibility is the quality aspect of a service that represents the degree it is capable of serving a Web service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time. There could be situations when a Web service is available but not accessible. High accessibility of Web services can be achieved by building highly scalable systems. *Scalability* refers to the ability to consistently serve the requests despite variations in the volume of requests.
- **Integrity**: Integrity is the quality aspect of how the Web service maintains the correctness of the interaction in respect to the source. Proper execution of Web service transactions will provide the correctness of interaction. A *transaction* refers to a sequence of activities to be treated as a single unit of work. All the activities have to be completed to make the transaction successful. When a transaction does not complete, all the changes made are rolled back.

- **Performance**: Performance is the quality aspect of Web service, which is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performance of a Web service. *Throughput* represents the number of Web service requests served at a given time period. *Latency* is the round-trip time between sending a request and receiving the response.
- **Reliability**: Reliability is the quality aspect of a Web service that represents the degree of being capable of maintaining the service and service quality. The number of failures per month or year represents a measure of reliability of a Web service. In another sense, reliability refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers.
- **Regulatory**: Regulatory is the quality aspect of the Web service in conformance with the rules, the law, compliance with standards, and the established service level agreement. Web services use a lot of standards such as SOAP, UDDI, and WSDL. Strict adherence to correct versions of standards (for example, SOAP version 1.2) by service providers is necessary for proper invocation of Web services by service requestors.
- **Security**: Security is the quality aspect of the Web service of providing confidentiality and non-repudiation by authenticating the parties involved, encrypting messages, and providing access control. Security has added importance because Web service invocation occurs over the public Internet. The service provider can have different approaches and levels of providing security depending on the service requestor.

QoS enabled Web services

The interface definition (WSDL) specifies the syntactic signature for a service but does not specify any semantics or non-functional aspects. QoS enabled Web services require a separate QoS language for Web services to answer the following questions:

- What's the expected latency?
- What's the acceptable round-trip time?

A programmer needs to be able to understand the QoS characteristics of the Web services while developing applications that invoke Web services.

Ideally, a QoS enabled Web services platform should be capable of supporting a multitude of different types of applications :

1. With different QoS requirements
2. By making use of different types of communication and computing resources

When considering QoS-aware Web services, we suppose that the interface specifications are extended with statements on QoS that can be associated to the whole interface or to individual operations and attributes. In the case of a service requestor, these statements describe the required QoS associated with the service required by the client, while from a service provider's perspective these statements describe the offered QoS associated with the service offered by the server object.

The Web service architecture design from IBM includes a separate layer called "endpoint description" to add additional semantics to service description like QoS properties.

QoS negotiation & binding establishment

The following steps should be performed during binding establishment using a QoS-enabled Web services platform:

1. The service requestor requests the establishment of the binding by specifying the reference to a Web service interface. This request also contains the required QoS.
2. The QoS broker searches for the service providers in the UDDI.
3. The QoS broker performs QoS negotiation as described below.
4. The Web service QoS broker compares the offered QoS with the required QoS and uses its internal information to determine an agreed QoS. This process is called QoS negotiation.
5. If the QoS negotiation has been successful, the service requestor and service provider are informed that a negotiation has been successful and a binding has been built. From this moment on these objects can interact through the binding.

Bottlenecks in performance of Web services

Web services can encounter performance bottlenecks due to the limitations of the underlying messaging and transport protocols. The reliance on common widely accepted protocols such as HTTP and SOAP, however, make them a permanent burden that must be shouldered. Thus it is important to understand the workings of these limitations.

HTTP

HTTP is a best-effort delivery service. It is a stateless data-forwarding mechanism which tends to create two major problems:

- There is no guarantee of packets being delivered to the destination.
- There is no guarantee of the order of the arriving packets.

If there is no bandwidth available, the packets are simply discarded. Bandwidth is clearly a bottleneck, as users and amounts of data running over the network increase. Traditionally, many applications assume zero latency and infinite bandwidth. Also traditionally, applications use synchronous messaging. Synchronous messaging is fine when you run an application on your own computers; components communicate with latencies measured in microseconds. However, with Web services, they communicate across the Internet, which means latencies are measured in tens, hundreds, or even thousands of milliseconds.

Although newly designed protocols like Reliable HTTP (HTTPR), Blocks Extensible Exchange Protocol (BEEP), and Direct Internet Message Encapsulation (DIME) can be used, widespread adoption of these new protocols for Web service transport like HTTPR and BEEP will take some time. Hence, application designers making use of Web services should understand performance issues of Web service such as latency, and availability while designing their systems. Some of the ways to improve Web service performance are given below.

Use of asynchronous message queues
Applications which rely on remote Web services can use message queuing to improve reliability, but at the cost of response time. Applications and Web services within an enterprise can use message queuing like Java Messaging Service (JMS) or IBM MQSeries for Web Service invocations. Enterprise messaging provides a reliable, flexible service for the asynchronous exchange of critical data throughout an enterprise. Message queues provide two major advantages:

1. It is asynchronous: A messaging service provider can deliver messages to the requestor as they arrive and the requestor does not have to request messages in order to receive them.
2. It is reliable: A messaging service can ensure that a message is delivered once and only once.

In the future, *Publish & Subscribe* messaging systems over the Internet such as the Utility Services package from alphaWorks , can be used for Web service invocations (see Resources).

Private WANs and Web service networks
Use of private WANs/extranets and Web services networks can be a suitable option for businesses depending on Web services which are mission-critical. These private networks provide low network latency, low congestion, guaranteed delivery, and non-repudiation. However, in some cases it could be costly to have a private network.

SOAP and performance

SOAP is the defacto wire protocol for Web services. SOAP performance is degraded because of the following:

- Extracting the SOAP envelope from the SOAP packet is time-expensive.
- Parsing the contained XML information in the SOAP envelope using a XML parser is also time-expensive.
- There is not much optimization possible with XML data.
- SOAP encoding rules make it mandatory to include typing information in all the SOAP messages sent and received.
- Encoding binary data in a form acceptable to XML results in overhead of additional bytes added as a result of the encoding as well as processor overhead performing the encoding/decoding.

The XML processor must be loaded, instantiated, and fed with the XML data. Then the method call argument information must be discovered. This involves a lot of overhead as XML processors grow to support more XML features.

The role of the XML parser in SOAP performance

Most existing XML parsers are too expensive in terms of code size, processing time, and memory foot print because these parsers have to support a number of features like type checking and conversion, wellformedness checking, or ambiguity resolution. All these make XML parsers require more computing resources. Some applications can consider using of stripped down version of XML parser which have a small code size and memory foot print.

Also, most of the current SOAP implementations are Document Object Model (DOM) based. DOM parsers are inherently slow to parse the messages. SAX-based SOAP implementations can be used to increase throughput, reduce memory overhead, and improve scalability.

Compressing XML

SOAP uses XML as its payload. And if we consider thousands of SOAP messages being transmitted over the Web,

the network bandwidth is being pushed to its limit. XML's way of representing data usually results in a substantially larger size than representing the same data in binary, which is on average 400% larger. This increase of the message size creates a critical problem when data has to be transmitted quickly, which effectively results in increase of the data transmission time. Some application designs should consider techniques for compact and efficient representation. One of the ways to achieve this can be to compress the XML -- especially when the CPU overhead required for compression is less than the network latency.

Other factors affecting Web service performance

There are still more factors that can affect Web service performance that are outside the control of the Web service application, such as:

- Web server response time and availability.
- Original application execution time like EJB/Servlets in Web application server.
- Back-end database or legacy system performance.

Approaches to provide proactive Web service QoS

Service providers can proactively provide high QoS to the service requestors, by using different familiar approaches like caching and load balancing of service requests. Caching and load balancing can be done at both Web server level and at Web application server level. Load balancing prioritize various types of traffic and ensure that each request is treated appropriately to the business value it represents.

A Web service provider can perform capacity modeling to create a top-down model of request-traffic, current capacity utilization, and the resulting QoS. A service provider can also categorize Web service traffic by the volume of traffic, traffic for different application service categories, and traffic from different sources. This will help in understanding the capacity that will be required to provide good QoS for a volume of service demand and for future planning like capacity and type of load balancing Web application servers and/or Web servers (for example, the number of servers required for setting up a clustered server farm).

Service providers can provide *differentiated servicing* by using the capacity model to determine the capacity needed for different customers and service types and by ensuring appropriate QoS levels for different applications and customers. For example, a multimedia Web service might require good throughput, but a banking Web service might require security and transactional QoS.

Transactional QoS

Transactional QoS refers to the level of reliability and consistency at which the transactions are executed. Transactional QoS is crucial for maintaining the integrity of a Web service. Transactions are very important for business processes to guarantee that a set of related activities are treated and completed as a single unit of work. If an exception occurs while executing a transaction, the transaction has to be recovered back to a consistent state. This property is called the "atomicity" of a transaction. Other than property of atomicity, transactions in a stricter sense should satisfy consistency, isolation and durability properties. All these four properties are together called "ACID" properties (see the sidebar).

ACID properties of transactions

Atomicity: A transaction is an atomic unit of processing; it is either performed in its entirety or not at all.

Consistency: A correct execution of the transaction must take the system from one consistent state to another.

Isolation: A transaction should not make its updates visible to other transactions until it is committed. That is, it should run as if no other transaction is running.

Durability: Once a transaction commits, the committed changes must never be lost in the event of any failure.

There are several approaches to provide transactional QoS. The most popular approach, which is traditionally used in Web application architectures is the two-phase commit. Two-phase commit provides a transaction coordinator which controls the transaction based on the idea that no constituent transaction is allowed to commit unless they are all able to commit. This approach of using a transaction coordinator to ensure atomicity is used in Java Transactional Service (JTS), CORBA OTS, and in most database management systems.

But there are new complications when we are thinking of transactions involving Web services. The Web services used by a particular application or Web service are often distributed remotely over the Web as well as owned by different parties. Having a central transaction coordinator, which dictates the rules and performs the commits and rollbacks, in a Web services environment is very tedious to implement considering the transaction

coordinator does not have full control over all the resources. Also, two-phase commit protocol involves one or other form of resource locking. Longer periods of resource locking will result in serious scalability issues. Therefore even though it is possible to use, extreme care should be taken to make sure that resources are not locked for long periods of time.

Web Services ToolKit proxygen

The Service Proxy Generator tool can be used to create a client proxy that can interact with a Web service. This tool inspects a WSDL document and generates the Java programming language classes that can be used to invoke a Web service. This tool was a part of the WSDL Toolkit. This tool can be run by using the proxygen command. This command is located in the WSTK_HOME/bin directory. This command has one required input parameter. This parameter is the filename of the WSDL service description document.

The OASIS Business Transactions technical committee has released the Business Transaction Protocol (BTP) which extends the two-phase commit transaction management approach to address the needs of disparate trading partners that use XML to exchange data over the Web. BTP allows both the normal ACID transactions and non-ACID transactions (termed as "cohesions") for long lasting transactions that span multiple enterprises.

Another approach called compensation is based on the idea that a transaction is always allowed to commit, but its effect and actions can be cancelled after it has committed. For example, it may be possible to order a shipment, and then later cancel the shipment if the required shipment process has not yet started. Canceling the shipment is an example of a compensating transaction; it compensates for the initial transaction that ordered the shipment. In compensating transaction, each "real" transaction has an associated "compensating" transaction. This "compensating" transaction element describes a way to revert changes done by the "real" transaction and return to a previous consistent state. If any transaction aborts, the caller executes the corresponding compensating transaction for all the transactions that have previously committed. Two major problems associated with compensating transactions are:

- Compensating transactions, unlike two-phase commit, may not satisfy all the four "ACID" properties at all times -- this means there is always a probability for a failure.
- Traditionally designed two-phase commit transactions have to be redesigned to provide way for compensation.

A simple method to measure response time of your Web services

A simple method to measure the performance characteristics of your Web services can be developed by adding a little bit of extra functionality in the service proxy. Service proxies in Web services are similar to stubs in Java RMI. They contain the code that is specific to a binding within the service interface, thereby hiding the complex network communications details from the client. For example, if the binding is a SOAP binding, then the service proxy will contain SOAP-specific code that can be used by the client to invoke the service.

The steps involved in developing a proxy capable of measuring response time is as follows:

1. Generate service proxy from the WSDL service definition file.
2. Modify the generated service proxy to add code to clock the time (see Listing 2).
3. Re-compile the modified service proxy.
4. Develop a client program to create a object of the service proxy and invoke the necessary methods .

Step 1: Generate a service proxy from service definition

Typically, service proxies are not written by the programmer. Service proxies can be easily generated from the WSDL file. Web Service Toolkits (including the alphaWorks WSTK) provide tools to generate service proxies (see the sidebar). A sample WSDL service definition for an EchoService is given in Listing 1. This is a simple Web service, which echos back the original string with "Hello" appended to it.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="EchoService"
  targetNamespace="http://www.echoserviceservice.com/EchoService-interface"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.echoserviceservice.com/EchoService"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<message name="InechoRequest">
  <part name="meth1_inType1" type="xsd:string"/>
</message>
<message name="OutechoResponse">
  <part name="meth1_outType" type="xsd:string"/>
</message>
<portType name="EchoService">
```

```
  <operation name="echo">
    <input message="InechoRequest"/>
    <output message="OutechoResponse"/>
  </operation>
</portType>
<binding name="EchoServiceBinding" type="EchoService">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="echo">
    <soap:operation soapAction="urn:echoservice-service"/>
    <input>
      <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:echoservice-service"
          use="encoded"/>
    </input>
    <output>
      <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:echoservice-service" use="encoded"/>
    </output>
  </operation>
</binding>
<service
      name="EchoService">
  <documentation>IBM WSTK 2.0 generated service definition file</documentation>
  <port binding="EchoServiceBinding" name="EchoServicePort">
    <soap:address location="http://localhost:8080/soap/servlet/rpcrouter"/>
  </port>
</service>
</definitions>
```

Step 2: Modify the generated service proxy

Even though the machine-generated Service Proxy code is not to be edited, let us slightly bend this rule by adding a few lines of code. These added lines instantiates a Timer object to measure the time it takes to bind to the server and invoke a method. This is illustrated in the sample code given in Listing 2.

```
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.encoding.*;
import org.apache.soap.rpc.*;
import org.apache.soap.util.xml.*;
import mytimer.Timer;
public class EchoServiceProxy
{
  private Call call = new Call();
  private URL url = null;
  private String SOAPActionURI = "";
  private SOAPMappingRegistry smr = call.getSOAPMappingRegistry();
  public EchoServiceProxy() throws MalformedURLException
  {
    call.setTargetObjectURI("urn:echoservice-service");
    call.setEncodingStyleURI("http://schemas.xmlsoap.org/soap/encoding/");
    this.url = new URL("http://localhost:8080/soap/servlet/rpcrouter");
    this.SOAPActionURI = "urn:echoservice-service";
  }
  public synchronized void setEndPoint(URL url)
  {
    this.url = url;
  }
  public synchronized URL getEndPoint()
  {
    return url;
  }
  public synchronized java.lang.String echo
    (java.lang.String meth1_inType1) throws SOAPException
  {
    if (url == null)
    {
      throw new SOAPException(Constants.FAULT_CODE_CLIENT,
      "A URL must be specified via " +
      "EchoServiceProxy.setEndPoint(URL).");
    }
    call.setMethodName("echo");
    Vector params = new Vector();
    Parameter meth1_inType1Param = new Parameter("meth1_inType1",
      java.lang.String.class, meth1_inType1, null);
    params.addElement(meth1_inType1Param);
    call.setParams(params);
```

```
    // Start a Timer
    Timer timer = new Timer();
    timer.start();

    Response resp = call.invoke(url, SOAPActionURI);

    // Stop the Timer
    timer.stop();
    // Print the response time by calculating the difference
    System.out.println("Response Time = " + timer.getDifference());

    // Check the response.
    if (resp.generatedFault())
    {
      Fault fault = resp.getFault();
      throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
    }
    else
    {
      Parameter retValue = resp.getReturnValue();
      return (java.lang.String)retValue.getValue();
    }
  }
}
```

Step 3: Re-compile the modified service proxy

The modified service proxy source file has to be recompiled simply by using javac command or by using any other compiler.

Step 4: Develop a client program

Develop a client application, which can use the service proxy to invoke the Web service. This could be a simple Java program or a AWT/Swing based Java GUI application.

Conclusion

Quality of services is an important requirement of business-to-business transactions and thus a necessary element in Web services. The various QoS properties such as availability, accessibility, integrity, performance, reliability, regulatory, and security, need to be addressed in the implementation of Web service applications. The properties become even more complex when you add the need for transactional features to Web services. Some of the limitations of protocols such as HTTP and SOAP may hinder QoS implementation, but there are a number of ways to provide proactive QoS in Web services.

Resources

- Reliable HTTP (HTTPR) implements atomicity to HTTP.
- The Blocks Extensible Exchange Protocol (BEEP) can also package Web services data for delivery.
- Read the Direct Internet Message Encapsulation (DIME) specification to understand data encoding in Web services.
- Learn more about the IBM Web services conceptual architecture.
- Download the Web Services ToolKit from alphaWorks.
- Learn more about Java Messaging Services.
- Learn more about the OASIS group's Business Transaction Protocol (BTP).
- Learn more about message queuing and IBM MQSeries.

About the authors

Anbazhagan Mani is a software engineer at IBM Software Labs in India. He has experience working in WebSphere family of tools, XML, Java technologies, BPM, Workflow, and Object technologies. Recently, he has been working on Web services QoS, P2P computing, and Business Process Integration. You can reach him at

manbazha@in.ibm.com.

Arun Nagarajan is a software engineer at IBM Global Services in India. He has previously worked in XML and Java technologies like JavaBeans, J2EE, and WebSphere. Currently, he has been working in different Web services technologies such as SOAP, WSDL, UDDI, etc. You can contact him at anagaraj@in.ibm.com.

Close [x]

developerWorks: Sign in

If you don't have an IBM ID and password, register here.

IBM ID:
Forgot your IBM ID?

Password:
Forgot your password?
Change your password

After sign in: Stay on the current page

Keep me signed in.

By clicking **Submit**, you agree to the developerWorks terms of use.

Submit Cancel

The first time you sign into developerWorks, a profile is created for you. This profile includes the first name, last name, and display name you identified when you registered with developerWorks. **Select information in your developerWorks profile is displayed to the public, but you may edit the information at any time**. Your first name, last name (unless you choose to hide them), and display name will accompany the content that you post.

All information submitted is secure.

Close [x]

Choose your display name

The first time you sign in to developerWorks, a profile is created for you, so you need to choose a display name. Your display name accompanies the content you post on developerWorks.

Please choose a display name between 3-31 characters. Your display name must be unique in the developerWorks community and should not be your email address for privacy reasons.

Display name: (Must be between 3 – 31 characters.)

By clicking **Submit**, you agree to the developerWorks terms of use.

Submit Cancel

All information submitted is secure.

Average rating (84 votes)

1 star 1 star
2 stars 2 stars
3 stars 3 stars

4 stars 4 stars
5 stars 5 stars

Submit

Add comment:

Sign in or register to leave a comment.

Note: HTML elements are not supported within comments.

Notify me when a comment is added1000 characters left

Post

Thank you. I understood the QoS from this article.

Posted by **AreegSamir** on 20 August 2013

Report abuse