The process of developing a solution architecture and successfully implementing it through
development and deployment phases requires that performance be considered from the start.
With any solution, the overall coordination and performance of the solution components need
to execute well together in order for the solution to be truly utilized in a production environment.
New technologies are often marketed with grand promises to solve business problems, but if
not employed properly with a mind toward operational efficiency, they can result in response
time and scalability issues for your solutions. With the introduction of web services as an open
standards integration technology for Enterprise Application Integration (EAI) and Business
to Business (B2B) Integration, you can do many things to promote operational efficiencies
and thus ensure the successful architecture and deployment of a solution. This article shares
real-world experiences and suggestions for how best to architect, develop, and deploy web
services-based solutions.

Introduction: Performance criteria

Performance of a Web application is often measured by how quickly it responds to URL requests.
However, a broader evaluation of performance should also include the effects of simultaneous
requests, latency in responding to requests, scalability of a solution to handle growth in demand,
and levels of operational degradation due to increases in transaction loads.

The requirements gathering process of any project should incorporate operational aspects that
quantify acceptable performance criteria of the solution. These criteria need to be defined in
measurable parameters such as being able to support a given number of users, a given number
of simultaneous requests, or transactions to be completed within given periods of time. The IT
Architect of the solution must make sure these non-function requirements are satisfied by the
overall design, selection of technology, development of solution components, and deployment
configurations.

The architecture should guarantee the behavior of the application and its systems operate within
a range of acceptable measurements; it must also ensure that the overall behavior of the solution
is predictable. For example, as the demand on the solution increases with a greater number of requests, its processing time of the requests, in the worst case, should increase gradually with increases in demand, as opposed to the solution suddenly not being available to process requests.

Based on the scope and size limitation of this article, I concentrate primarily on considerations for how to reduce processing times of web services functions to improve overall response time in processing SOAP request and response messages.

**Leveraging web services**

Since this article is part of a series on web services best practices, I focus on performance considerations for web services-based solutions. Let's first acknowledge that your decision to use web services as an integration technology will never be justified based on its performance capabilities (at least not within the foreseeable future). The value proposition of web services is its enablement of interoperability, flexibility, reuse of assets, reduction in development and operational costs, and better relationships with business partners and customers.

The first article in this series outlined three types of web services:

- **Enterprise web services** are web services that provide a WSDL description but MAY make use of proprietary application messaging or transport protocols. An example of such a service is one that sends SOAP messages over IBM MQSeries messaging systems using JMS.
- **Internet web services** are enterprise web services that MUST only use open application messaging or transport protocols. An example of such a service is one that sends XML messages over HTTP.
- **XML web services** represent the very small subset of Internet web services that MUST use the W3C's adopted XML-based messaging protocol over a narrow range of transport protocols. Specifically, XML web services will only send SOAP messages, and only send them over HTTP, SMTP, or raw TCP/IP connections.

Careful analysis is recommended to determine which type of web services you'll be utilizing in your solution in order to best leverage the value of a service-oriented architecture.

**Validating the use of web services**

It's important to remember that it is acceptable not to use web services at the boundary between solution components. In fact, you need to make conscious decisions about where you utilize web services and make these decisions based on technical requirements that address business needs. In some cases these decisions will involve trade-offs, but with proper requirements gathering and prioritization, many of these choices will be obvious and easily justified. For example, interoperability with business partner applications over the Internet may be a higher priority than achieving the most optimal performance; thus, XML web services is justified. Similarly, ensuring authenticity and confidentiality of messages exchanged over the Internet using SSL may be sufficient from a security perspective as opposed to providing this level of protection between the application end points themselves using XML Encryption and XML Digital Signatures when you consider the performance impact of the latter.
Web services should only be employed where you have clear requirements that justify them. The justification can be either quantitative (interoperability is possible between disparate systems with different programming models) or qualitative in nature, such as positioning your business assets to align with a corporate strategy that in the future will result in measurable benefits (such as reuse of legacy code to enable J2EE, .Net, and Web applications to provide aggregated services). Using Enterprise web services between intranet Java applications would be easy to justify because RMI/IiOP provides significantly better performance than SOAP/HTTP with today's J2EE runtimes and parsers.

**Granularity of Services**

Once you have requirements that dictate web services as a viable integration technology, determining how best to leverage them to gain the most benefit will be your next step. This involves deciding on the level of granularity of the functions to be exposed as web services to avoid unnecessary message exchanges across the Internet. As a performance rule, you should strive to publish coarse grain services that themselves accept all necessary parameters and information, thereby allowing the service provider to accomplish as much as possible on behalf of the consuming application. The goal is to minimize the number of requests a consumer makes in order to accomplish a set of business tasks. This will ensure minimal effects due to network latency, system I/O, and thread/process wait states that when aggregated with multiple requests can result in significant delays. For example, you might consider supporting a purchase order service by allowing the consumer to specify product SKU numbers, quantities, credit/debit card information, billing and shipping address information, and discount coupons all within a single request. The request itself may initiate multiple atomic transactions (credit card authorization, submission of charges, inventory query and update, and order fulfillment) at the service provider that can each be undone or reversed if needed while supporting the overall business process.

**Web service performance bottlenecks**

Without a doubt, the three most common bottlenecks in today's web services-based solution are related to:

- Parsing of SOAP messages. The larger the message, the longer the time required to parse it.
- Marshalling and un-marshalling of Objects to XML and XML to Objects. The more complex the structure of a message, the longer the time required to map between programming objects and XML elements.
- Processing of WS-Security capability that includes XML Digital Signatures and XML Encryption. Security between application end points isn't free and can add surprisingly long latency to the processing of service requests.

The web services components that represent these functions are illustrated in Figure 1 for the processing of a Web Service request.
A similar component stack exists for the processing of the web service response with the lower stacked function being swapped for the consumer and service provider.

Below are a set of common practices that will direct your design to optimize your solution's efficiency with respect to the three bottlenecks itemized above. Most recommendations involve trade-offs for improving performance, but should be considered with respect to your overall set of requirements.

**Payload size versus payload complexity**

The obvious rule for optimizing your web service performance is to keep your payload small and simple. However, in the real world where you're trying to solve real business problems, you do not always have the luxury of adhering to this rule. Long running business processes may require that XML documents be exchanged that capture not only the relevant business information, but also the state of the process. Larger messages result in longer parsing times while complex XML structures with nested elements result in longer times for the marshalling and un-marchalling of Objects and XML elements. The goals should be an awareness of these impacts and spending time architecting your programming objects to minimize the size and complexity of the XML message structures. However, you should choose to support a single invocation that includes a somewhat larger and more complex message versus supporting separate individual message transactions. Consider multiple messages when the business functions or performed tasks are truly independent, or when one task may significantly delay the processing of other combined tasks.
**Figure 2: Impact of parsing and marshalling**

*Historical Perspective*

<table>
<thead>
<tr>
<th>Web Services Improvements</th>
<th>Web Services Implementation</th>
<th>WAS Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Stream</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serialization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>De-Serialization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parsing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Container</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2** represents the performance of popular SOAP runtimes, Apache SOAP, and IBM WebSphere Application Server (IBM source: Harvey Gunther's performance analysis). It diagrams the impact of parsing XML messages that average between 20% to 35% of the processing time of simple business functions. It should be noted that much progress has been made recently in optimizing parsers as shown for the WebSphere Application Server V5.02 SOAP runtime. However, they still make up a significant portion of the web services-specific processing.

**Document/Literal versus RPC/SOAP encoding**

The Document/Literal messaging style should be used for your web services whenever possible for two reasons. First, it promotes interoperability in compliance with the Web Service Interoperability (WS-I) Basic Profile 1.0. The second reason falls within the scope of this article on performance. Document/Literal results in smaller and less complex messages: the XML data in the body of the SOAP message does not have to be wrappered with a method name element, and no data type attributes are inserted into the XML elements. One other benefit of the Document/Literal messaging style is that today's Integrated Development Environment (WebSphere Application Studio Development) and runtime (WebSphere Application Server) support JAX-RPC serializers and de-serializers routines for marshalling of Objects and XML elements that are optimized based specifically on the XML schemas included in WSDL, as opposed to serializers and de-serializers associated with SOAP encoding.

**Parsing of SOAP messages**

**Minimize parsing of XML data**

If a business function is to be exposed as an XML web service that leverages SOAP for both internal consumption (EAI) and for external consumption by business partners (B2B), intermediaries such as gateways or service agents should avoid or minimize parsing of the SOAP Body. If a gateway component is used to centralize access of web services to the Internet, but no network transport or message manipulation is needed (such as SOAP/HTTP to RMI/IIOP), then the gateway should not perform parsing of the SOAP body. Many system management vendors today provide service agents that front-end the actual web services. These components rely on
business context information within the SOAP body, such as business partner IDs, transaction correlators, message IDs, and authorization codes in providing their system management capabilities. Using the business context, the service agents provide statistics on business events, enforce business policies, and route requests to meet quality of service commitments. Recently, the Web Services Gateway in WebSphere Application Server V5.1 supports partial parsing of SOAP messages. Likewise, system management vendors have recently started to provide the capability to partially parse SOAP messages to minimize their impact on performance, so it's vital that these capabilities be utilized.

**DOM versus SAX parsing**

DOM (Document Object Model) is an object-based programming interface developed by the World Wide Web Consortium (W3C). It enables a programmer to access data stored in an XML document as a tree of nodes. SAX (Simple API for XML), on the other hand, is an event-based programming interface that was offered by members of the XML-DEV mailing list. SAX enables a programmer to accesses data in an XML document as a sequence of events. Since the Apache Xerces2 Java parser supports both programming models, you can easily choose the one for your programmers that best fits your needs. Both interfaces enable a programmer to manipulate the XML; however, the manner in which they perform their tasks are very different. DOM creates a tree of objects in memory with no regard to the XML elements' data types. The entire XML document is represented in memory. Thus this approach requires more memory and is not considered the best method for very large XML documents. In contrast, SAX is event-driven, and doesn't require that the entire document be in memory. However, programmers must create their own object model and manage the SAX events. As a result, SAX is more efficient than DOM when the resulting object model is simple, resulting in faster parsing by SAX. It is recommended that if you use the Xerces parser that you ensure you have the most recent version (V2.6.0 versus 1.4.0) as significant performance enhancements have been included over the past year.

As a note, the improvements in WebSphere Application Server's SOAP parsing as illustrated in Figure 2, were partially the results of moving to SAX parsing.

Do not be surprised if your solution utilizes up to 35% of your processing time parsing SOAP messages. Remember, you’re using web services to enable interoperability and to improve operational costs and asset reuse.

**Marshalling and un-marshalling Objects and XML**

The more complex your Objects and XML Schemas are for messages, the more processing will be required by both the client and service provider. The client will have to marshal their programming objects to XML elements before issuing a request, and the service provider will have to map the XML elements to programming objects in processing the request. Objects that are constructed with arrays of arrays or XML elements consisting of nested elements of nested elements will certainly be a bottleneck if conscious decisions are not made in architecting your data for both request and response messages. Today, many companies are standardizing on using Open Application Group Information System's (OAGIS) Business Object Documents (BOD). The XML schema for these XML documents include several levels of nested XML elements, so it is important that when
using these BODs, you evaluate the impact they may have on your overall performance. The recommendation is to be selective in what BODs you use within your solution.

It is also important to architect your messages so that you maximize the amount of real information that's being exchanged. Messages with many elements and attributes and little data are usually the result of complicated XML Schemas. Most often, parsing of SOAP messages is viewed as the major contributor to performance issues with web services. However, a complex message structure can also result in more than 50% of the processing time being associated with marshalling and un-marshalling of Objects and XML elements.

Refer to Figure 2 to see examples of how marshalling and un-marshalling (also known as serialization and de-serialization) of programming objects and XML elements can contribute to your overall processing times.

Choosing your security approach

In real-world solutions, all information that is confidential or has market value must be carefully protected when transmitted over the open Internet. This often means that senders and receivers of information are authenticated (verification of both parties), the authenticity and integrity of the message is ensured (verification the message has not been altered), and the message is kept confidential (encrypted to avoid its contents being understood by anyone but the intended receiver). Providing security of messages can involve very complex processes, but there are several alternatives available today within the industry. For the IT Architect and IT Specialist, it's a matter of deciding which approach best addresses their requirements and then configuring middleware and infrastructure components to enable these security features.

Security between network nodes over the Internet is traditionally provided using SSL over HTTP (HTTPS) through web servers and application servers. With HTTPS, you can perform mutual authentication of both the sender and receiver of messages and ensure message confidentiality. This process involves X.509 certificates that are configured on both sides of the connection and are typically associated with the network nodes (systems on which the consumer and service provider are deployed or gateway systems that are hosting SOAP intermediaries).

When security is needed from end to end up through the application stack, or when security must be independent of the networking protocol, then other means must be considered. WS-Security provides authentication and message integrity through XML Digital Signatures and message confidentiality through XML encryption using X.509 certificates in both instances. However, there are performance trades-offs that must be understood and evaluated against the overall set of requirements. It's probably safe to say that enabling security through WS-Security technologies is at least twice the cost of proving similar capabilities using traditional SSL with HTTP. However, if the business logic and databases that are used in processing your requests are also a majority of your execution time (in addition to parsing and serialization), the differences in the two security approaches may not be significant enough to warrant any concern.

A common practice is to combine the two approaches by using SSL for encryption and then using XML Digital Signatures to authenticate the application end points and to ensure message
integrity. Keep in mind that SSL will also involve at least one authentication of the server to whom a message is being sent, thus some redundancy is occurring. Your business and technical requirements and evaluation of trade-offs will direct you to one of the three alternatives as a viable option.

**Summary**

Successfully optimizing performance for web services is part experience, part art, and part discipline in being systematic in your approach to measuring criteria, analyzing information, and making sound adjustments. First, you must make good decisions during your architectural and design phases as described above. Then once you have a solution that is operational, it is an iterative process to fine tune your solution by capturing measurements from simulated loads and making adjusting and measuring again to understand their influence.

The next article in this column will discuss other performance related issues to web services interactions.
Related topics

- Check out the entire Best Practices column series.
- You can find more Patterns for e-business Redbooks from IBM.
- Requirements associated with Web Services Interoperability Organization’s Basic Profile V1.0.
- Get more information on Apache Xerces parser.
- Find more information on Performance Tuning of WebSphere Applications at InfoCenter.
- You can find information on the invocation Pattern for UDDI Inquiry from the UDDI.org site.

© Copyright IBM Corporation 2004
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)