ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology
Volume 1, Issue 3, May 2012

Caching and SOAP compression techniques in Service Oriented Architecture

B. Chandra Mouli,
M. Tech, Department of CSE,
Audisankara College of Engg & Technology,
Gudur, Andhrapradesh, India,
bathalamouli@gmail.com

Prof. C. Rajendra,
H.O.D, Department of CSE,
Audisankara College of Engg& Technology,
Gudur, Andhrapradesh, India,
srirajendra.c@gmail.com

Abstract— **Service Oriented Architecture (SOA) is addressed to integrate the multiple solution environments. The Web-Services plays an important role in this integration process. Performance/Scalability issues have grown to be the one of the peak concerns for the SOA environment. Many IT enterprises and research organizations introduced so many techniques to address these issues. In this paper we try to illustrate the different techniques to improve the performance of SOA. First we exposed Data Caching techniques then we exposed the SOAP compression techniques, and we exposed the impact of these two techniques in the performance of SOA. Finally we presented that Data Caching and SOAP compression techniques positive choices for the SOA environments.**

Keywords-**SOA; SOAP; data caching; SOAP- compression**;

I. INTRODUCTION

Software has turn into progressively more complex, during the history of computing. A number of approaches have been proposed to address the complexity, at different levels, such as structured programming, conceptual integrity and so on. The SOA is an architectural approach to address the complexity.

Service-oriented architecture (SOA) is mainly introduced to integrate the multiple solution environments and other words it is the advancement of distributed computing. In the early days, distributed applications communicated using proprietary protocols. Many standards have been developed over the years to reduce the costs of deployment and maintenance, with varying degrees of success. Today, the key technologies in distributed systems are service-oriented architecture (SOA). [1]

Before knowing about the *service-oriented architecture*, it is useful to review the key terms.

Architecture is the structure of a system, defining its functions, externally visible properties, and interfaces and internal components and their relationships, along with the principles governing its design, operation, and evolution [2].

A *service* is a software component that can be access via a network to provide functionality to a service client and used in a technology neutral standard form [3].

The term *service-oriented architecture* is a pattern of constructing distributed systems that transport functionality as *services*, with the loosely coupling and interacting services.

In early environments, the building of the solution was so straight forward that the task of abstracting and defining its architecture was hardly ever performed. With the rise of multi-tier applications, the variations with which applications could be delivered began to dramatically increase. IT departments started to recognize the need for a standardized definition of a baseline application that could act as a template for all others. This definition was abstract in nature, but specifically explained the technology, boundaries, rules, limitations, and design characteristics that apply to all solutions based on this template. This was the birth of the application architecture.

Application architecture is a blueprint for the application development team; different organizations may construct different applications architectures. Some may document it as high level providing abstract physical and logical representations of the technical blueprint, and others include data models, communication flow diagrams, application-wide security requirements, and aspects of infrastructure. It is not unusual for an organization to have numerous application architectures. A single architecture document typically represents a distinct solution environment. For example, an organization that houses both .NET and J2EE solutions would very likely have separate application architecture specifications for each. A key part of any application-level architecture is that it reflects immediate solution requirements, as well as long-term, strategic IT goals. It is for this reason that when multiple application architectures exist within an organization, they are almost always accompanied by and kept in alignment with governing enterprise architecture.

In larger IT organizations, the need to control and direct IT infrastructure is critical, when various, disparate application architectures co-exist and sometimes even integrate, the demands on the underlying hosting platforms can be complex and onerous. Therefore, it is necessary for create a master specification, providing a high-level overview of all forms of heterogeneity that exist within an enterprise, as well as a definition of the supporting infrastructure. An *enterprise architecture* specification is to an organization what an urban plan is to a city. Typically, changes to enterprise architectures directly affect application architectures, which is why architecture specifications often are maintained by the same group of individuals. Further, enterprise architectures often contain a long-term vision of how the organization plans to evolve its technology and environments.

Service-oriented architecture integrate both enterprise and application architectures. The main potential benefit of SOA is applying across multiple solution environments. This

All Rights Reserved © 2012 IJARCET

is where the investment in building reusable and interoperable services based on a vendor-neutral communications platform can fully be leveraged. Both the architectures are integrated by placing the service layer in between them. The service layer acts as an interface between the both the architectures.

Service is a functionality offered to satisfy the consumer's need according to a negotiated contract which includes Service Agreement. *Web Service* is a software component that can be access via a network to provide functionality to a service client and used in a technology neutral standard form. A software system designed to support interoperable machine-to-machine interaction over a network. A Web service is defined as "a standardized way of integrating Web-based applications using the XML, SOAP, WSDL, and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available, and UDDI is used for listing what services are available".

Many IT enterprises and research organizations introduced so many techniques to address these issues. In this paper we try to illustrate the different techniques to improve the performance of SOA. First we exposed Data Caching techniques then we exposed the SOAP compression techniques, and we exposed the impact of these two techniques in the performance of SOA. Finally we presented that Data Caching and SOAP compression techniques positive choices for the SOA environments.

II. CACHING TECHNIQUES

The performance of any SOA application is directly proportional to the amount of time it takes to retrieve the underlying data. Web applications are accessed by multiple users. A web site can have a heavy load on the site which can increase exponentially, which can slow down the server as well as the access of the site. Slow access is the most common problem for web sites when accessed by a large number of clients simultaneously. For resolving this problem, we can use a high level of hardware configuration, load balancer, high bandwidth, but load is not the only reason that makes a website slow, so we need to provide a kind of mechanism which will also provide fast data access and provide performance improvements. Caching provides the solution.

Data within a SOA generally falls into one of two categories:

Service state **-** This data pertains to the current state of the business process/service, i.e. where is current process instance at this point in time, which processes are active vs. closed vs. aborted, and so on. This data is particularly useful for long-running business process, and is typically stored in a database to provide insulation against machine failures.

Service Result **-** This data is delivered by the business process/data service back to the presentation tier. Typically, this data is persistent and stored in backend databases and data warehouses. Caching can play a very important role in improving the speed to access service state/result data. Caching is used to minimize the amount of traffic and latency between the service using the cache and underlying data providers. In common with any caching solution, caching design for a business service must consider issues such as: how frequently the cached data needs to be updated, whether the data is user-specific or application-wide, what mechanism to use to indicate that the cache needs updating, and so on.

It is possible to cache both types of data: *service state* and *service result.*

Service State Caching allows you to share service state data between services in a business process in memory.

Service Result Caching allows you to cache results from frequently accessed business services or data services.

Caching in a web application can be done either on the client side (client browser), in between the client and the server (proxy and reverse proxy caching), or on the server side (data caching/page output caching). So we can classify caching locations like *Client caching, Proxy caching, Reverse Proxy caching, Web Server caching.*

A. *Client Caching*

In Client Caching, the client browser performs caching by storing cached data on the local disk as a temporary file or in the browser internal memory. This provides quick access of some information which reduces the network load and the server load also. This information can't be shared by other clients so it is client specific. The Figure 1 shows the Client Caching strategy.

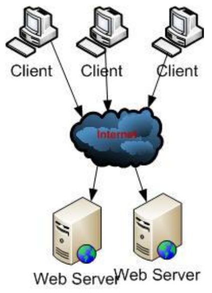

Figure 1 Client Caching

The advantages of Client Caching includes data that is cached on the local client can be easily accessed and reduces network traffic, the disadvantages of Client Caching is Cached data is totally browser dependent, so it is not shareable

B. *Proxy Caching*

The main disadvantage of client caching is data that is stored on the client browser is client specific. Proxy caching uses a dedicated server that stores caching information in between the client and the web server in a shared location so that all clients can use the same shared data. The proxy server (e.g., Microsoft Proxy Server) fulfills all the requests for the web page without sending out the request to the actual web server over the internet, resulting in faster access. Figure 2 shows the proxy caching strategy.

All Rights Reserved © 2012 IJARCET

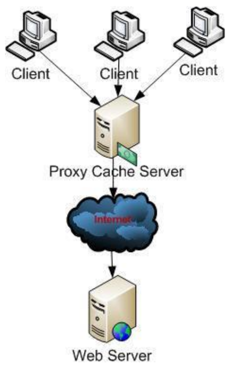

Figure 2 Proxy Caching

Proxy caches are often located near network gateways to reduce bandwidth usage. Some times multiple proxy cache servers are used for larger number of clients. This is called a cache array. The Figure 3 shows the Cache array.

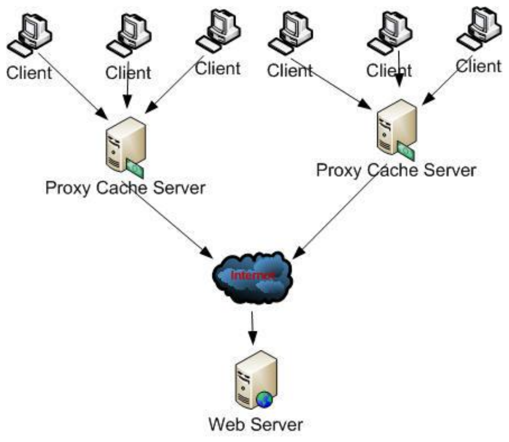

Figure 3 Cache Array

The advantages of Proxy Caching are Data that is cached on a proxy server can be accessed easily and Reduces network traffic. The disadvantage of Proxy Caching is Involves deployment and infrastructure overhead to maintain a proxy cache server

C. *Reverse Proxy Caching*

Some proxy cache servers can be placed in front of the web server to reduce the number of requests that they receive. This allows the proxy server to respond to frequently received requests and only pass other requests to the web server. This is called a reverse proxy. Figure 4 shows the Reverse Proxy Caching strategy. The advantage these caching techniques are data that is cached on a reverse proxy server can be accessed easily and reduces the number of requests. The disadvantage is as the server is configured in front of the web sever, it could increases network traffic.

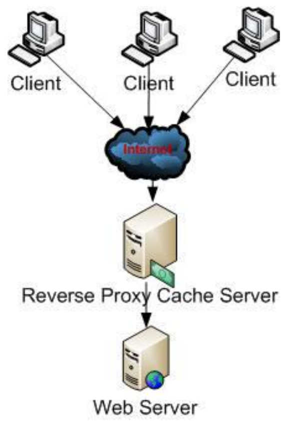

Figure 4 Reverse Proxy Caching

D. *Web Server Caching*

In web server caching, cached data is stored inside the web server. Data caching and page caching uses the web sever caching mechanism. Figure 5 shows the web server caching strategy.

The advantages of web server caching is improves the performance of sites by decreasing the round trip of data retrieval from the database or some other server. The disadvantage is increases network load. So the main advantage of Caching is to reduce server load and reduces bandwidth consumption.

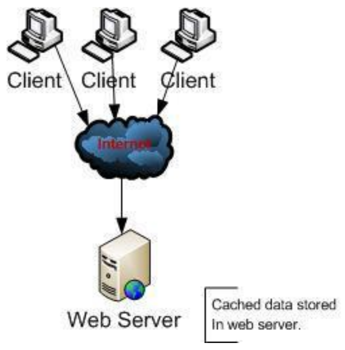

Figure 5 Web Server Caching

III. SOAP COMPRESSION

SOAP is a simple XML-based protocol to let applications exchange information over HTTP. Or more simply: SOAP is a protocol for accessing a Web Service. It is important for application development to allow Internet communication between programs. A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

In environments with limited network bandwidth or resource-constrained computing devices the high amount of protocol overhead caused by SOAP is disadvantageous. There fore, recent research work concentrated on more compact, binary representations of XML data. However, due to the special characteristics of SOAP communication most of these approaches are not applicable in the field of web services.

All Rights Reserved © 2012 IJARCET

In this section we exposed on some SOAP compression techniques.

A. *Overview of SOAP Compression*

With the growing popularity of XML more sophisticated XML compression concepts were developed. They separate the markup from the character data in a document and compress both independently with different algorithms. This technique is used by *XMill* [5] and *xmlppm* [6]. Additionally, the XML syntax rules can be exploited for data compression: in all well- formed documents the name of any end tag can be inferred from the name of the corresponding opening tag and hence, the name of an end tag can be omitted. This technique is applied by the *Fast Infoset* [7] compressor, which produces a binary serialization that is optimized with a special focus on processing speed. A most recent approach in the field of non-Schema-aware XML compressors is *Exalt* [8]. This compressor learns about typical tag sequences in the input document and stores this information in automata structures. From these automata *Exalt* can predict the next tag at a certain stage of compression and encodes only the difference between the prediction and the read value.

Another group of XML compressors is custom-tailored for selected XML languages. By knowing the vocabulary from a fixed grammar description, it is possible to create a highly specialized compressor that maps the markup-structures of this language to shorter binary tokens using fixed built-in coding tables. WBXML [9] and Millau [10] are such tools supporting various languages in the field of mobile devices like WML or SyncML. A similar approach is the *Binary Format for MPEG-7 Metadata (BiM)* [11] which is a specialized compressor for MPEG-7 Metadata.

Although such highly specialized Schema-based XML compressors exhibit very promising compression results [12], their value for practical applications is limited. Their most severe disadvantage is that they do not support the extensibility XML has been designed for. SOAP is a typical example of this problem: The SOAP body may carry all kinds of application specific data. Hence, it cannot be encoded using fixed coding tables.

B. *XML Compressors with Dynamic Schema Processing*

To overcome this limitation while still leveraging the outstanding performance of Schema-based compression, re-searchers started to develop compressors that can be customized to application specific XML grammars. If information on the document structure is available through an XML grammar description like XML Schema or DTD, two additional compression strategies become available. First, efficient binary content encodings can be inferred from the data type definitions in the grammar. For example, all numeric values in the XML document could be represented as 32 bit integer numbers instead of using verbose text representations. Second, the grammar prescribes how valid instance documents are structured. Therefore, parts of the structure in-formation can be omitted in the instance document. Of course, the Schema information is also required for reconstructing the original message from the compressed representation.

Although the *XGrind* compressor [13] employs the cor-responding DTD when processing an XML file, it neither omits tags that can be inferred from the grammar nor uses efficient binary encodings for numeric content like `xsd:int` or `xsd:DateTime` etc. The reason for this is that it focuses on so called context-free compression, i.e. XGrind allows for parsing and querying selected parts of the binary encoded file without decompressing it.

Anyhow, even under the constraint of context-free com-pression XGrind implements some features for generating compact XML representations. It uses shorter identifiers that represent the text values of tag and attribute names: It uses the DTD to identify all possible tag names, which are then mapped to compact 8 bit identifiers in the binary encoding. Additionally, the names of closing tags are omitted since in well-formed XML documents these can be inferred from the name of the corresponding opening tag. The possible values of enumeration types are also binary encoded using binary block codes. All other content is Huffman encoded in a per-tag-name fashion, i.e. for each element with a certain name a separate Huffman table is maintained.

A second approach for XML compression with optional dynamic grammar support is *Xebu* [14]. It encodes the sequence of SAX events generated by a parser. Again, on their first appearance tag names are indexed with a byte value which is then used as a short-hand pointer on repetition. Unlike XGrind, the Xebu encoder can detect numeric data and encodes it in a more compact binary format.

These two features of Xebu are also available if no grammar is present. In addition, so called omission automata can be generated from a Relax NG [15] grammar. These can then be used to omit the encoding of SAX events which can be inferred from the grammar on decoding. This approach increases com- pression efficiency, e.g. if the grammar prescribes a sequence with each element occurring exactly once. Unfortunately, the paper does not describe how these automata are generated from the grammar. Additionally, the Relax NG grammar can also be used for setting up the coding tables with initial values, since possible tag and attribute names are specified in the grammar. This feature is called pre-caching.

Another Schema-aware XML processor is *XML Xpress* [14]. It employs a two step processing scheme. In a first step a so called Schema Model file is generated offline from an XML Schema document and a set of sample XML files that represent typical data the compressor will be exposed to later on. Step two comprises the actual compression process and takes the Schema Model file and the XML source file to be compressed as input. The decompressor also uses the Schema Model file in order to decode the compressed representation. Because

Differential Encoding presented in [12]. The authors employ a differential encoding algorithm to achieve compact SOAP message representations. Instead of sending the entire SOAP message only the difference between the message and a skele- ton, which is previously generated from the WSDL service description, is transmitted. There are many data formats for de- scribing differences between two XML documents. One very efficient solution is to use the

All Rights Reserved © 2012 IJARCET

Document Update Language (DUL) in combination with the xmlppm compressor [6].

C. ZippedSOAP

The entire SOAP transmissions are depends on the serialization to de-serialization, and de-serialization to serialization. Where the *ZippedSOAP* compress the SOAP message before the SOAP request transmits via internet and SOAP response transmits via internet. The zippedSOAP structure is described in Figure 6. The diagram shows when the compression will take place [4].

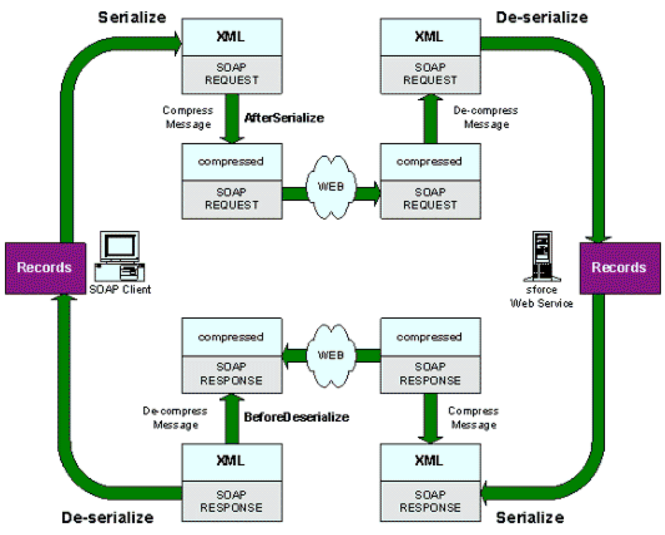

Figure 6 Zipped Soap process

IV. Conclusion

The main potential benefit of Service Oriented Architecture (SOA) is integrating the multiple solution environments. To integrate the multiple solution environments Web Services please an important role. Performance/Scalability issues have grown to be the one of the peak concerns for the SOA environment. Many IT enterprises and research organizations introduced so many techniques to address these issues. We exposed all the available caching and SOAP compression techniques to improve performance of the SOA.

References

[1] An Overview of Service-Oriented Architecture, Web Service and Grid Computing by Latha Srinivasan and Jem Treadwll HP Software Global Business Unit (Nov, 2005).
[2] Software Architecture in Practice, Second Edition by Len Bass, Paul Clemgents, Rick Kazman.
[3] Service Oriented Architecture: Concepts,, Technology, and Design by Thomas Erl.
[4] SOAPCompression, http://wiki.developerforce.com/page/SOAP_Compression
[5] H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, Texas, USA, 2000, pp. 153–164.
[6] J. Cheney, "Compressing XML with multiplexed hierarchical PPM models," in *Data Compression Conference*, 2001, pp. 163–173. [Online]. Available: http://citeseer.ist.psu.edu/cheney01compressing.html
[7] P. Sandoz, A. Triglia, and S. Pericas-Geertsen, "Fast infoset," jun 2004. [Online].Available: http://java.sun.com/developer/technicalArticles/xml/ fastinfoset/
[8] V. Toman, "Syntactical compression of XML data," in *Proceedings of the International Conference on Advanced Information Systems Engi- neering*, Riga, Latvia, June 2004.
[9] [Online]. Available: http://wbxmllib.sourceforge.net/
[10] M. Girardot and N. Sundaresan, "Millau: An encoding format for effcient representation and exchange of XML over the web," in *9th International World Wide Web Conference*, Amsterdam, Netherlands, May 2000, pp. 747–765.
[11] U. Niedermeier, J. Heuer, A. Hutter, W. Stechele, and A. Kaup, "An MPEG-7 tool for compression and streaming of XML data," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, Lusanne, Switzerland, Aug. 2002, pp. 521–524.
[12] C. Werner, C. Buschmann, and S. Fischer, "WSDL-Driven SOAP Compression," *International Journal of Web Services Research*, vol. 2, no. 1, 2005.
[13] P. Tolani and J. R. Haritsa, "XGRIND: A query-friendly XML compres- sor," in *Proceedings of the International Conference on Data Engineer- ing*, San Jose, California, USA, Feb. 2002, pp. 225–234.
[14] J. Kangasharju, S. Tarkoma, and T. Lindholm, "Xebu: A binary format with schema-based optimizations for xml data." in *Proceedings of the International Conference on Web Information Systems Engeneering*, New York City, New York, USA, Nov. 2005, pp. 528–535.
[15] J. Clark and M. Makoto, "Definitive specification for RELAX NG using the XML syntax," Dec. 2001. [Online]. Available: http://www.relaxng.org/spec-20011203.html

All Rights Reserved © 2012 IJARCET