Optimizing Web Service Messaging Performance Using a Context Store for Static Data

Sangyoon Oh[(1,2)*], Mehmet S. Aktas[(1,2)*], Marlon Pierce[(1)], Geoffrey C. Fox[(1,2)]
(1) Community Grids Lab, Indiana University, Bloomington, Indiana, 47404, USA
(2) Computer Science Department, School of Informatics, Indiana University
{ ohsangy, maktas, mpierce, gcf }@cs.indiana.edu
* Corresponding Authors

Abstract: - The performance and efficiency of Web Service messaging can be greatly increased by removing the redundant parts of SOAP messages. This paper describes our research work in optimizing SOAP message contents. This area is particularly important to those applications that are physically constrained mobile computing environments. The redundant or static parts of the SOAP message may be treated as metadata and stored in shared metadata space. We integrate our optimized SOAP messaging system with our information management research framework. We evaluate our approach by testing the performance of the resulting system. The empirical result shows that we save on average 83% of message size and on average 41% of transit time.

Key-Words: - Grid/Web Service, WS-Context, Mobile Applications, Information Systems, Quality of Service

1 Introduction

The SOAP message enables applications on heterogeneous platforms interoperate with each other by defining text-based remote procedure call (RPC) mechanism. However, the verbose nature of a SOAP message holds potential overheads. For example, when data is converted to and from a SOAP message, both size and processing time of the message is increased substantially. This creates performance inefficiencies in some application domains, such as mobile computing. The mobile computing environment, which holds many physical constraints like limitations in processing power, battery life, and wireless connections, needs an efficient solution to the problem of expensive processing cost of SOAP messages. We proposed and implemented a research framework, which is designed to provide efficient and optimized message exchange paradigm in mobile Web Service environment, the Handheld Flexible Representation (HHFR) [1] [2]. By using the HHFR architecture, participating applications can a) exchange messages in flexible presentation, such as binary format, and b) optimize message contents by removing the redundant parts of the SOAP message.

In this paper, we discuss the use of a Context-store, which is a meta-data repository for HHFR, and present our particular investigations and experiences of using information management research framework, Fault Tolerant High Performance Information Service (FTHPIS) [3] [4], as the Context-store. In this research, the redundant message parts of the SOAP message, which is exchanged between participants, are treated as metadata and stored in the Context-store. This way, the size of exchanging SOAP messages is being minimized. The redundant parts of a SOAP message can be considered as XML fragments, which are encoded in every SOAP message. These XML elements are stored as context, i.e. metadata associated to a conversation, into the Context-store. Each context is referred with an URI. The uniqueness of the URI is ensured by the system. Upon receiving the SOAP message, the corresponding parties may interact with the WS-Context Specification [5] compliant Context-store to retrieve the context associated with the URIs in the SOAP message. In addition to minimizing message size, the use of the Context-store guarantees integrity of exchanging messages. For example, a late-joined participant should access and retrieve stored negotiated information from the Context-store to understand an ongoing conversation between a service and other participants. Our presentation is particularly focusing on applications in mobile computing environments, but the approach may be more general.

We describe here a novel approach to minimize a) the size of SOAP messages, b) the cost of handling XML messages for high performance Mobile Grid/Web service applications. We discuss our implementation, which uses WS-Context-complaint Information Services as a Context-store.

This paper is organized as follows. In section 2, we discuss relevant works. We discuss our architectural and implementation details in Section 3. In Section 4, we present the performance evaluations of proposed

approach. We summarize and discuss the future work in Section 5.

2 Background

In this section we overview previous efforts that focus on improving Web Service communication performance. We also discuss existing researches focusing on standardization of web service communications.

2.1 The lineage of SOAP alternatives for mobile Web Services

Because of increasing demand of binary form of XML-Based communication methods, W3C Workshop [10] was held and produced the report on Binary Interchange of XML Information Item Sets (Infoset) [7]. The report includes conclusions of the workshop meeting on September 2003 as well as several dozens of position papers from various institutes [11] – [13]. The purpose of the workshop was to study methods to compress XML documents and transmit pre-parsed and schema specific object. It identified requirements of binary XML Infoset, for examples a) maintaining universal interoperability, b) producing a generalized solution that is not limited to a specific application domain, c) reducing process time including a data binding time, and d) negotiation about falling back to XML/SOAP text format if receiver can't understand binary. Web Service performance has been more recently reviewed at the 15th Global Grids Forum workshop (GGF 15) [22].

We put current approaches of improving Grid/Web Service communication performance into different categories. First, most proposals that follows the W3C XML Binary Characterization have a goal of producing a self-contained alternative to an XML message, which is optimized for faster processing and has smaller packet size. Approaches in this category replace a redundant vocabulary with indexes. Sun's Fast Infoset project [14], XML Schema-based Compression (XSBC) [15], XML Infoset Encoding (XBIS) are several examples of the category. The second category is a non self-contained alternative, such as Sun's Fast Web Services [11] and the Indiana University Extreme! Lab's recommendation [16]. Our HHFR also falls on to this category. The last category is a message compression approach. Compression reduces the size of a XML document, but increases processing time. XML-specific compressions like XMill [17] achieve better ratio than conventional compression utilities like GZip [18]. But even XMill doesn't improve performance much because of the additional layer of processing – compression and decompression.

The Global Grid Forum's Data Format Description Language (DFDL) [19] is a descriptive language. It is proposed to describe a file or a stream in a binary format for Grid computing. Like the older Extensible Scientific Interchange Language (XSIL) [20], it is XML-based and comes with an extensible Java Data model. DFDL defines the structure of data. For example, it defines a number format of data, such as whether it is a big-endian or little-endian, and a complex data format such as an array. Also DFDL is designed to be processable through a DFDL parser and its data model. We designed the message format description of our Flexible Representation based on DFDL. In our Handheld Flexible Representation architecture, we define simple XML-schema based descriptive language and develop a language parser using XML Pull Parser (XPP) [21]. Our prototype implementation is not as in-depth as DFDL, though it will be enough to show advantages of our approach.

The HHFR architecture is categorized the non self-contained alternative approach and it focuses on optimizing message stream that we believe the most appropriate approach for mobile Web Service applications with high latency connections and limited computing power.

2.2 Specifications for Grid/Web Service interactions

Often an application for a specific purpose is composed of multiple Grid/Web Services. For example, an airline reservation system could consist of several Web Services, which are combined together to process reservation requests, update customer records, and send confirmation. To standardize the way of sharing information or meta-data between multiple collaborating services, a specification is needed. There are many specifications, such as Web Service Composite Application Framework (WS-CAF) [6], Web Service Resource Framework (WSRF) [7], and WS-Metadata Exchange [8]. They are introduced to define stateful interactions among services.

The WS-CAF specification is a collection of three specifications, WS-Context, WS-Coordination Framework (WS-CF), and WS-Transaction

Management (WS-TXM). WS-Context specification defines a simple mechanism to share a common context for multiple participating Web Services. A participating application can discover results of other participants' execution, which is stored as context. WS-CF defines a coordinator, which makes a context operation persistent and message delivery guaranteed. WS-TXM defines three distinct transaction protocols: two phase commit, long running actions, and business process flows. They make existing transaction managers interoperable. Three specifications comprise a stack of functionality. WS-Context is at the bottom and adding WS-CF and then WS-TXM.

Web Service Resource Framework (WSRF) specification, which is proposed by Globus alliance, IBM, and HP, is an alternative specification to WS-CAF [9]. WSRF defines conventions for managing state, so that collaborating applications discover, inspect, and interact with stateful resources in standard and interoperable ways like WS-CAF. Web Service Metadata Exchange (WS-ME) provides a mechanism a) to share information about the capabilities of participating Web Services and b) to allow querying a WS Endpoint to retrieve metadata about what to know to interact with them. The combination of two specifications would achieve what WS-CAF targets. But the combination of WSRF and WS-ME have limitations to accomplish state management, since WSRF just enables access and update rights and WS-ME defines only how to access interaction independent metadata.

Among the existing specifications, which standardize service communications, we choose WS-Context Specifications to design our architecture. Unlike the other service communication specifications, WS-Context models a session metadata repository as an external entity where more than two services can easily access/store highly dynamic, shared metadata.

3 Architecture Overviews and Implementation

The architecture of the proposed approach consists of two individual systems: HHFR and FTHPIS, as depicted in Figure 1. The HHFR architecture provides layers, which optimize and stream messages to achieve high performance mobile Web Service communication. HHFR uses FTHPIS as the Context-store to store redundant or unchanging parts of messages. It should be noted that HHFR uses only a part of FTHPIS's functionalities, WS-Context compliant functions. These functions enable both participants of mobile Web Service conversation and the HHFR System to store unchanging parts of the communication state, such as the related message schema and the HHFR design specification schema.

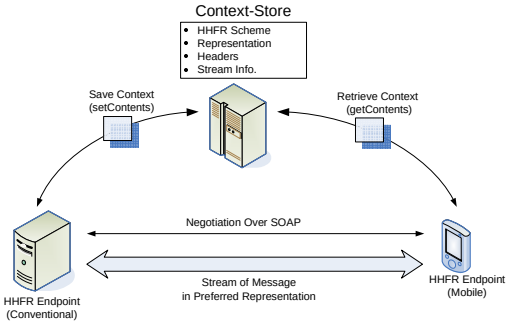

Figure. 1. Overview Of The Architecture

3.1 An Overview of Handheld Flexible Representation (HHFR)

As we discuss briefly in the introduction, the HHFR is a software architecture designed to provide an efficient mobile Web Service communication. There are three key features of the architecture: a) separation of message contents, b) streaming messages, and c) the use of Context-store.

3.1.1 Separation of message contents

Similar to other alternatives of the conventional SOAP-based Web Service communications, one of the key issues of HHFR architecture is optimizing conventional SOAP messages by separating redundant parts of the message content from SOAP messages. Some advantages of using optimized SOAP message format can be summarized as a) reduced bandwidth consumption, b) reduced message transit time and c) reduced time required for processing SOAP messages. By simplifying the structure of messages, our aim is to remove text parsing and serializing overhead. We achieve optimized message format/representation by two steps. First, HHFR separates message contents from its syntax (i.e. XML type and structure) and then convert the message content into a preferred

representation formed from the application's *in-memory* representation. To this end, one can avoid text-to and text-from data conversion, which consumes lots of process cycles. Second, HHFR streams message with preferred representation. To map non-XML based data (i.e., data separated from SOAP message content), and XML data (i.e., SOAP message itself), we define a DFDL-style data descriptive language, termed as *the HHFR schema*. It is a small subset of XML Schema Definition (XSD) [27] with some additions.

3.1.2 Streaming messages

The HHFR approach works best for stream of messages. For example, messages between a specific service and a client might have the same structure and data type, if the client keeps requesting the same type of information. Although, the values in the message may change, the structure and type of data will remain the same during consecutive transactions, which we call *a stream*. Therefore the structure and type of messages in the stream can be transmitted only once, and the rest of messages in the stream contain only payloads. The structure and type of the message are defined in HHFR Schema language. To establish such message stream, two endpoints should negotiate about the characteristics of the stream at the beginning of the session. In this case, the two endpoints negotiate a preferred representation (a binary representation), a transport protocol (TCP or UDP), and quality of service issues (messaging reliability and/or security). The negotiation uses a conventional SOAP message, so that two endpoints fall back to conventional SOAP messaging when the negotiate fails.

3.1.3 The use of Context-store

The Context-store, which is our research focus in this work, is a repository for storing redundant or unchanging parts of the messages in the stream and the context of the stream, such as message structure and type as a HHFR Schema document and characteristics of the stream.

3.2 An Overview of Fault Tolerant High Performance Information Service (FTHPIS)

Information Services may be thought of as a solution to general problem of managing metadata about Grid/Web Services. The benefits of using an Information Service in Mobile Computing Environment are two fold.

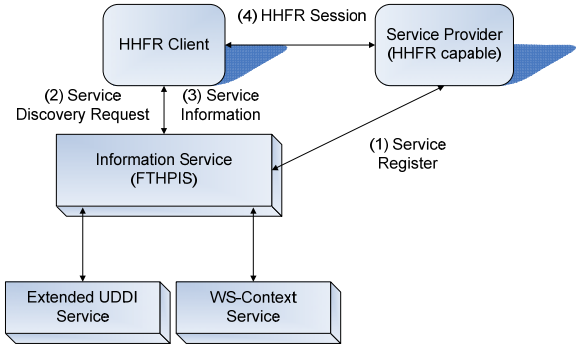

Figure. 2. UDDI Usage For Service Discovery

The first advantage is related to dynamic discovery capabilities of mobile Web Services. Mobile computing environment presents fragile communication characteristics, such as intermittent wireless connections and low bandwidth. These characteristics cause high communication failure rates and limitations in fast message exchanges. Such limitations require a solution where each mobile device can interact with an available service which may satisfy the requirements of fast and reliable communication between mobile devices and mobile Grid/Web Services. Also, mobile services have a volatile behavior. In other words, these services may come and go. It is important to locate mobile services in a dynamic fashion. In order to solve these limitations, an Information Service is required to dynamically locate the most reliable and high performance mobile services among those which are of the same type.

The second advantage is related to providing fast and efficient Web Service transactions in order to satisfy fast and reliable communication requirements of mobile computing Environment.

We have designed and implemented Fault Tolerant High Performance Information Services (FTHPIS) that supports both dynamically generated and semi-static metadata. As depicted in Figure 2, we have utilized two existing Web Service standards (Universal Description, Discovery, and Integration (UDDI) [23] and WS-Context) in our implementation. We have extended existing UDDI Specifications and designed an extension to UDDI Data Structure and UDDI XML API to be able to associate both

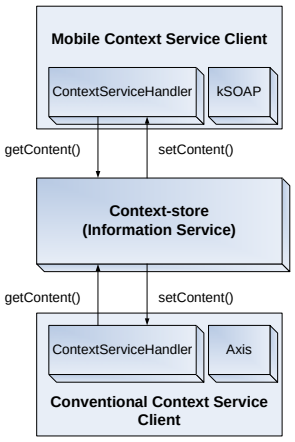

Figure. 3. Mobile and Conventional Context Service Clients

prescriptive and descriptive metadata with service entries [24]. We used name-value pairs to describe characteristics of services. Apart from the similar methodologies [25], [26], we provide both general and domain-specific query capabilities. An example for domain-specific query capability could be XPATH/RDQL queries on the auxiliary and domain-specific metadata files stored in the UDDI Registry. As depicted in Figure 3, a possible scenario where FTHPIS may be used in mobile computing environment is as follows: 1) a HHFR capable service provider registers its service to FTHPIS, 2) HHFR client sends discovery request to locate available HHFR capable services, 3) FTHPIS response with a list of services matching the request, 4) HHFR client initiates the session with one of the available services. We have also implemented extended version of WS-Context Specifications to provide an interface for publishing and accessing session metadata. This way the FTHPIS supports not only quasi-static, stateless metadata, but also more extensive metadata requirements of interacting systems. Here, an Information Service forms a hybrid XML metadata store for both dynamic, high updated session data and static/semi-static, rarely changing metadata.

We use our FTHPIS as a medium between communicating parties to store redundant parts of the XML based SOAP messages. Here, the Information Service keeps track of context information shared between multiple participants in Grid/Web Service interactions. It also maintains user profiles and preferences, application specific metadata, information regarding sessions and state of entities in these sessions. A context may contain arbitrary information which in turn enables sessions to be shared across multiple services and shared session content to be changed over time. A context might be associated to either to a service or the session itself. In this research, our main focus is session related context which is generated dynamically during the conversation of participating entities in a session.

3.3 Implementation of Mobile Web Service application with Context-store

As the FTHPIS system is implemented as Apache Axis compatible Web Service, a general client to the FTHPIS system requires a SOAP-Java binding of the Apache Axis [28] library. The Axis implementation for the J2ME environment has not been developed yet and is not expected to be developed in the near future because of the lack of related programming libraries, such as advanced XML parsers, utility libraries, and memory space limitation. Poring the existing client interface code in Java 2 platform, Standard Edition (J2SE) [30] to Java 2 platform, Micro Edition (J2ME) [29] is not a viable approach because of limited functionalities of J2ME. Due to these limitations, we choose an alternative approach which serializes a) SOAP request messages direct from an *in-memory* representation and b) parses response messages with a simple SOAP parser without the Axis' Java-SOAP binding. To do this, we use the kSOAP [31] library, which is an open source SOAP library for J2ME applications.

We use the two primary WS-Context related functions, `getContent()` and `setContent()` methods, to access and store redundant information into the Context-store. In the implementation, we made method calls from mobile clients not tied to any other operations in the HHFR session by making our implementation as multi-threaded. This way, these methods may be called at anytime when the HHFR runtime or the HHFR client service needs to create, update or retrieve context in the Context-store. The following Java program a) creates a `ContextServiceHandler` object with the Context Service URl and the version of the service as parameters, b) stores a given context paired with a unique identifier, and c) retrieves context.

```
ContextServiceHandler handler =
  new ContextServiceHandler(SERVICE_URL, 0);
try {
   boolean result =
      handler.setContext(identifier, context);
   Object contextData =
      handler.getContext(identifier);
      }
catch (java.lang.InterruptedException exception) {
   exception code…
}
```

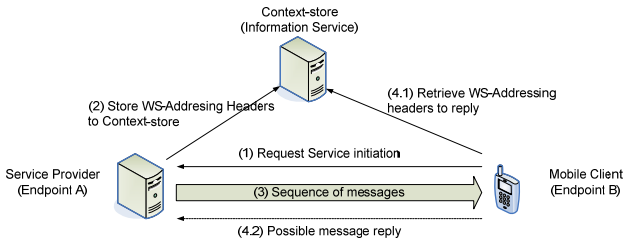

Figure. 4. A scenario for WS-Addressing example

As the reader observe in the code sampled given above, the `getContent()` and `setContent()` methods throw an interrupt exception (`java.lang.InterruptedException`), which is thrown when a thread is waiting, sleeping, or otherwise paused for a long time. As the handler runs as a thread, it avoids a possible deadlock situation, which may be occur when network failed or operational error.

The ad-hoc method to generate a SOAP message is the biggest obstacle to automate client code generation, compared to automatic Java binding generation of Axis. The current implementation of using FTHPIS Service as a Context-store holds a limitation in high level Java-binding Interface to SOAP message for HHFR clients due to limited programming library of mobile computing.

4 Evaluation

In this section, our main goal is to investigate the performance of proposed approach. In the system evaluation section, we particularly address the following questions:

- How much performance gains by using a Context-store in Web Service Messaging?
- What is the baseline performance of the Context-store implementation (the FTHPIS system) from the perspective of a mobile client?

4.1. Evaluation of Performance Measurement With and Without Context-store Usage

In this experiment, we measure the bandwidth gain from reducing the size of message. Our choice for a sample SOAP header is from the WS-Addressing Specification [32]. The WS-Addressing Specification defines a transport neutral mechanism to address Web Services and messages. It defines two constructs, endpoint references and message information headers, to convey information between Web Service endpoints, the target of Web Service messages.

Participating nodes store unchanging SOAP parts to the Context-store (Information Service) and retrieves them when it is needed. So we can store most of WS-Addressing header parts to improve message communication performance. A practical example of this usage case is as following: two Web Service endpoints: A, which is a service provider and B, which is a mobile Web Service client start a series of Web Service transactions. Endpoint B requires WS-Addressing headers only if it needs to send a reply or addresses an individual message. So the header is archived in the Context-store. Among elements of WS-Addressing headers, `<messageID>` shouldn't be archived because it is unique for each message. The example scenario is depicted in Figure 4.

Since a purpose of the experiment is to show how much performance gain in time by adopting the Context-store, we measure the performance of a HHFR transaction, not a conventional Web Service transaction. It is because messages are exchanged as a HHFR stream after a negotiation stage, which includes a context-saving process at the beginning.

As depicted in Figure 4, a service we use is a simple echo service for experiment, which immediately returns a received message back to a sender. First, we measure round trip times (RTTs) of echo transactions with a full WS-Addressing header over HHFR communication channel, which simulates a *without the Context-store* case. Then, we measure RTTs of echo transactions with a minimized header, which simulates a *with the Context-Store* case. Since we are interested in a one-way transit time, we divide result

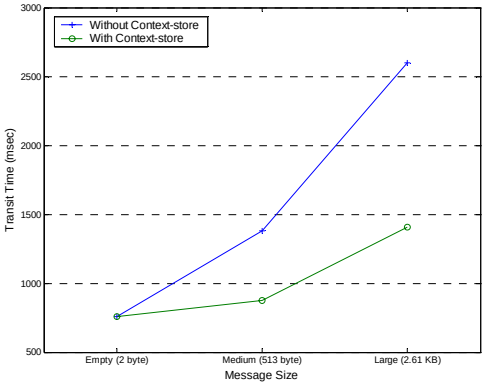

Figure. 5. Transit time of message exchange: each test set represents 30 iterations and the average value is presented on the graph. Messages are exchanged using HHFR communication channel

RTTs by two.

The results show we save on average 83% of message size and on average 41% of transit time by using our approach. The summary of the result is shown in Figure 5. The scenario is simplified to remove possible Context-store accesses during a stream session. So our experiments are results of the best case scenario.

MIDP's `System.currentTimeMillis()` call is used for timing measurements. Table 1 contains a summary of testing environments.

Table 1 Summary of Machine Configurations

Context-Store: GridFarm 8	
Processor	Intel® Xeon™ CPU (2.40GHz)
RAM	2GB total
Bandwidth	100Mbps
OS	GNU/Linux (kernel release 2.4.22)
Java Version	Java 2 platform, Standard Edition (1.5.0-06)
SOAP Engine	AXIS 1.2 (in Tomcat 5.5.8)

Service Client: Treo 600	
Processor	ARM (144MHz)
RAM	32MB total, 24MB user available
Bandwidth	14.4Kbps
OS	Palm 5.2.1.H
Java Version	Java 2 platform, Micro Edition CLDC 1.1 and MIDP 2.0

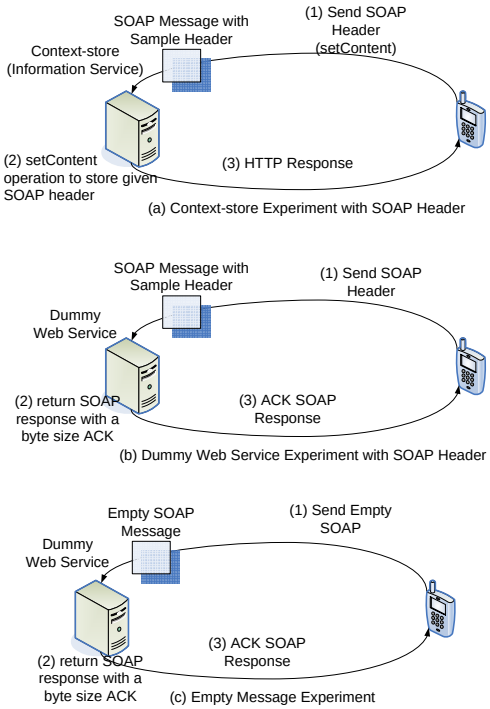

Figure. 6. The configuration of experiments (setContent)

4.2. Evaluation of Performance Measurement With and Without Context-store Usage

In this experiment, we investigate baseline performance of the Context-store (FTHPIS with version 0.9v1) accessing overhead. We use a Dummy Web Service as a yard stick to compare results of the Context-store access. It echoes back a received message, like the service of the previous experiment. The purpose of the test is to measure a pure data delivery-overhead of the given XML fragment (the unchanging SOAP header) without processing any Context-store related operation. The experiment is conducted with the same configuration of the previous experiment. We also conduct a simple empty message service that receives an empty SOAP message and returns it with the same configuration setup. The scenario of the `setContent` experiment and two other yardstick experiments are depicted in Figure 6.

The message used as a sample SOAP header that is stored in the Context-store is a message example in Web Service Reliable Message Specification (WS-RM) [33]. The size of the payload is 847 byte and the entire SOAP message size is 1.58KB. The size

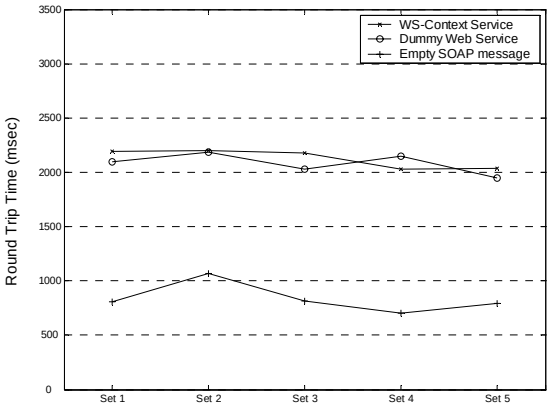

Figure. 7. Round trip time of experiments: each test set represents 30 iterations and the average value is presented on the graph.

of empty SOAP message for the latency test is 359 byte. We measure RTTs of the entire message trip except a message generation overheads. The summarized test results are shown in Figure 7.

As depicted in the figure, the Context-store (FTHPIS) service access adds up minimal or no overhead to the overall performance by comparing its results with Dummy Web Service test results, because the only difference between two experiments is whether the service accesses the Context-store service. We note the time difference between two experiments is less than 100 msec, which we claim the Context-store access overhead. The major factor that has a linear relationship with RTTs is the size of the message. Messages in the Context-store and the Dummy service test are 1.2KB bigger than the Empty message test, and this causes big time differences between two groups.

5 Conclusion

We investigated a novel approach to optimize Web Service messaging performance, in which the system stores unchanging or redundant metadata exchanged in the messages in a third-party repository, the Context-store. This approach increases the efficiency of conversational Web Service message exchange, since the messages contain many redundant SOAP message elements in a conversation.

We evaluated the performance gains of adapting our approach. As anticipated, the empirical result shows that the message size is reduced to 17% of original and transit time is saved in 41% (on average). The Context-store accessing adds maximum 100msec overhead, which is 2.5% of adding to the RTT of the same size message delivery, to RTT of the dummy Web Service.

Combined with separation of message contents and message streaming, our approach – metadata storing to repository (the Context-store) – confirms the feasibility of our experimental architecture for efficient and fast mobile Web Service applications.

References:

[1] S. Oh, G. C. Fox, "Optimizing Web Service Messaging Performance in Mobile Computing," *Community Grids Lab Technical Paper*, Feb. 2006.

[2] S. Oh, H. Bulut, A. Uyar, W. Wu, G. Fox, "Optimizing communication using the SOAP Infoset for mobile multimedia collaboration applications," in Proceedings of *The IEEE 2005 International Symposium on Collaborative Technologies and Systems (CTS 2005)*, St. Louis, Missouri, USA, May. 2005.

[3] Bunting, B., Chapman, M., Hurley, O., Little M, Mischinkinky, J., Newcomer, E., Webber, J., and Swenson, K., "Web Services Context (WS-Context)," http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CTX.pdf

[4] M. S. Aktas, G. C. Fox, and M. Pierce, "Managing Dynamic Metadata as Context," in Proceedings of *The 2005 Istanbul International Computational Science and Engineering Conference (ICCSE2005)*, Data Istanbul, Turkey, Jun. 2005.

[5] Community Grids Lab, "Extended UDDI and Fault Tolerant and High Performance Context Service," http://www.opengrids.org

[6] E. Newcomer et al., "Web Service Composite Application Framework (WS-CAF)," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf

[7] K. Czajkowski et al., "The WS-Resource Framework", *Globus Project*, White Paper, 2004, http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[8] K. Ballinger, et al, “Web Services Metadata Exchange,” Sep. 2004, http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-metadataexchange.pdf.
[9] The Globus Alliance, http://www.globus.org
[10] World Wide Web Consortium, “XML Information Set”, http://www.w3.org/TR/xml-infoset/
[11] P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E Pelegri-Llopart, “Fast Web Services”, Aug. 2003, http://java.sun.com/developer/technicalArticles/WebServices/fastWS/
[12] J. H. Gailey, “Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation”, Dec. 2002, http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx
[13] D. Brutzman, and A. D. Hudson, “Cross-Format Schema Protocol (XFSP)”, Sep. 2003, http://www.movesinstitute.org/openhouse2003slides/XFSP.ppt
[14] P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E Pelegri-Llopart, “Fast Infoset”, Jun. 2004, http://java.sun.com/developer/technicalArticles/xml/fastinfoset/.
[15] E. Serin and D. Brutzman, “XML Schema-Based Compression (XSBC)”, http://www.movesinstitute.org/xmsf/projects/XSBC/03Mar_Serin.pdf
[16] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, and D. Gannon, “Requirements for and evaluation of RMI protocols for scientific computing,” In Proceedings of *Supercomputing 2000 (SC2000)*, Dallas, TX, Nov. 2000.
[17] P. Deutsch, “GZIP file format specification version 4.3,” May 1996, http://www.ietf.org/rfc/rfc1952.txt.
[18] 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 (SIGMOD2000)*, pp. 153-164, 2000.
[19] M. Beckerle, and M. Westhead, “GGF DFDL Primer”, http://www.gridforum.org/Meetings/GGF11/Documents/DFDL_Primer_v2.pdf
[20] R. Williams, “XSIL: Java/XML for Scientific Data”, Jun. 2000, http://www.cacr.caltech.edu/projects/xsil/xsil_spec.pdf
[21] Java Community Press, “Java Specification Requests 173: Streaming API for XML”, http://www.jcp.org/en/jsr/detail?id=173
[22] Global Grid Forum 15 Community Activity, ”Web Services Performance: Issues and Research” http://www.gridforum.org/GGF15/ggf_events_schedule_WSPerform.htm
[23] Mehmet S. Aktas, Galip Aydin, Geoffrey C. Fox, Harshawardhan Gadgil, Marlon Pierce, Ahmet Sayar, “Information Services for Grid/Web Service Oriented Architecture (SOA) Based Geospatial Applications”, Technical Report, Community Grids Lab, June, 2005
[24] T. Bellwood, L. Clement, and C. Riegen, “UDDI Version 3.0.1: UDDI Specification,” *Technical Committee Specification.* http://uddi.org/pubs/uddi-v3.0.1-20031014.htm.
[25] A. ShaikhAli, O. Rana, R. Al-Ali and D. W. Walker, “UDDIe: An Extended Registry for Web Services,” In Proceedings of *the Service Oriented Computing: Models, Architectures and Applications*, Orlando, Florida, January 2003.
[26] V. Dialani, “UDDI-M Version 1.0 API Specification,” *Report University of Southhampton*, UK 2002
[27] World Wide Web Consortium, “XML Schema Definition (XSD),” http://www.w3.org/XML/Schema
[28] Apache, AXIS, http://ws.apache.org/axis/
[29] Java 2 platform, Micro Edition (J2ME), http://java.sun.com/j2me/
[30] Java 2 platform, Standard Edition (J2SE), http://java.sun.com/j2se/
[31] kSOAP, http://kobjects.org/
[32] D. Box et al., “Web Service Addressing (WS-Addressing), Aug. 2004, http://www.w3.org/Submission/ws-addressing/
[33] R. Bilorusets et al., “Web Services Reliable Messaging Protocol (WS-ReliableMessaging), Feb. 2005, ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200502.pdf