

Think Inside the Box!

Optimizing Web Service Performance, Today

Stephen J. Zilora
Sai Sanjay Ketha
Rochester Institute of Technology
Rochester NY, 14623

ABSTRACT

While Web Services Technology holds great promise for universal integration, several obstacles stand in the way of its acceptance. Work is being done to address these obstacles to allow adoption of web services technology in the future, but where do we stand today? In particular, what can be done today to combat the often cited problem of slow response times for web services? While XML hardware acceleration and SOAP compression schemes can improve the overall response, the authors have found that appropriate selection of client software, server software, and data structures can have a substantial impact. It is possible to have a profound impact on performance using tools that are routinely and dependably available to us now.

INTRODUCTION

For years, the holy grail of programming has been the “write once, run anywhere” objective. In the world of distributed computing, this notion has evolved into “write once, access anywhere”. Web Services Technology (WST) aims to support the model of a backend service that is accessible from any client platform. WST utilizes message protocols and standards (e.g., SOAP, WSDL) that can be implemented by virtually any application to provide both hardware and software independence. While this greatly facilitates the realization of true global computing, the resulting reality also places tremendous demands on telecommunication infrastructures. Exponentially increasing demands for bandwidth and processing power demand that we seek the most efficient means for implementing WST. The impact of WST on the telecommunications industry is not limited to this role of supporting internet traffic. Just like any other industry, its customers now expect the transparent integration of services that WST supports. This promise of ubiquitous, seamless integration of programs is quite alluring. So, have we arrived at this land of programming nirvana?

Not quite. There are several major obstacles preventing widespread adoption of web services today [1]. While Heffner cites security, transactions, and management as three of the obstacles, one of the most frequently discussed concerns is performance. (A Google search of “web service” and “performance” results in over one million hits.) This list of obstacles may seem rather ominous, but there is plenty of

good news. First, there are many scenarios where all of these issues do not come into play. Second, when they do come into play, there are ways of mitigating their effect or developing less desirable but workable alternatives. This article presents information on dealing with the issue of performance.

BACKGROUND

Imagine two business people. Yi is from China and speaks only Mandarin. Jacques is from France and speaks only French. They meet in Egypt to conduct business, but, of course, cannot understand one another. They cannot find anyone who speaks both Mandarin and French, but they do find Ahmed, who speaks Mandarin and Arabic, and Atef, who speaks French and Arabic. The four gentlemen then gather and discuss business. Yi speaks Mandarin to Ahmed; Ahmed translates to Arabic and speaks to Atef; Atef translates and speaks French to Jacques. The process is reversed when Jacques replies to Yi. While this may seem complicated and burdensome, it works. Yi and Jacques are able to successfully conduct business.

Web services mimic this scenario by providing a framework for translators and a common language. The SOAP protocol used by web services technology defines what each message should look like and that the message be an XML document. So as long as program 1 can compose a message and format it according to SOAP, and program 2 can receive that SOAP-formatted message and translate it into its own native format, programs 1 and 2 can exchange information.

This new model of information exchange brings with it several issues. First, there is the issue of security. Anytime we, or programs, have a conversation, there is potential that we will be overheard. There are actually two different scenarios here. The first, and most common today, is this simple case of being overheard. Using security protocols such as Secure Sockets Layer (SSL), Transport Layer Security (TLS), IP security (IPsec), Virtual Private Network (VPN), or others can mitigate this exposure. How much security is necessary? It depends on your situation. For a small company running a contact management application, SSL (or something comparable) is probably sufficient. For a large corporation conducting e-commerce, it probably is not. The second part of this security issue surfaces when we need to use intermediaries. Many WST messages are not simple point-to-point exchanges.

Intermediate processing such as authentication, notarization, or other value-added services may need to be performed. These services which may consist of viewing or modifying a part of the message are performed by nodes known as “intermediaries”. The issue of which intermediary sees which part of the message (and its rights) is the subject of WS Security (from OASIS) and many other research initiatives.

The second significant issue is transaction integrity. The internet is stateless. As such, we cannot guarantee basic transactional properties. Depending on the complexity of the services, this can be addressed with security tokens, cookies, sessions, and other similar flags to establish state. Again, the practicality of these workarounds depends upon the complexity and the demands of the application.

The management issue is not one that is borne directly from WST. Rather, it is the result of the Service Oriented Architecture that WST implements. IBM [2] and others have identified the need for a well-constructed plan for designing, deploying, and managing services as corporate assets. WST does compound the problem, though. Not only are client applications running on a wide variety of platforms, but there is no record of who the users are. It is difficult to manage something you cannot control.

The remaining issue is one of performance. A great deal has been published regarding this topic. Trade journals and blogs are rife with complaints that the SOAP protocol is too “heavy”; there are calls for developing totally new approaches [3, 4]; and there are reports of the better performance of more tightly coupled protocols such as RMI and CORBA [5]. But while it is wonderful to explore what web services technology can become, a key question is what we can do with it today. It is necessary to think inside the box and optimize performance subject to current limitations. We have examined many facets of this topic and found several definitive steps that can be employed to improve web service performance.

THE BOX

There are several factors that can influence the performance of web services. Let’s look at the steps involved in exchanging information via web services.

1. The client constructs a “request” in its native language
2. The client converts this request to XML and re-formats it according to the SOAP protocol.
3. The message is sent over TCP/IP to the server.
4. The server listens on some specified port and receives the incoming message.
5. The server unwraps the message and converts it from XML to its native language.
6. The server processes the request.

A response is then constructed and returned to the client following these steps in reverse. Overall, this is a lot of work

and explains why web services are often perceived as being slow.

Step 1, Request Construction. The time it takes to construct a request in the client’s native language depends upon both the client “horsepower” and the client language. Clearly, a more powerful machine will execute code more quickly. Additionally, different languages (in particular procedural vs. interpretive) will result in different execution times. While these differences can be dramatic, they are outside the scope of web services technology so we will not examine this step directly.

Step 2, SOAP Message Construction. Again, additional “horsepower” on the part of the client will enhance the performance, but the method a particular language uses to create the SOAP message can result in more dramatic differences. Some languages use static proxy classes. This is an “early-binding” approach that limits flexibility, but provides the greatest efficiency. Other languages take a “late-binding” approach and dynamically create the proxy classes. As one would expect, this maximizes flexibility but costs several CPU cycles at runtime.

Step 3, Transmission. Bandwidth can have a tremendous impact on web service performance. The difference between a 1 GB fiber connection and a cellular connection (for a smart phone) will cause an obvious difference in access time. But even any particular connection type, there can be significant variations due to factors such as message routing. This is a well-known problem with web performance. As such, we considered this to be outside the scope of our work and attempted to hold these transmission effects nearly constant by placing all servers and clients on the same hard-wired subnet.

The principles of component design and service oriented architecture call for fine-grained objects that can be assembled into larger entities as necessary. The question becomes at what point this assembly should occur. Should this be done in the presentation layer on the client or in the web services layer on the server? We have examined this question from the point of view of performance by considering the effects of message size on overall performance.

Step 4, Server Listening. Three application servers that can be used for web services include: Internet Information Server (IIS) from Microsoft, Tomcat from the Apache Project, and Sun Application Server from Sun Microsystems. Since Sun Application Server and Tomcat have essentially the same code base, only one needs to be considered. (In order to confirm this, identical tests were run using Tomcat and Sun Application Server. Both servers yielded the same results within measurement error.) In this article we will look at the inherent performance differences between IIS and Sun Application Server.

Step 5, SOAP Message Deconstruction. Just as with client construction of a SOAP message packet, both the server “horsepower” and the choice of language can impact the time it takes to perform this step. While it may appear obvious that a more powerful server will result in better performance, it is useful to quantify that difference. How noticeable is a difference in server configuration? Similarly, how noticeable is a difference in server programming language?

Step 6, Request Processing. The time to process a request is a function of the server’s native programming language, its “horsepower”, and the nature of the request (accessing a database or other external files as opposed to a CPU-bound calculation). While this can be of interest to the overall process, it is not a web service issue. Similarly, if the time to process the request is substantial (hours or days), then the time to process the SOAP message becomes immaterial. Our work focuses on those processes where a change in the handling of the SOAP protocol will have a measurable change in the overall request time.

BASIS

When doing a performance assessment one needs to decide what to measure. In an effort to get measurable and reproducible results, it is desirable to use simple, well-defined scenarios. However, these do not always translate well into a real business scenario. We attempted to strike a balance by creating a simple contact management system. The system contained approximately 13,000 records. For this system we created the following web services:

- Service 1: returns a simple scalar representing the number of contacts stored in the system;
- Service 2: returns a 1-dimensional array containing the names of some number of contacts;
- Service 3: returns a 2-dimensional array containing the name and phone number for some number of contacts.

Services 2 and 3 each accept a parameter that specifies the number of array elements (contacts) to return. This selection of services allows us to test the earlier described parameters in the following manner.

- Server Software: we are able to run any or all of the services on two different software server platforms (Sun Application Server and Internet Information Server) and compare results.
- Client Software: we are able to access any or all of the services with different client software (Java, C#, and PHP) and compare results.
- Message Size: we are able to vary the size of the arrays and examine performance. We also are able to compare returning one large 2-dimensional array vs. two 1-dimensional arrays.

Server Horsepower: we implemented the server software and web services on two separate physical servers. One was a low-end machine with 256 MB memory and a 1.6 GHz processor. The high-end server had 1.5 GB memory and a 3.0 GHz processor. Each server was running Windows 2003 Server as its operating system.

Collecting performance data is often a debatable topic. Others have written on techniques to measure end-to-end internet performance [6] or on taking a strict engineering approach and testing at the design phase [7]. Our aim when calculating times was to look at business needs while still isolating operations.

On the client side, start time and end time were calculated just before and after service invocations. Any required processing was done outside of the timing loop. This includes construction of the connection to the server as well as the proxy classes themselves. Figure 1 depicts the timing process.

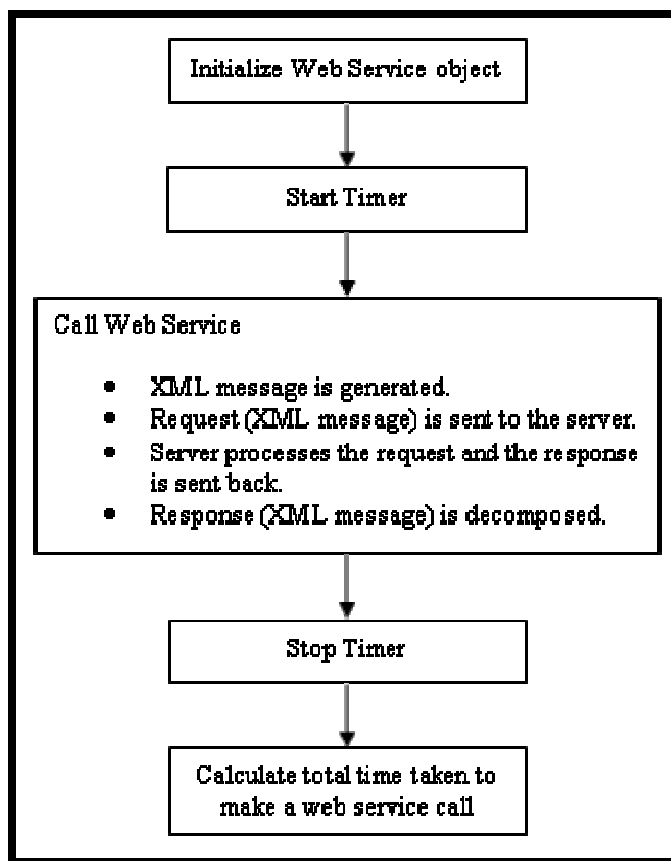


Figure 1—Timing Process

Because the values are so small, we did not rely on a single invocation. In most cases we ran the test consecutively multiple times and then calculated an average value to reflect the time of one invocation. This also helps to minimize the impact of the overhead associated with gathering the timing information.

On the server side, access to the database was isolated from the methods. The data was pre-loaded into memory with the start-up of the service. The actual web services simply return the requested data.

RESULTS

There are many parameters to study and nearly 500 tests were run to explore the various combinations. The results of these tests were quite consistent allowing us to look at slices of information and examine particular aspects in isolation. For example, Table 1a shows the total response time for the three services using each of the three client languages. The values shown in Table 1a were obtained with C# web services running in IIS on the high-end machine. Table 1b shows the results of the same clients run against Java web services running in Sun Application Server also on the high-end machine. The Scalar test required the return of 15 bytes of data; the 1-d array test required the return of approximately 200kb of data; and the 2-d array test required the return of approximately 300kb of data. (Both the 1-d and 2-d arrays contain approximately 13,000 elements.) Comparing the two data sets shows different values, but consistent trends.

Client Lang. Return Data Type	Java	C#	PHP
Scalar	98	94	101
1-d array	265	150	293
2-d array	565	325	643

Table 1a—Response Time(ms) for Client Language and Return Type Combinations (IIS Server)

Client Lang. Return Data Type	Java	C#	PHP
Scalar	7	2	15
1-d array	186	63	205
2-d array	466	253	554

Table 1b— Response Time(ms) for Client Language and Return Type Combinations (Sun Server)

The data in Tables 1a and 1b show that in all cases, the C# client yields the fastest response times. PHP was the slowest by a narrow margin for simple data types, but by a more significant margin for more complex/larger data types.

The next logical step is to examine the impact of message size on response time. Clearly C# appears to be the least sensitive to data type in Table 1, but is it a matter of data type complexity or message size? Table 2 lists the response times (using the Sun Server) for various client languages accessing Web Service 2 (return a 1-dimensional array) for increasing message sizes (as measured by the number of 15-byte array elements returned). We can see that the effect of increasing message size on response time is generally flat for all

languages. This suggests that PHP and Java begin to slow down with increasing complexity of data type, not increasing message size.

Client Lang. # Contacts	Java	C#	PHP
1	15	16	15
10	16	16	16
100	16	16	16
1000	16	16	24

Table 2—Response Time(ms) for Client Language and Message Size Combinations

But if the response time is flat with respect to increasing message size (for this range of message sizes), should we always return as large a message as possible? Consider the scenario where you wish to obtain the name and phone number of some number of contacts. Is it better to return all the information at once, or make several calls returning the name and phone number for just one contact each time. That is, is it better to return a single, large, 2-dimensional array or return multiple, small, 1-dimensional arrays? Tables 3a through 3c show the results of this examination for 3 different client languages and 4 different array sizes. Each row represents the results for a different number of contacts (array size). The first row contains the results of obtaining the information for just one contact; the second row contains the results of obtaining the information for ten contacts, and so on. There are two columns for each client language. The first column lists the time required to return the information all at once in a single, two-dimensional array via Web Service 2. The second column lists the time required to return the information via successive calls to Web Service 1. The data clearly shows that it is far better to return a single, large message regardless of the client language. The difference can be several orders of magnitude.

Array Type # Contacts	2-d Array	1-d Arrays
1	16	16
10	16	62
100	16	531
1000	47	3140

Table 3a—Response Time(ms) for Return Data Type and Message Size Combinations for the Java Client

Array Type # Contacts	2-d Array	1-d Arrays
1	16	16
10	16	31
100	16	250
1000	31	1859

Table 3b—Response Time(ms) for Return Data Type and Message Size Combinations for the C# Client

Array Type # Contacts	2-d Array	1-d Arrays
1	7	7
10	16	156
100	18	1562
1000	36	15624

Table 3c—Response Time(ms) for Return Data Type and Message Size Combinations for the PHP Client

So far we have looked at the client language and the message size for the data being returned. Next we need to look at the backend servers. The first question is does it matter whether the server is running IIS or Sun Application Server? Tables 4a and 4b list the results of running various clients against Service 1 (return a 1-dimensional array) on each of these servers. The web services were identical except that for IIS the service was written in C# and for Sun Application Server the service was written in Java. In both cases, no specific tuning was done—the application servers were run using default settings. The table shows that regardless of client language and message size, Sun Application Server always shows a substantially faster performance.

Client Lang. # Contacts	Java	C#	PHP
1	15	16	15
10	16	16	16
100	16	16	16
1000	16	16	24

Table 4a—Response Time(ms) for Client Language and Message Size Combinations for the Sun Server

Client Lang. # Contacts	Java	C#	PHP
1	94	94	96
10	94	94	96
100	94	109	96
1000	109	141	104

Table 4a—Response Time(ms) for Client Language and Message Size Combinations for the IIS Server

We can also consider the power of the server. Using the Java client running against both the Sun server and the IIS server on both the low-end and the high-end computers results in the data shown in Table 5. Not unexpectedly, performance is better on a high-end machine regardless of the server software being used. However what is also shown is that the performance difference is significantly more dramatic in the case of IIS.

Server Config. Data Type	Sun High	Sun Low	IIS High	IIS Low
Scalar	7	9	98	292
1-d array	186	190	265	468
2-d array	466	516	565	751

Table 5—Response Time(ms) as a Function of Application Server, Physical Server, and Data Type

CONCLUSIONS

Web services are about delivering interfaces to services, not components. While it is certainly appropriate to use fine-grained components as part of our overall architecture, at the level of service delivery, course-grained large data-structures are more appropriate. The cost of transmitting these large data messages over the TCP/IP protocol is small compared to the cost of wrapping and unwrapping the SOAP message envelope as evidenced by the cost of making successive calls to retrieve information versus a single call. This is universally true regardless of client language, server software, or server horsepower.

There also appears to be a clear benefit to use Sun Application Server (or Apache Tomcat) relative to Microsoft's Internet Information Server. It is important to note that the servers were not tuned in any way. It is also important to note that no load response tests were conducted. That is, when there are tens, hundreds, or even thousands of simultaneous requests, how well will these application servers respond?

A more powerful server is marginally better for the Sun application server and significantly better for the Microsoft application server. Again, this was in a low-load environment. The difference in performance between a low-end and high-end server is likely to be more pronounced with heavier loads.

On the client side, while PHP may be one of the most attractive languages to select (because it can drive a web-based interface), it is also the slowest of the languages we examined. This is most likely due to its dynamic nature. While Java and C# use a similar execution model (partial compilation plus runtime interpretation), C# appears to be more efficient.

FUTURE WORK

While we have answered several questions, our work has raised many additional questions. Key areas that need to be examined include: a more thorough analysis of the performance of Sun Application Server (or Tomcat) and IIS, examination of other languages (e.g., C++) and servers (e.g., WebSphere, WebLogic), effects of server load, sensitivity testing of more complex return types such as arrays of objects, effects of the underlying operating system, and consideration for compression techniques.

REFERENCES

- [1] Heffner, Randy, "Status Of SOA And Web Services Specifications", Forrester Research, Inc., December, 2006.
- [2] http://www-306.ibm.com/software/solutions/soa/gov/index.html?S_TACT=107AG01W&S_CMP=campaign
- [3] Suzumura, T., Takase, T., Tatsubori, M., "Optimizing Web Services Performance by Differential Deserialization", 2005 IEEE International Conference on Web Services, 2005.
- [4] Ng, A. "Optimising Web Services Performance with Table Driven XML", 2006 Australian Software Engineering Conference, 2006.
- [5] Juric, M., Kezmah, B., Hericko, M., Rozman, I., Vezocnik, I., "Java RMI, RMI Tunneling and Web Services Comparison and Performance Analysis", *ACM SIGPLAN Notices*, Vol.39(5), May 2004.
- [6] Zhu, L., Gorton, I., Liu, Y., Bui, N., "Model Driven Benchmark Generation for Web Services", *IW-SOSE'06*, Shanghai, China, May 2006, pp. 33-39.
- [7] Cherkasova, L., Fu, Y., Tang, W., Vahdat, A., "Measuring and Characterizing End-to-End Internet Service Performance", *ACM Transactions on Internet Technology*, Vol. 3(4), November 2003, pp. 347-391.

BIOGRAPHIES

Stephen Zilora (stephen.zilora@rit.edu) received his M.S. in Computer Science from the New Jersey Institute of Technology in 1996. He has designed or written numerous applications and managed software development operations for companies ranging from small firms to Fortune 50 firms. He is currently an assistant professor in the Information Technology Department of Rochester Institute of Technology. There he has spent the last 5 years investigating the benefits and shortcomings of service-oriented architectures and web services technology.

Sai Sanjay Ketha (ssk3995@rit.edu) is a graduate student in the Information Technology Department of Rochester Institute of Technology where he soon expects to receive his M.S in Information Technology. His studies are focused on application development, database design, and XML.