Performance Evaluation of XML Web Services for Real-Time Applications

Hazem M. El-Bakry and Nikos Mastorakis

Abstract- Service-Oriented Architecture (SOA) has been becoming one of the most widely used methodologies for building and integrating different types of software applications. This because the extreme benefits that it offers to their adopters including agility, dynamicity, and loose-coupling. These benefits are usually missed in traditional software terminologies and practices. XML Web Services is the most used technology for realizing SOA because it is easy to use. Furthermore, it allows high interoperability between different systems due to its dependency on standards that are widely accepted and supported by almost all large software vendors. However, XML Web Services suffers from a number of drawbacks such as low performance, bad utilization of hardware resources, and high network latency. These pitfalls may prevent some adopters from utilizing SOA in large and complex systems. Therefore, these issues should be first addressed and resolved before leveraging it into real-time systems. Here, an experimental evaluation for the performance of XML Web Services in real-time business systems is presented. Moreover, this study offers some tactics and strategies that might be used to enhance the overall performance of XML Web Services.

Keywords-SOA, XML Web Services, Problem Root Causes, Systems Integration, Performance Evaluation, Optimization Tactics and Strategies.

I. Introduction

Nowadays, SOA has gained momentum from almost all players in software market for building and integrating systems, especially complex ones that demand continuous changes to meet market ever-changing requirements [1,2,10].

Technically, many technologies could be used for realizing and implementing service-oriented systems including message queuing, remote procedure calls (RPCs), Common Object Request Broker (CORBA), and Common Object Model (COM) [4]. However, XML Web Services is the most used technology for realizing SOA due to number of factors including [5]:

- **Ease-of-Use:** Using XML Web Services does not require deep technical knowledge, as it is very easy to learn and use, especially if compared with other tools like CORBA and COM.
- **Support:** Because it has appeared as a result of cooperation between large software vendors such as Microsoft, IBM, BEA, and Sun Microsystems, it is supported in almost all software tools, frameworks, and programming languages.
- **Modularity:** It is modular by nature, so, it is easy to encapsulate logic in terms of modules that could be deployed and used separately according to business needs.
- **Compose-ability:** It is very easy to aggregate a number of XML Web Services to construct a new one that covers more complex needs.
- **Low Costs:** It is much cheaper than traditional and proprietary technologies.
- **Commonality with SOA Model:** The architectural model of XML Web Services is almost identical to the basic model of SOA, as they both have service provider, service consumer, service registry, service contract, and service itself.

Unfortunately, the aforementioned advantages of XML Web Services were not enough to enable SOA adopters to use it effectively in real-time business systems, as it still suffers from number of problems such as low performance, bad utilization for hardware resources, and high network latency. These issues must be first addressed and resolved to allow optimum utilization for SOA.

This paper examines the performance of XML Web Services in building and integrating real-time business systems taking a hypothetical scenario for banking solutions as an example to these systems in order to compare found results with those of using other traditional methods. Additionally, it discusses root causes of found problems in order to give some tactics and strategies that could be leveraged to make better use of XML Web Services.

II. Root Causes

Many studies have discussed the performance of XML Web Services from different perspectives, and they all concluded that poor performance goes back to a number of reasons including:

- **XML Data Format:** The technology of XML Web Services depends on XML (Extensible Markup Language) for representing data being transmitted between different systems and nodes. As known, XML is a tag-based language rich with different capabilities that allow easy and powerful integration between different systems. For example, it offers different mechanisms for validating, querying, and transforming data [6]. These capabilities made XML

Manuscript received June 9, 2008.
H. M. El-Bakry is assistant professor with Dept. of Information Systems - Faculty of Computer Science and Information Systems – Mansoura University – Egypt. (e-mail: helbakry20@yahoo.com).
N. E. Mastorakis is professor with Technical University of Sofia, – BULGARIA

the preferred choice of software vendors who look for interoperability and loose-coupling between integrated systems. However, this richness causes data files to be bloated with long named tags, complex data structures, and big amounts of plain-text data that make their generation and processing very complex, heavy, and slow.

- **Encoders/Decoders:** XML Web Services use encoders to transform data into a sequence of bytes before transmission, and on the other side, they use decoders to return transmitted data to its original form. ASCII was one of the most used encoding formats in legacy systems. Depending on ASCII format in XML Web Services is very expensive and slow especially for numerical values and floating points [7].
- **Parsing Techniques:** XML being transmitted between different nodes must be parsed and validates before any further processing. Using inefficient parsing techniques may require much hardware resources (including RAM and processing cycles) and long times to complete required tasks.
- **Serialization/De-serialization:** Serialization (marshaling) is the process that converts the state of objects in a form that can be transmitted over network media (such as wires and Wi-Fi) between different nodes. Conversely, de-serialization (de-marshaling) process is responsible for bringing the state information to original formats. Efficient serialization techniques must be very fast and generate serialized data in compact formats. Depending on bad serialization techniques may generate very large data outputs that might clog network during transmission process. As mentioned, XML Web Services serialize data using XML format, and because XML is text-based format, then generated messages are always very large in size if compared to original data (before serialization) [7, 8].
- **Transport Protocol:** XML Web Services use Simple Object Access Protocol (SOAP) as a lightweight communication framework that is based on XML. Although SOAP messages can use any transport protocol to send requests and receive responses, it uses HTTP as a default transport protocol**.** HTTP is a request-response protocol that supports only synchronous interaction between clients and servers, and this makes it ill-suited for message-based communications that require asynchronous interactions [9]
- **Network Infrastructure:** There is no question that, network is one of the most important factors for the success of any client/server implementations, and thus, depending on weak and slow networks can lead to unreliable, inefficient, and intermittent interactions between clients and servers. Furthermore, slow networks may lead to less utilization for available processing power because CPUs will wait longer times until data arrives to be processed.
- **Extra Elements in Software Stack:** In service-oriented systems, XML Web Services are defined in a separate layer that accepts clients' requests to be formatted before sending them to underlying components. As known, the more layers and processing logic defined in any software architecture, the slower performance of overall system. This is due to extra processing that is needed to reach final element in software stack including discovery, reflection, initialization, instantiating, and invocations of needed components and objects as well as transformations for incoming requests and out-coming data formats.

III. Technical Scenario

Integrating different systems together is a very common scenario in software field, and leveraging SOA for building and integrating different business systems like banking solutions offers many advantages over other traditional integration methods including simplicity, dynamicity, agility, and loose-coupling between integrated systems [3]. For this reason, we are going to illustrate one of these scenarios in subsequent sections.

Assume that we have the following two banking systems that need to be integrated together with minimum impacts on underlying architecture and components:

- **Core-Bank:** It is a legacy monolithic system that acts as a back-end for other operational (front-end) systems. Core-bank system consists of different business modules including (customer management, deposits, foreign exchange, commercial loans, Islamic loans, etc). Additionally, this system offers some non-functional features such as transaction support, role-based security, and auditing and logging for user actions. The core-bank system allows users to send requests and receive responses via its GUI while all processing occurs into database itself. This database consists of large number of precompiled stored procedures, and each stored procedure takes a huge number of input parameters to process them and return back generated result sets.
- **Loan-Origination:** It is a front-end system built with .NET framework 2.0 and c# language and it is responsible for allowing bank clients to issue loans, define installments, schedule payments, etc. Because all information about bank clients and loans are stored into the core-bank system, we have to integrate the loan-origination system with it. To enable this integration, we will build an XML Web Service that comprises a large number of methods responsible for accepting requests from different client (front-end) applications including the Loan-Origination application, and passing them to stored procedures that reside under the core-banking system. To enable efficient use of the new XML Web Service, software components should be built to wrap available stored procedures and encapsulate their logic. These components will divide underlying business logic that is scattered over different stored procedures into modules that could be easily used and modified whenever needed [11, 12]. Defined components could be realized (designed and built) with any modern programming environment (such as J2EE or .NET) that supports advanced features like OOP, RAD, XML Web Services, etc. After realizing needed components, they will be placed into a new layer that resides between the new XML Web Service and underlying stored procedures to act as a mediator/wrapper that accepts different invocations from XML Web Service and turn them into formats

that could be accepted by stored procedures. Figure 1 illustrates a high level view for integration architecture of our systems, whereas, figure 2 illustrates simple request and response messages that are handled by the system.

Request Message

```
<?xml version = "1.0"?>
<Request ID="86C86720-42A0-
1069-A2E8-08002B30309D"
 Date ="1/1/2008 4:30:35 PM">
<CustomerProfile>
<CustomerID>10028160</CustomerID>
<BankID>10</BankID>
<BranchID>2</BranchID>
</CustomerProfile>
</Request>
```

Response Message

```
<?xml version = "1.0"?>
<Response ID="8664DA16-DDA2-42AC-926A-
C18F9127C302" Date="1/1/2008 4:30:40 PM"
RequestID="86C86720-42A0-1069-A2E8-08002B30309D">
<CustomerProfile>
<CustomerID>10028160</CustomerID>
<CustomerName>Qusay Fadhel</CustomerName>
<CustomerStatus>Active</CustomerStatus>
<CustomerType>V.I.P</CustomerType>
<Address>
<Country>Egypt</Country>
<City>Cairo</City>
<Street>Free Zone, Nasr City</Street>
</Address>
<BankID>10</BankID>
<BankName>HSBC</BankName>
<BranchID>2</BranchID>
<BranchName>HSBC-Cairo-Al Mohandeseen</BranchName>
<Accounts>
<Account1>
<AccountNumber>00001110012586</AccountNumber>
<AccountType>Checking</AccountType>
<AccountStatus>Open</AccountStatus>
</Account1>
</Accounts>
</CustomerProfile>
</Response>
```

Figure 2: An Example for Request/Response Messages

IV. Performance Evaluation

Due to high dependency on XML Web Services, the illustrated architecture slightly suffers from low performance. In fact, leveraging SOA is always hampered by large sizes of XML files being transferred between clients and servers. These large data files always clog network and drain almost all hardware resources including RAM, CPU, and storage infrastructure. In original architecture the core-banking system was entirely built over a set of stored procedures to execute needed business logic, but it now depends on a service layer that acts as a wrapper that receives client requests and maps them to appropriate stored procedure(s). Figure 3 illustrates a comparison between the total times needed for receiving the response message of the request that gets basic information about one bank client using both methods.

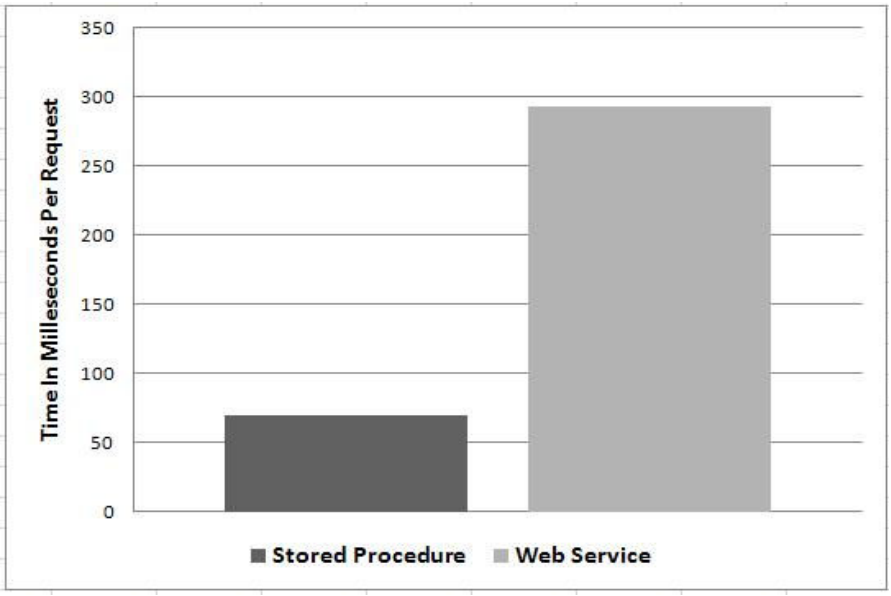
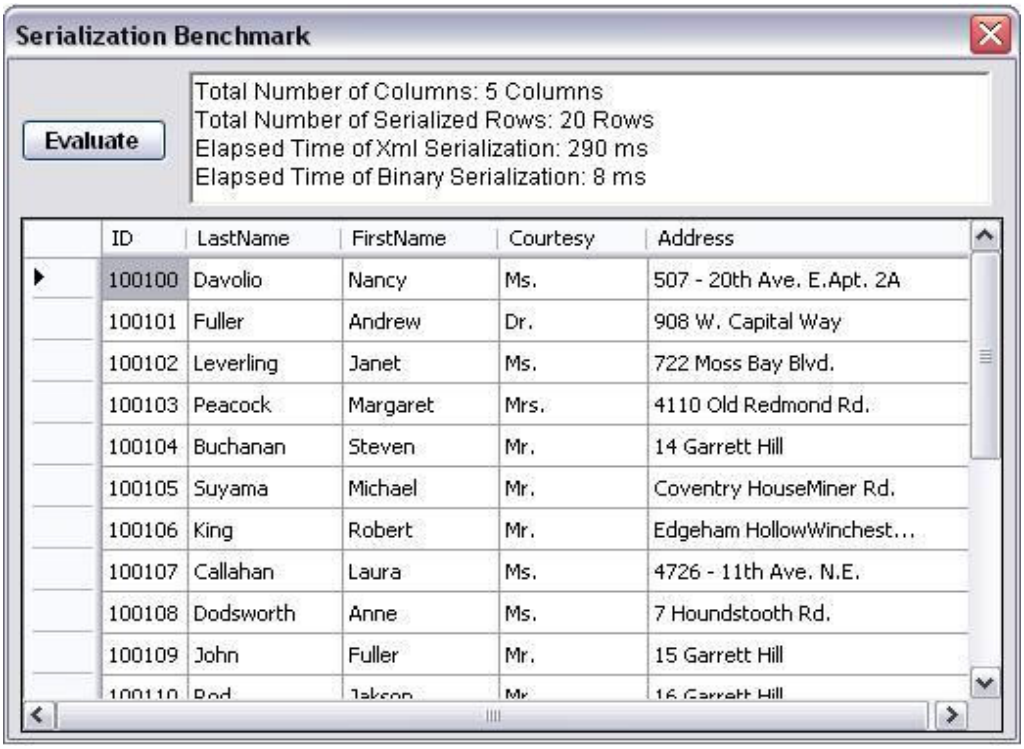


Figure 3: Invocation Time needed for Database Stored Procedure vs. XML Web Service

V. Optimization Tactics and Strategies

As illustrated, the original method that uses stored procedures is 4 times faster than the new XML Web Services method. To mitigate this problem, many experiments have been conducted to yield the following list of recommended tactics and techniques:

- **Utilize Better Encoders/Decoders:** Utilizing or even customizing more optimized encoders/decoders and encoding formats can save much of time needed for preparing data. UTF8 is a well known and standard encoding format that has been tuned to replace the traditional slow ASCII format. It is now known to be one of the fastest encoding options available in software market that supports almost all commonly used characters as well as special characters [13].
- **Leverage Binary XML:** As mentioned, most of current XML implementations depend on using *plain-text* format that causes data files to be very complex and large in size. W3C has announced that formatting XML data using binary format is more efficient for both network and hardware utilization [14]. Different techniques might be utilized to use binary data, for example, the data being transmitted between network nodes might be serialized (marshaled) and de-serialized (un-marshaled) using binary format instead of text-based XML format. Figures 2, 3 illustrate the results generated by a simple benchmark (windows forms) application that was written (using c# language and .NET Framework 3.5) to estimate the serialization time needed by binary and text-based serialization techniques on a PC that uses Intel Dual Core 2.6 GHz processor and 2 GB memory. As depicted, the application shows that XML serialization takes *290 milliseconds* to serialize some of stored (dummy) information. The amount of data is represented in *20 rows* and *5 columns (948 bytes)*, whereas, the binary serialization takes only *8 milliseconds* to serialize the same data. This means that binary serialization in our scenario saves more than 97% of time needed by XML serialization. Certainly the serialization time will vary according to number of factors including amount and complexity of data being serialized, for instance, if we serialized *10,000* rows of the aforementioned data (*569,808 bytes*) on the same PC, the XML serialization will take *709 milliseconds* versus *322 milliseconds* for binary serialization, which means that binary serialization saves more than 55% of total serialization time. We should note that there is a fixed time that is needed in each method whatever the size and amount of data to initialize serialization process. This initialization operation is solely needed to read the schema of data being serialized. Figure 5 illustrates a simple comparison between total times needed for binary serialization versus XML (text-based) serialization to return the response of a simple request.



ID	LastName	FirstName	Courtesy	Address
100100	Davolio	Nancy	Ms.	507 - 20th Ave. E.Apt. 2A
100101	Fuller	Andrew	Dr.	908 W. Capital Way
100102	Leverling	Janet	Ms.	722 Moss Bay Blvd.
100103	Peacock	Margaret	Mrs.	4110 Old Redmond Rd.
100104	Buchanan	Steven	Mr.	14 Garrett Hill
100105	Suyama	Michael	Mr.	Coventry HouseMiner Rd.
100106	King	Robert	Mr.	Edgeham HollowWinchest...
100107	Callahan	Laura	Ms.	4726 - 11th Ave. N.E.
100108	Dodsworth	Anne	Ms.	7 Houndstooth Rd.
100109	John	Fuller	Mr.	15 Garrett Hill
100110	Rod	Jakson	Mr.	16 Garrett Hill

Figure 4: Performance of Binary Serialization vs. XML Serialization

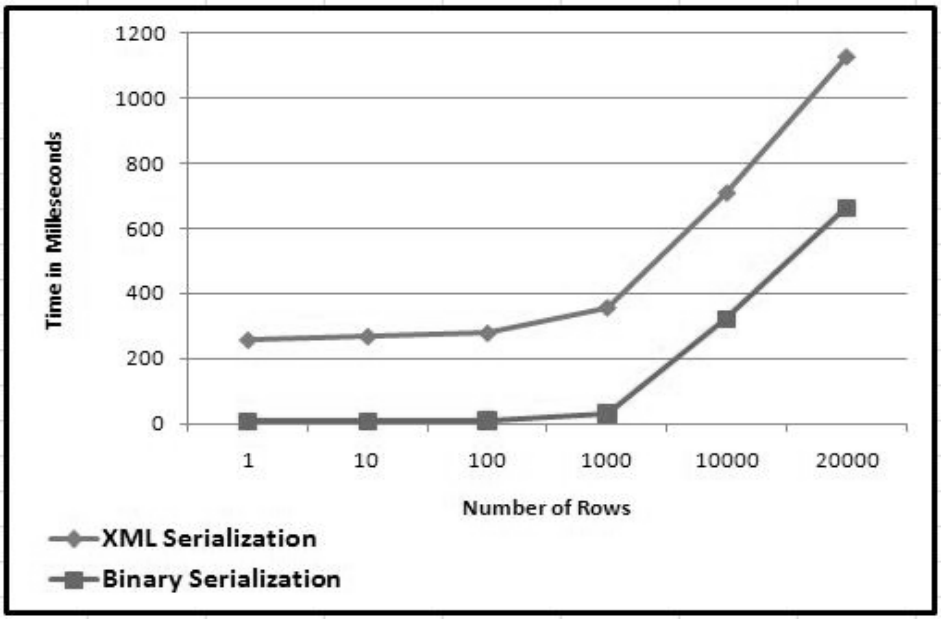


Figure 5: Binary Serialization vs. XML Serialization

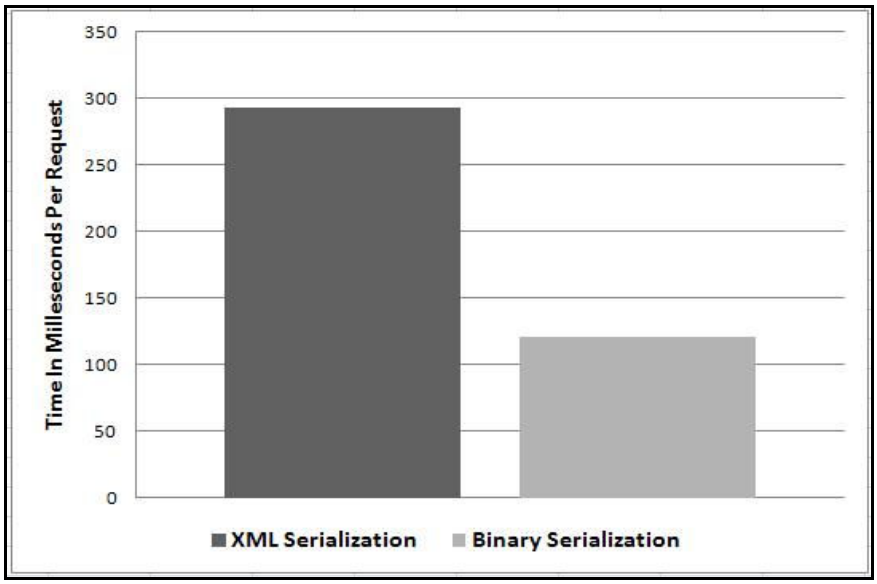


Figure 6: Invocation Time needed for XML Serialization vs. Binary Serialization

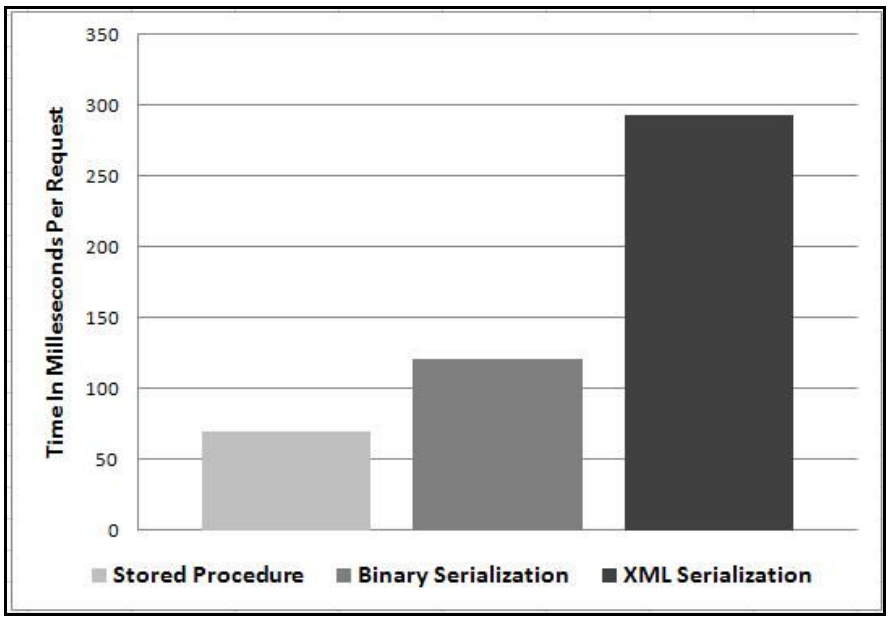


Figure 7: Invocation Time needed for Stored Procedures vs. Binary Serialization vs. XML Serialization

- **Apply Data Compression Techniques:** Since XML is a *text-based* format, and XML documents always have too many white spaces, then using traditional compression algorithms such as ZIP/GZIP for compressing data transferring between clients and servers can get rid of many bytes of the volume out of data files. To apply compression technique on traveling data, both requestors and responders must understand the used compression algorithm to be able to recognize and use these data. This assurance should be identified and guaranteed by SOA governance team during preliminary implementation phases. Figure 8 illustrates the results generated by a benchmark (console) application that has been written (using c# language and .NET Framework 3.5) to calculate total save in size of *Response* document that was illustrated in figure 2 using ZIP/GZIP algorithms. The results shows that the original size of *Response* document was *771 bytes*, whereas, the size of compressed file is only *550 bytes* which means that ZIP/GZIP algorithms save *221 bytes* (approximately 29%) from original document size. Another way that could make XML documents smaller is to avoid long element and attribute names. For example, the *Response* document illustrated in figure 2 could be abbreviated as illustrated in figure 9. The size of new abbreviated *Response* document takes only 559 bytes with total save *212 bytes* (approximately 28%) from original document size. This option will only be applicable where the human readability of request/response messages is not required. Combining two techniques together in our scenario allowed us to eliminate about 57% (the most) of total document size which is excellent to our issue while keeping the great benefits of SOA including the ease-of-use and simplicity of XML Web Services and overall architecture. Certainly, these values may vary depending on the structure and size of original documents being compressed, however, saving total size of documents (especially complex and large ones) being processed and transmitted over network has with no doubt positive

impact on memory consumption, network utilization, and transmission time. One note that we should take into consideration regarding to applying ZIP/GZIP algorithms is that processing compressed data may require more CPU usage at the receiving (last) node, as it will be a part of its responsibilities to unzip data in order to be able to use it in any further processing. This issue can be easily resolved by using more powerful servers that use high speed multi-processing cores.

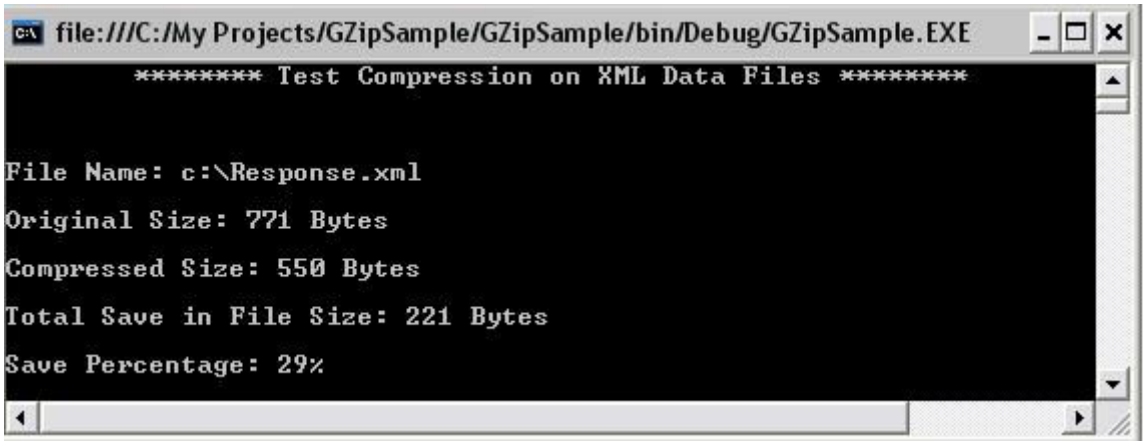


Figure 8: Total Save in Response Document Using ZIP/GZIP Algorithms

Request Message	Response Message
<?xml version = "1.0"?> <Req Id = "**86C86720-42A0-1-69-A2E8-08002B30209D**" Date = "**1/1/2008 4:30:02 PM**"> <CustProf> <CustId>**10028160**</CustId> <BkId>**10**</BkId> <BrchId>**2**</BrchId> </CustProf> </Req>	<?xml version = "1.0"?> <Res Id = "**8664DA-DDA2-42AC-926A-C18F9127c302**" Date = "**1/1/2008 4:30:29 PM**" ReqId = "**86C86720-42A0-1-69-A2E8-08002B30209D**"> <CustProf> <Id>**10028160**</Id> <Name>**Qusay Fadhel Hassan**</Name> <Stat>**Active**</Stat> <Type>**V.I.P**</Type> <Addr> <Ctry>**Egypt**</Ctry> <Cty>**Cairo**</Cty> <Strt>**Free Zone, Nasr City**</Strt> </Addr> <Bk> <Id>**10**</Id> <Name>**HSBC**</Name> <Brch> <Id>**2**</Id> <Name>**HSBC-Cairo-Al Mohandeseen**</Name> </Brch> <Accts> <Acct1> <Num>**00001110012586**</Num> <Type>**CK**</Type> <Stat>**Open**</Stat> </Acct1> </Accts> </Bk> </CustProf> </Res>

Figure 9: Abbreviated Request/Response Messages

- **Use Better Parsers:** SAX and DOM are the most popular parsing techniques available in XML market. Many benchmarks on their throughputs have shown that they require slightly long times to parse XML files at different sizes and complexity levels. The long time is mainly needed because these parsers read data files more than once (at least two times) to be able to discover their structure and to validate entire data. Furthermore, big amount of memory might be needed to store extracted data during parsing phase, and this of course may degrade overall performance of used CPUs when no more memory is available (in paging and caching operations performed by operating system). To resolve this issue, we may depend on faster and more efficient techniques such as Virtual Token Descriptor XML (VTD-XML) which depends on "non-extractive" tokenization approach to parse XML files [15]. Additionally, using *schema-specific* parsers rather than general purpose parsers can greatly enhance the performance of parsing phase [16]. Using more enhanced parsing algorithms and tools can save time needed to parse data which in turn save the overall processing time and resources.
- **Pre-generate Serialization Assemblies:** Many of development tools (including .NET 2.0 and latter) allow developers to pre-generate and cache serialization assemblies that could be deployed with applications to save time needed for discovering, extracting, and recognizing structures of objects being serialized and de-serialized [17].
- **Divide Large Files:** It is known that large files always have bad impacts on the utilization of hardware resources (memory, processor, and network) and processing time. Dividing large and complex data files into smaller pieces (if possible) can deliberately enhance both processing and network performance.
- **Install Silicon-based XML Engines:** Many silicon-based engines are available now to handle XML at higher speeds. These engines can be embedded into different network and hardware equipments including switches, routers, load balancers, PCI-cards and servers [18].
- **Apply Parallelism Techniques:** Data files could be yielded and processed in parallel using grid-based technologies that depend on mutli-threaded systems (one thread for each process/sub-process) to allow faster processing and better utilization of available hardware resources including processing cycles, memory, and storage infrastructure [19, 20, 21].
- **Utilize High Speed Networks:** There are now Ethernet implementations that enable enterprises to have transmission speed that varies from traditional 10Mbits/s Ethernet to 100Gigabit Ethernet [22, 23, 24, 25]. Also, fiber channels and links could be installed to allow high speed and reliable transmission for data encapsulated into XML requests and replies.

VI. Conclusion

The evaluation for XML Web Services in real-time business systems has been presented by illustrating a common scenario for two banking systems that need to be integrated together with contemporary SOA methodologies and terms. The use of XML Web Services in our scenario caused a number of problems including slow performance and bad utilization for hardware and network recourses over RPC implementation (stored procedures) that is widely used in traditional point-to-point integration methods. These pitfalls could not be accepted under any circumstances in real-time business systems that receive and handle tons of requests every single second. For that reason, some tips and recommended actions has been given to allow efficient and better use of SOA and XML Web Services in building and integrating real-time business applications and systems.

References

[1] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics, and Directions", Proceedings of the fourth IEEE international conference on web information systems engineering (WISE'03), 2003.

[2] K. Channabasavaiah, K. Holley, E. Tuggle, "Migrating to a service-oriented architecture, Part 1, 2", http://www-128.ibm.com/developerworks/library/ws-migratesoa

[3] A. M. Riad, A. Hassan, Q. F. Hassan, "Leveraging SOA in Banking Systems' Integration", Journal of Applied Economics Science, Romania (JAES), Volume III, Issue2 (4), Summer 2008.

[4] D. Krafzig, K. Banke, D. Slama, "Enterprise SOA Service-Oriented Architecture Best Practices". Prentice Hall. April 2005.

[5] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: An introduction to SOAP, WSDL, UDDI", IEEE Internet Computing, 6(2):86-93, March-April 2002.

[6] "Extensible Markup Language (XML) Specification Version 1.0". W3C. February 04, 2004, http://www.w3.org/tr/rec-xml/

[7] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the limits of SOAP performance for scientific computing", Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, pages 246-254, 2002.

[8] F. E. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener, "Efficient wire formats for high performance computing", Proceedings of the 2000 Conference on Supercomputing, 2000.].

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol - HTTP/1.1, 1999. IETF RFC 2616", http://www.ietf.org/rfc/rfc2616.txt.

[10] Thomas Erl, Service-Oriented Architecture (SOA): Concepts, Technology, and Design, Prentice Hall/Pearson PTR, 2007.

[11] H. M. Sneed, "Integrating legacy Software into a Service oriented Architecture", Proceedings of the Conference on Software Maintenance and Reengineering (CSMR'06), IEEE, 2006.

[12] Z. Zhang, H. Yang, “Incubating Services in Legacy Systems for Architectural Migration”, Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC’04), IEEE, 2004.
[13] “Understanding Encodings”, http://msdn.microsoft.com/en-us/library/ms404377.aspx.
[14] D. Geer, “Will Binary XML Speed Network Traffic?”, IEEE Computer Society, April 2005.
[15] “VTD-XML: The Future of XML Processing” http://vtd-xml.sourceforge.net/.
[16] K. Chiu, M. Govindaraju, and R. Bramley, “Investigating the limits of SOAP performance for scientific computing”. In Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, pages 246-254, 2002.
[17] “XML Serializer Generator Tool' (Sgen.exe)” http://msdn.microsoft.com/en-us/library/bk3w6240.aspx.
[18] “WebSphere DataPower SOA Appliances” http://www-01.ibm.com/software/integration/datapower/.
[19] Y. Pan, Y. Zhang, K. Chiu, W. Lu, "Parallel XML Parsing Using Meta-DFAs," e-science, pp.237-244, Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007), 2007
[20] W. Lu , K. Chiu, Y. Pan, “A Parallel Approach to XML Parsing”, http://grid.cs.binghamton.edu/projects/publications/parallel-Grid06/parallel-Grid06.pdf
[21] R. D. C. , K. S. Herdy, D. Lin, “High Performance XML Parsing Using Parallel Bit Stream Technology”, http://www.cs.sfu.ca/~cameron/parabix-study-preprint.pdf
[22] http://www.ieee802.org/3/ba/PAR/par_0308.pdf
[23] http://www.ieee802.org/3/hssg/
[24] http://www.ieee802.org/3/ba/public/index.html
[25] J. D. Ambrosia, "IEEE P802.3ba Objectives", September, 2007.