International Journal of Recent Trends in Engineering & Research
ISSN(ONLINE) : 2455-1457
IMPACT FACTOR : 3.344

A Performance Model for the Measurement of Web Service Framework

Venkatesh.G[1] Basi Reddy.A[2],S.Siva Prasad[3]
[1,3] *Department of IT, Sree Vidyanikethan Engineering College, Tirupati, India*
[2] *Department of CSE,S.V College of Engineering, Tirupati, India*

Abstract— Web services are the main implementation technology of the service oriented architecture (SOA). Web services exchange SOAP XML messages, thus they provide a true platform and language independent distributed communication. SOAP XML messages burden the communication with a significant serialization and deserialization overhead which can even be comparable with the execution time of the service's application logic itself. Web services are built on XML.It use XML for describing the service interface, message exchange and the WS- protocols that provide addressing, security and reliable messaging. When the interface of a web service is being designed, it is important to find the right granularity for the parameters, i.e., the most reusable interface with the best response times. To determine the best response times, it is essential to have the capability to predict the estimated response time based on the interface and on its expected usage of the service. This problem inspired us to examine the response time overhead of the various frameworks implementing the web service stack, and to give a performance model equipped with performance prediction capability for web services. I proposed a performance model with which the response time overhead of web services with arbitrary interfaces can be predicted if the coefficients of the model are calculated from some simple measurements for the given set of test cases. It shows the measurement results for two web service frameworks and also gives a detailed description of the performance model.

Keywords— *Performance model;Web services;WSDL;SOAP; WS-* protocols.*

I. INTRODUCTION

Services are intended to be independent building blocks that collectively represent an application environment. Services have a number of unique characteristics that allow them to participate as part of a service-oriented architecture (SOA). An SOA is a design model with a deeply rooted concept of encapsulating application logic within services that interact via a common communications protocol. When Web services are used to establish this communications framework, they basically represent a Web-based implementation of an SOA.

The most widely accepted and successful type of service is the XML Web service. From here on referred to as Web service or, simply, service. This type of service has two fundamental requirements: 1) it communicates via Internet protocols i.e most commonly HTTP. 2) It sends and receives data formatted as XML documents. This XML-based messaging format established a transmission framework for inter-application (or inter-service) communication via HTTP. SOAP provided an attractive alternative to traditional proprietary protocols, such as CORBA and DCOM.

Web services describes 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. It is used primarily as a means for businesses to communicate with each other and with clients. Web services allow organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall. Unlike

traditional client/server models, such as a Web server/Web page system, Web services do not provide the user with a GUI. Web services instead share business logic, data and processes through a programmatic interface across a network.

The applications interface, not the users. Developers can then add the Web service to a GUI such as a Web page or an executable program to offer specific functionality to users. Web services allow different applications from different sources to communicate with each other without time consuming custom coding, and because all communication is in XML, Web services are not tied to any one operating system or programming language. For example, Java can talk with .Net, Windows applications can talk with UNIX applications.

Web services exchange SOAP XML messages, thus they provide true platform and language independent distributed communication. However ,this interoperability come at a price: SOAP XML messages burden the communication with a significant serialization and deserialization overhead which can even be comparable with the execution time of the service's application logic itself. This burdens the communication with a large response time overhead compared to binary distributed communications like CORBA or RMI. Therefore, it is important to know how to design the interface of a web service so we can minimize the communication overhead.

When the interface of a web service is being designed, it is important to find the right granularity for the parameters, i.e., the most reusable interface with the best response times. To determine the best response times, it is essential to have the capability to predict the estimated response time based on the interface and on its expected usage of the service. This problem inspired us to examine the response time overhead of the various web service frameworks implementing the web service stack, and to give a performance model equipped with performance prediction capability for web services.
A dataset is used to measure the response time overhead of web services. The dataset cover the most commonly used primitive types, their combinations into arrays and structured types and even the most widespread WS-*protocols including WS-ReliableMessaging, WS-Security and WS-SecureConversation.

A dataset is implemented for the web service frameworks and also made performance measurement. Our measurement show that the web service frameworks have the same performance characteristics therefore, it is possible to make a common performance model which can be used to approximate the measurement results and even be used to make predictions on other services with other interfaces.
Proposed a performance model with which the response time overhead of web services with arbitrary interfaces can be predicted if the coefficients of the model are calculated from some simple measurement for the given dataset. The measurements results for the web service frameworks and also give a detailed description of the performance model.

II. RELATED WORK

This section summarizes the related works for a performance prediction of web services. However, to my knowledge, there has not yet been any propositions of predicting the performance of web services based on their interface descriptions.

M. Novakouski et. al. 1 examined how different security mechanisms in web services influence the response time and other resource usages. They identified the key tradeoffs between different security options and their results were that security mechanisms for web services have a considerable impact on the response time. They only tested the Apache Axis2 Rampart framework and no other frameworks, and they concluded that the results rely heavily on the execution environment. My proposition takes into account this result, and provides a solution that can be customized to specific environments.

D. Rodrigues et. al. 2 also analyzed the security mechanisms for web services. They also tried different algorithms with different configurations. They even called the services from multiple clients, which resulted in expected performance degradation. They also used the Apache Axis2 Rampart framework, and their results are similar to the work of M. Novakousky et. al.

M. B. Juric et. al. 3 Compared web services security with other distributed communication techniques, such as RMI and RMI over SSL. Their conclusion is the same as in the previous two related works: WS-Security has a huge overhead. Earlier they also compared the performance of CORBA and RMI in 4. Although they identified the factors contributing to the response time overhead, they did not provide a performance model for the communication that could be used for performance prediction.

S. Shirasuna et. al. 5 compared security mechanisms for grid services. Their results are similar: SSL is the fastest, although, it does not provide end-to-end security; WS-Security is faster than WS-SecureConversation for one-time invocations; WS-SecureConversation has better response times for multiple calls, however, it has a huge disadvantage, too, since the server has to hold the connection status information for each client, and this can lead to serious scalability issues if the number of clients increases.

R. A. van Engelen and W. Zhang et. al. 6 Evaluated some optimizations regarding the security algorithms used in web services security using the gSOAP framework. They provided some guidelines, how better performance can be achieved. They made measurements also for large messages. However they did not provide a performance model.

L. Cheung et. al. 7 Proposed a framework based on queuing models for performance prediction of third party web services. Their work is similar to mine, since it builds on measuring the response time of the target service including the communication overhead and the actual time of the execution of the server side logic. However, my aim is to predict the communication overhead based on the interface of an arbitrary service, even if the service is not yet implemented.

Y. Leu et. al. 8 Developed a performance model also based on queuing models for analyzing and predicting the performance of service applications composed on a COTS ESB platform. Their work focuses on ESB operations, such as routing and composition, and not on predicting performance based on the interface of the services.

H. H. Liu and P. V. Crain et. al. 9 proposed a performance model for web service implementations. They built a queuing model for the client side, the network, the web service and for the database behind the service. Their observation is that most of the response time comes from the serialization of the SOAP messages. Therefore, it is important to be able to measure and predict this overhead more accurately, and my work focuses on this problem.

S. Oh and G. C. Fox et. al. 10 also identified this problem, and they proposed their own framework called Handheld Flexible Representation (HHFR) to be used in mobile computing instead of SOAP. They also made measurements to prove that their solution out performs SOAP.

M.Tian et. al. 11Also examined mobile web services and suggested a solution in which the clients can ask the server to compress the SOAP messages. They showed that compression is useful for poorly connected clients with resource-constraint devices despite the CPU time required for decompressing the responses. However, if standard SOAP communication is used in mobile environments, it is important to know, how the clients and services will perform. My proposition can also be generalized to this area.

G. Imre et. al. 12 Developed a cost model for XML serialization in Java and .NET. XML serialization is the heart of SOAP message serialization, therefore, it is an important contribution in this area, although, they only examined three primitive types (string, int and double). My measurements show that other types and the WS-* protocols introduce other contributions to the SOAP message serialization and the response time overhead cannot be predicted merely based on the XML serialization overhead.

Have not found any related work that gives such an extensive examination of the response time overhead of web services as mine. My aim is to be able to predict the response time overhead of a web service based on its interface if the characteristics of the hosting server is known. These characteristics can be measured by the services and clients and introduce services and clients and also the performance model which can be used for predicting the response time overhead of web services based on their interfaces.

III. WEB SERVICE STACK IMPLEMENTATION ARCHITECTURE

The most widely implemented WS-* protocols are WS-Addressing, WS-Reliable Messaging, WS-Security and WS-SecureConversation.The Microsoft .NET Framework, the GlassFish server, the Oracle WebLogic server, the IBM WebSphere server and the Apache CXF framework support most of them. The difference between these frameworks is in the configuration of these protocols. There are many other WS-* protocols. However, the most widely used SOA products do not implement them, therefore, they are also omitted from this paper.

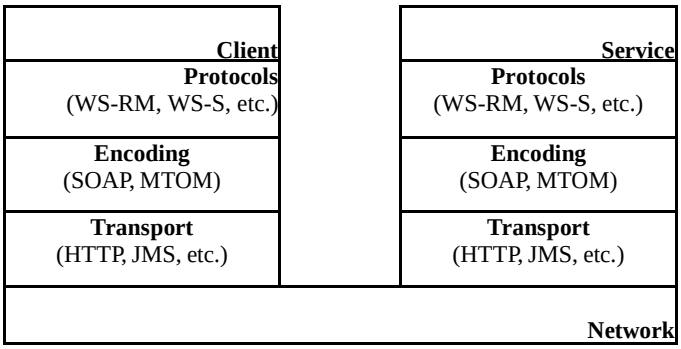

Fig. 1. Typical architecture of the web service stack implementations.

Fig. 1 shows the general architecture of the web service stack implementations. The Windows Communication Foundation (WCF) 13 stack (part of the Microsoft.NET framework) and the Metro 14 stack (the reference implementation of JAX- WS) follow this structure, and other framework implementations can also be modeled this way.

The network is used for sending bytes from one end to the other. The network protocol for web services is usually HTTP, but it can be replaced with other protocols.

A. Transport layer

The transport layer is responsible for handling the network protocol. On the service side it waits for client connections, on the client side it connects to services. It also transfers bytes between the two participants.

B. Encoding Layer

The encoding layer translates between bytes and a framework specific message object representation, i.e. it is responsible for serialization and deserialization (e.g. into SOAP, with or without MTOM). The transport and encoding layers are always mandatory.

C. *Protocol Layer*

The protocol layers are optional, and they implement the various WS-* standards (e.g. WS-Reliable Messaging, WS-Security, etc.). The protocol layers usually produce bootstrap messages or insert additional headers into service invocation messages. For example, the WS-ReliableMessaging protocol includes bootstrap messages for initiating and terminating the reliable session. WS-ReliableMessaging also extends the messages with additional SOAP headers containing acknowledgement information about the messages already received by the parties. WS-SecureConversation also has bootstrap messages for establishing the security context and the session key. WS-Security and WS-SecureConversation have additional security headers in the messages for timestamps, digital signatures and encrypted keys.

This section enumerates the most important web service stack implementations in Java and in .NET. Table I shows the exact versions of the implementations which are examined and these are the versions which are supported by the generator of the web service framework.

The descriptions in this section focus on the configuration properties of the web service frameworks, since configuring the WS-* protocols are usually the most challenging part of the development. The WS-Security protocols have the most complex configuration, because the WS-Policy standards do not specify how X.509 certificates can be set for the services and for the clients, and so the different software vendors came up with different ways to configure these certificates.

This section also shows that even if standard WS-Policy assertions are used, the different configuration solutions of the different software vendors may result in interoperability issues. In addition, the configuration solutions can be very complex and hard to maintain. These are clear indicators that an easy to use platform independent domain specific language with the appropriate level of abstraction is required for the configuration of the various WS-* protocols.

TABLE I. ***WEB SERVICE STACK IMPLEMENTATIONS***

Vendor	**Framework**	**Server**	**Dev. Environment**	**API**	**Config uration**
Microsoft	WCF .NET4.5	IIS 8.5	Visual Studio 2013	WCF	Web. Config/ App. Config
Oracle	Metro 2.3	GlassFish 4.0	Netbeans 8.0	JAX-WS	WS-Policy

IV. OVERVIEW OF THE PERFORMANCE PREDICTION METHOD

The major performance overhead of web services results from the use of XML for serializing messages. The WS-* protocols extend simple SOAP messages with additional headers, bootstrap messages, and even XML encryption and digital signatures can burden the communication further. In this paper, proposed a performance model for web services in order to be able to predict the response time overhead of web service calls for services with arbitrary interfaces. The coefficients of the performance model have to be calculated in advance so that the performance model can be used for prediction.

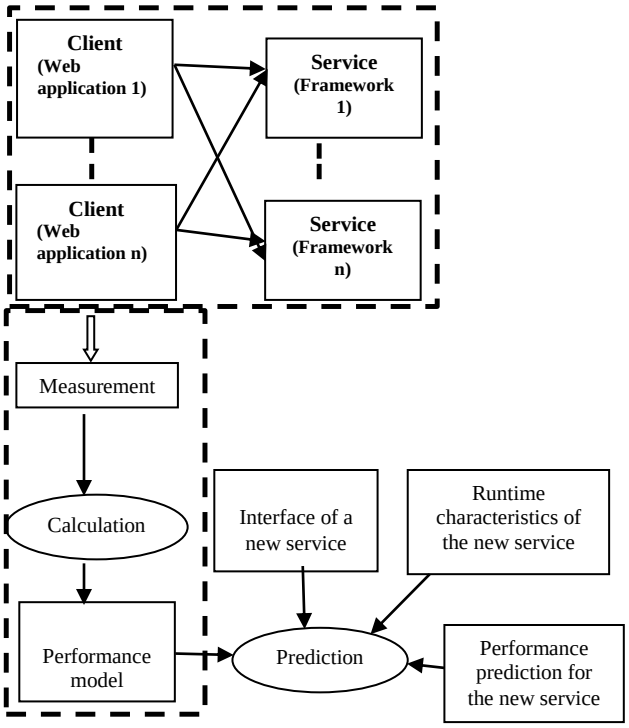

Fig. 2. Performance prediction process overview

The proposed method is used to calculate these coefficients, see Fig. 2.This method requires the execution of some measurements between the various frameworks by using a predefined set of services and clients. These are called reference services and reference clients. The interface of the reference services defines operations, where the input and output parameters are arrays of structures containing fields of various types. These types are the most common primitive types used in programming languages. The reference services differ only in the enabled WS-* protocols. All the reference services and all their respective clients have to be implemented in every web service framework. Then all the clients have to call their respective services, even between different frameworks, and the response times have to be measured. The measurements have to be performed with different array lengths and with different number of calls.

My measurements show that different environments have the same characteristics: the response time is in linear correlation with the array length and also with the number of calls. Hence, a common performance model can be developed for the different environments. If this performance model is used, the environments only differ in the coefficients of the model. These coefficients can be calculated from the measurements performed between the reference clients and reference services. The response time overhead of a service with an arbitrary interface can be predicated based on the model if the model's coefficients and the runtime characteristics of the Service is known (e.g. the number of calls, the lengths of the arrays).

V. MEASUREMENT RESULTS

The measurements were performed on a single computer using local access between the clients and the services. The computer had the following configuration:

- Intel Pentium Dual-Core 2.30GHz CPU
- 4GB RAM
- Microsoft Windows 7 Professional SP1 64-bit
- Oracle JDK7(1.7.0) 64-bit
- GlassFish Server 4.0 with Metro 2.3
- Netbeans IDE 8.0

In order to measure the combined effect of the factors of the different layers of the web service stack implementations introduced in section III, a number of service and client have to be defined and implemented.

The most common basic types in programming languages and in XML are: byte, boolean, int, long, float, double, string, DateTime and TimeSpan. Restrictions can also be defined in XML Schema for these types, however, apart from enumerations these restrictions are not supported by the JAX-WS API. XML Schema includes other data types, but they are only restrictions or subsets of the types listed here. Therefore, we can assume that the listed types are the basic building blocks of composite types. Composite types are usually either arrays or structured types. The following SOAL code describes the interface of the service that can be used for performance measurements:

```
struct TrainSchedule {
string Name;
string  Track;
string StartStation;
string  EndStation;
 long Distance;
DateTime OriginalSchedule;
DateTime  ExpectedSchedule;
string[]  Services;
}
interface IWsPerfMav  {
TrainSchedule[] GetTrainSchedule
(DateTime  fromTime, DateTime toTime);
}
```

Here for implementation I am using JAX-WS with SOAP Protocol supporting and calculating response time of input parameter TIMESTAMP and STRING.

The document/wrapped SOAP encoding style was used to implement the service, since the JAX-WS 15 implementations use this style by default. The types were mapped to XSD. The service and the client were generated using the SOA modeling framework 16 17. The dataset are sufficient for measuring the overhead of the transport and encoding layers. However, for the protocol layers, the various WS-* standards had to be enabled.

The Service was implemented as part of a single web application, and was deployed to the respective application servers (GlassFish for Metro). Server was using their default settings except for the following:

• On GlassFish, the monitoring of web services was turned off so that it would not affect the performance. In addition, the virtual memory of the JVM was increased to 8 GB, since the deployment of the service required 1.5 GB of memory, and the memory was still leaking, so the server had to be restarted daily.

Since the overhead of web service calls results mainly from the XML serialization, the selected XML parser may have an impact on the observed response times. This is especially important in the Java world, where the selected JAXP (Java API for XML Processing) API may have multiple implementations. The default XML parser for the GlassFish server is a StAX (Streaming API for XML) implementation, called SJSXP (Sun Java Streaming XML Parser). However, as my

measurements show, the overhead of web service calls relies also on other factors (e.g. data types), too. StAX is a low level parser, and it does not deal with data types. Data types are handled by the higher level data binding performed by JAXB (Java Architecture for XML Binding). Therefore, my goal is not to compare the various parsers with each other. My aim is to find a model based on which these XML parsers can be evaluated against each other.

The client was implemented as simple standalone console applications. The measurements were made from the client side with timers of at least millisecond precision (System.nanoTime in Java).

Fig.3 shows the general diagram for the SOAP Request and Response between service requestor and service provider.

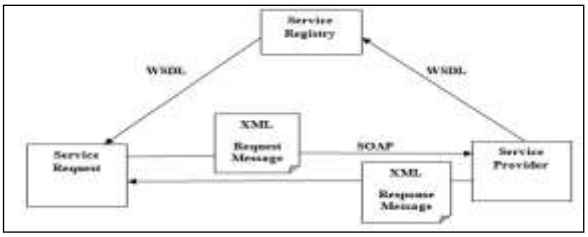

Fig. 3. SOAP messaging between service requestor and service provider

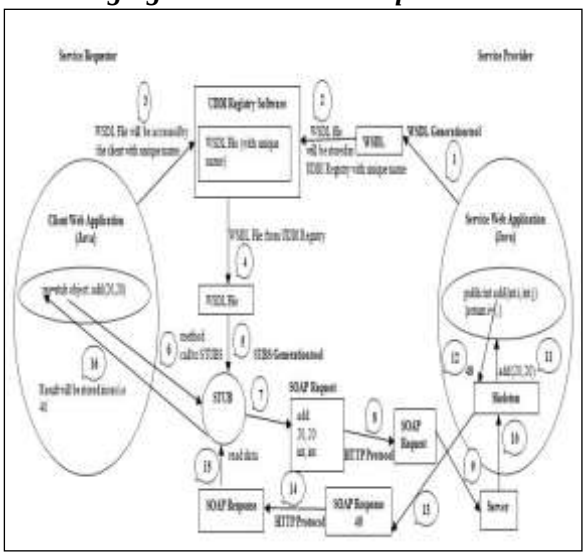

Fig. 4. SOAP Request and SOAP Response between Client application and service application

Web services does not provide APIs, it provide specifications that means set of rules and guidelines how I will communicate with two web applications.

To interact with two interoperable web applications or two web service frameworks or web application and web service framework either it may be developed in two different programming languages or same programming language web applications.

Web services provide six components:

- WSDL
- UDDI
- SKELETON
- STUB
- SOAP Protocol
- .HTTP Protocol

Out of these six components UDDI is optional and remaining all are mandatory.

Fig. 4 shows the Service application must be developed with web services web application that is developed in different programming language like java, .NET.

Client application may be a standalone application or web application that is developed in different programming language like java, .NET.

To interact with client (standalone web application developed in java) and service application (web services web application developed in java), i have followed these steps:

Client application want to invoke add() method in CalService.java that is in service application. Client application should know about name of the class ,method name, parameter types, return types then only client application can invoke add() method from that calservice.java that is in service application

Step 1: Service application need to share that service class details in the form of one format, that format should be understandable by the client application (that may be developed in java, .NET).service application will share that information in the form of XML.this XML file contains the name of the class, method name, parameter types and return types. This information will share to the client application who want to invoke this add () method in CalService.java that is service application.

This XML file should be understood by different programming languages like java, .NET. This XML file is known as WSDL File (web services description language).WSDL file describes about our service class that is name of the class, method name, parameter types and return types, in addition to that it will have URL i.e Endpoint URL, Endpoint URL means the location where our services are running, Endpoint URL is also stored in WSDL file.this information is enough for the client application to invoke service application.

WSDL file will be automatically generated by the WSDL generation tool.WSDL file is used by the client application to know about the service application details.

Step 2: Service application will share WSDL file to client application with the help of UDDI Registry Software.UDDI Registry Software will store WSDL file with some unique name, that unique name will share to the client application.

Step 3: Client application will send the location of the UDDI Registry Software. It will receive WSDL file i.e client application will interact with UDDI Registry Software by sending unique name and with that unique name client application will get the WSDL file.

Step 4: Client application will get the WSDL file.

Step 5: Using WSDL file, client application will be generated some classes called STUBS or Proxies. i.e how client application will generate stubs means by using stub generation tool. If client application is developed in java programming language then that STUBS must be a java programming language or if client application is developed in .NET programming language that STUBS must be a .NET programming language .whatever the methods are available in CalService.java that is in Service application,the same methods are available in STUBS but the implementation is different in STUBS as well as CalService.java that is in Service application.

Step 6: Client application will create STUB Object with dot (.) of the method and results will be stored. i.e res=StubObject.add(); The method call(res=StubObject.add();) is going to STUBS ,now

the STUBS is having method details like name of the class, method name, parameter types and return types.

Step 7: STUBS will prepare XML file and store that details in the form of XML file, this XML file is known as SOAP Request. Now the SOAP XML Request will have name of the method, parameter values and parameter types. i.e SOAP Request is going to send to the service application, may be service application is developed in different programming language like java,.NET and service application need to understand the method that why it will store these details in SOAP XML Request. STUBS use some set of predefined tags to prepare the XML file. These predefined tags are called as SOAP Tags. So STUBS uses SOAP tags and prepare a SOAP Request.

Step 8: SOAP Request will send to service application with the help of HTTP Protocol, now it is having name of the method, parameter values and parameter types.

Step 9: SOAP Request is receiving to the server.

Step 10: Now the server will handover SOAP Request to the SKELETON, SKELETON is a predefined class. If Service application is developed in java programming language then SKELETON should be a java class or Service application is developed in .NET programming language then SKELETON should be a .Net class. SKELETON will take SOAP Request and read the XML file to get the method name, parameter values, and parameter types.

Step 11: SKELETON will invoke the client application requested method details by passing Parameter values i.e (add (20, 20)) and need to invoke a method i.e (add (20, 20)) on CalService.java that is in service application.

Step 12: SKELETON will get return value i.e (40) and that value need to send to the client application.

Step13: SKELETON will prepare one XML file that XML file is known as a SOAP Response. In this SOAP Response it will store the service application returned value i.e (40).SOAP Response is an XML file, so it can be understandable and read to any programming language like java,.NET. SKELETON uses some set of predefined tags from the SOAP to prepare the SOAP Response .SOAP Response contain the SOAP Tags.

Step 14: The SOAP Response is available at the service application side and sending the SOAP Response to the client application with the help of HTTP Protocol.

Step 15: Now the SOAP Response is available at the client application side which contains the returned value.i.e (40).and returned value is read by the STUBS. This SOAP Response will read the STUBS and STUBS will get the return value i.e (40)

Step 16: STUBS will get the return value i.e (40) and that returned value is hand over to the Client application.

On the service side SKELETON (predefined class) and WSDL Generation tool (predefined class) is mandatory and it will be provided by the programming language i.e java means Java API and.NET means .NET API.on the client side STUBS Generation tool (predefined class) is mandatory and it will be provided by the programming language i.e java means Java API and.NET means .NET API.

After executing the client standalone application and service web application, the following results will be generated.

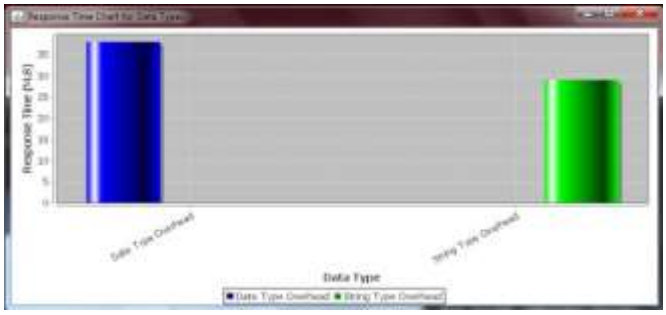

Fig. 5. Response time chart for the Data Types(Timestamp,String)

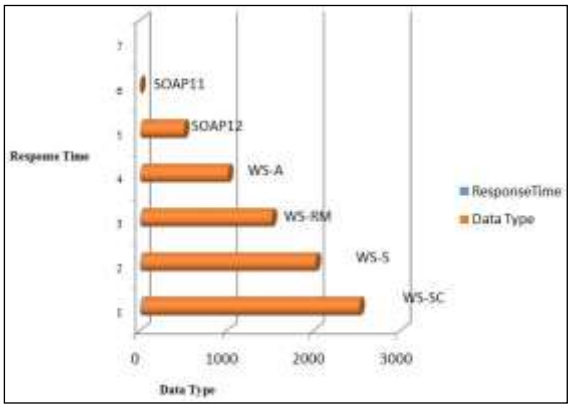

Fig. 6. Response time chart for protocols SOAP11,SOAP12,WS-A,WS-RM,WS S,WS-SC

The Fig. 6 summaries the results for different protocols SOAP11,SOAP12,WS-Adressing,WS-Reliable Messaging does not effect the response time for different data types ,but WS-S,WS-SC effect the response time.

TABLE II. MEASURED AND CALCULATED VALUES FOR WEB SERVICE FRAMEWORK (WEB APPLICATION) AND STANDALONE APPLICATION.

Client	Service	Protocols	Calculated	Measured
Standalone Application	Metro (Web Application)	SOAP11	0.79	0.85
		SOAP12	0.89	0.75
		WS-A	0.91	0.56
		WS-RM	2.95	1.45
		WS-S	14.34	13.65
		WS-SC	6.35	5.15
Metro (Web Application)	Standalone Application	SOAP11	0.45	0.55
		SOAP12	0.75	0.95
		WS-A	0.96	0.91
		WS-RM	2.45	1.95
		WS-S	13.34	12.23
		WS-SC	8.35	7.65

VI. CONCLUSION

Web services have a high communication overhead. If strict QoS requirements have to be met, it is useful to have a design time prediction capability about this communication overhead to be able to find the right granularity of the service interface. This challenge is solved by the proposed performance prediction framework for web services. The framework can be applied to a wider range of distributed systems, and not only for web services. Component systems, RESTful web services and future distributed communication technologies can be examined.

In the future we are planning to measure the performance of RESTful web services, although RESTful services may use other serialization methods (e.g., JSON) instead of XML and so they require other considerations.

REFERENCES

1. M. Novakouski, S. Simanta, G. Peterson, E. Morris, and G. Lewis. "Performance Analysis of WS-Security Mechanisms in SOAP-Based Web Services,"Nov. 2010.
2. D. Rodrigues, J. C. Estrella, and K. R. L. J. C. Branco, "Analysis of security and performance aspects in service-oriented architectures," Int. J. Security Appl., vol. 5, no. 1, pp. 13–30, Jan. 2011.
3. M. B. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko, "Comparison of performance of web services, WS-security, RMI, and RMI-SSL," J. Syst. Softw., vol. 79, no. 5, pp. 689–700, May. 2006.
4. M. B. Juric, I. Rozman, and M. Hericko."Performance comparison of CORBA and RMI," Inform. Softw. Technol. 42(13), pp. 915–933,May. 2000.
5. S. Shirasuna, A. Slominski, L. Fang, and D. Gannon, "Performance comparison of security mechanisms for grid services," in Proc. 5th IEEE/ACM Int. Workshop Grid Comput., pp.360–364,2004.
6. R. A. van Engelen and W. Zhang, "An overview and evaluation of web services security performance optimizations," in Proc. IEEE Int. Conf. Web Serv., pp. 137–144,2008.
7. L. Cheung, L. Golubchik, and F. Sha, "A study of web services performance prediction: A client's perspective," in Proc. IEEE 19th Annu. Int. Symp. Model., Anal., Simulation Comput. Telecommun.Syst., pp. 75–84,2011.
8. Y. Liu, I. Gorton, and L. Zhu, "Performance prediction of serviceoriented applications based on an enterprise service bus," in Proc. 31st Annu. Int. Comput. Softw. Appl. Conf. - Vol. 01,pp. 327–334,2007.
9. H. H. Liu and P. V. Crain."An analytic model for predicting the performance of SOA-based enterprise software applications,"Proc. Int. CMG Conf., pp. 821–832,2004.
10. S. Oh, and G. C. Fox, "Optimizing web service messaging performance in mobile computing," Future Gener. Comput. Syst., vol. 23,no. 4, pp. 623–632, May 2007.
11. M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller,"Performance considerations for mobile web services," Comput.Commun., vol. 27, no. 11, pp. 1097–1105, Jul. 2004.
12. G. Imre, M. Kaszo_, T. Levendovszky, and H. Charaf, "A novel cost model of XML serialization," Electron. Notes Theor. Comput. Sci.,vol. 261, pp. 147–162, Feb. 2010.
13. Microsoft, Windows Communication Foundation [Online]. Available: http://msdn.microsoft.com/enus/netframework/aa663324.aspx ,Accessed on 15 Jan. 2014.
14. Oracle, Metro [Online]. Available: http://metro.java.net/ ,Accessed on 15 Jan. 2014.
15. Oracle, JSR 224: Java API for XML-Based Web Services (JAX-WS)2.0 [Online].Available:http://jcp.org/en/jsr/detail?id=224 , Accessed on 15 Jan. 2014.
16. B. Simon, B. Goldschmidt, and K. Kondorosi, " A metamodel for the web services standards," J. Grid Comput., vol. 11, no. 4,pp. 735–752, 2013.
17. B. Simon and B. Goldschmidt, "A human readable platform independent domain specific language for WSDL," in Proc. 2nd Int. Conf. Netw. Digit. Technol., pp. 529–536,2010.

