Differential Caches for Web Services in Mobile Environments

P. Bodorik, M.S. Qaiser

Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia, Canada
{bodorik, qaiser}@cs.dal.ca

D.N. Jutla

Sobey School of Business, Saint Mary's University
Halifax, Nova Scotia, Canada
Dawn.Jutla@smu.ca

***Abstract* —** Although web services have been espoused due to their many benefits, it is known that overhead delay associated with invocation and execution of web services is high. Consequently, much research has been expended on minimizing those delays. In many situations an application invokes a web service repeatedly such that some or most of the data returned by the web service does not change. For instance, many web services that return schedules, such as bus or train schedules, exhibit this property. We use caches to avoid repeated transfer of data sent by a web service to an application, if that data does not change between invocations of the web service. We present *Differential Caches* with the accompanying *Differential Updates* method and the *Mobile SOAP (mSOAP)* protocol. We present two cache designs: one based on the server supporting a cache for each application, while in the other one the server supports a shared cache for all applications. The protocol is flexible in that other optimization techniques, such as encoding, can also be applied with the Differential Updates method. We created a research prototype and performed experiments to evaluate the method's potential benefits and also its overhead. The results of experiments show clearly that potential benefits outweigh the overhead. The mSOAP protocol with Differential Caches obtained a speedup of up to 800%, in delivery of the web services' replies in comparison to the SOAP communication. Further improvements in delays were gained when encoding was used in conjunction with Differential Caches.

I. Introduction

Although web services, which generally use SOAP *(Simple Object Access Protocol)* for communication with clients, provide high interoperability for different platforms, they also increase delays due to the large size of messages exchanged in the loosely-coupled XML-based communication. The large size of SOAP messages causes difficulties particularly to the developers of mobile client platforms and applications because of the constraints on memory, processing speed, communication bandwidth, and associated power consumption. In mobile communication, the size of transferred data has been recognized to be the most important parameter for most applications (Varshney, 2002)). It is therefore no surprise that much research focused on reducing the overhead due to XML messaging in SOAP. Most of the research focuses on either compression techniques or converting a text based message into the binary encoding format to reduce the messages size and thus reduce the network delays. Examples include TDXML, WBXML, WSOAP, Millau, Gzip and Jzlib ((Apte, 2005), (Ng, 2006a, 2006b)). The concept of encoding is simple in that a binary encoding table is created for the verbose XML tags. As long as both the client and the server have the encoding table, the tags in the transferred messages are replaced by their codes. This works well for applications and web services that have a static and predictable vocabulary of tags – otherwise complexities arise due to potential inconsistencies.

We propose another approach, to minimize overhead due to the verbose XML communication – by performing optimization for mobile applications that exhibit certain characteristics. Frequently, an application invokes a web service repeatedly and, in addition, frequently the data returned by the web service is such that some or most of it does not change from one invocation of the web service to another. Many web services that are invoked to provide a schedule of some activity are of this type. For instance, transportation schedules for trains, buses, and airlines and training courses schedules fall into this category (Ion, 2007). An application invokes a web service for a schedule, e.g., a bus or a train schedule, and most of the time there are not too many changes in the schedule from one invocation of the web service to another (if the user is interested for a schedule for a particular bus/train). Another example includes services that provide ratings, such as ratings of stocks. A user application may repeatedly invoke a web service to find ratings of stocks/bonds of interests to see if there are any changes in the ratings. It is web services with this type of characteristics that we are targeting in this paper. We avoid repeated transfer of data, sent by the web service to an application, if that data does not change between invocations and hence we reduce the size of XML messages.

A. Differential Caches for Reducing Communication Delay

We introduce the notion of a *Differential Cache* consisting of a pair of software caches: one on the server, which executes the web-service, and one on the client, which invokes the web service. The pair of caches is used to store the data sent by the web service to the invoking application. On the server, the cache is used to remove from the web service's reply data that has already been sent in the previous message. On the client, the cache is used to reconstruct the message sent by the web service. When a web service is executed, we ensure that the reply sent to the invoking application contains only data that has changed since its previous invocation by that application. By not including the data that has not changed from the previous invocations of the web service, we reduce the size of data transferred over the network and hence reduce communications delays.

B. Objectives

Our objective is to create a novel cache-based design and a protocol that support the Differential Caching method to

reduce the SOAP payload data. There are a number of issues that need to be addressed:

1. Transparency: Considering that most web services are relatively simple, the cost of software development of creating and using the caches may be relatively high in relation to the software development cost of creating the web service and invoking the web service by an application. As a consequence, the architecture and the protocol must be transparent to the software-developers of the web services and also to the software-developers of the applications that invoke the web service. The design and the protocol should not assume any knowledge about the web services or applications invoking them. That is, the cache system should be transparent to the developers of web services and applications that invoke them.
2. Efficient implementation: Clearly, the caches and their management have to be efficient as they constitute overhead. Furthermore, the Update Managers modify XML messages exchanged between the client and the web service – the modified content should be minimal in size.
3. The protocol should be such that, in addition to Differential Caching, it can support other optimization techniques used to reduce the size of transferred data.
4. Evaluation: Introduction of caches is overhead and it is justified only if the benefits, due to the decrease in the size of exchanged messages, outweigh the overhead costs – this trade-off must be evaluated.

C. Outline

The second section presents the system architecture and the message exchange protocol. It also discusses issues arising when utilizing the server and client-side caches to capture data that does not need to be transferred over the network. The system design of the caches and encoding are presented in Section III. A shared cache is described in Section IV. Section V discusses the issues of complexity, scalability, and fault tolerance. We implemented the differential cache as a proof of concept and use it to explore overhead delays and potential benefits under various scenarios. Experimentation is described in the sections VI and VII. The last two sections respectively provide related literature, and a summary and conclusions.

II. Architecture and Mobile SOAP (mSOAP) Protocol

A. Assumptions

Recall that we are targeting environments in which a client repeatedly invokes a web-service that returns, to the invoking application, a collection of data such that some or most of it does not change (Gudgin, 2007).

Our architecture is for the very desirable case in which the Differential Updates method and the Differential Caches are transparent to the software developers of the web service and to the developers of applications that invoke the web service. On the server-side, the Differential Updates method is incorporated within the platform used for provisioning of web services; in particular, for the purposes of the description we assume that web services are provided using the Apache Axis2 Framework (Axis-dev, 2010) platform and that handlers can be added to the processing chain of web service requests or replies. On the client-side, our Differential Updates software is an extension of the framework used to facilitate communication by applications with web services. We assume zero knowledge about the web service and any application invoking the web service. Furthermore, the web service and the applications are not touched/modified by inclusion of the Differential Updates and Caches.

B. Architecture

We introduce a pair of caches, one on the server and one on the client, as shown in Figure 1.

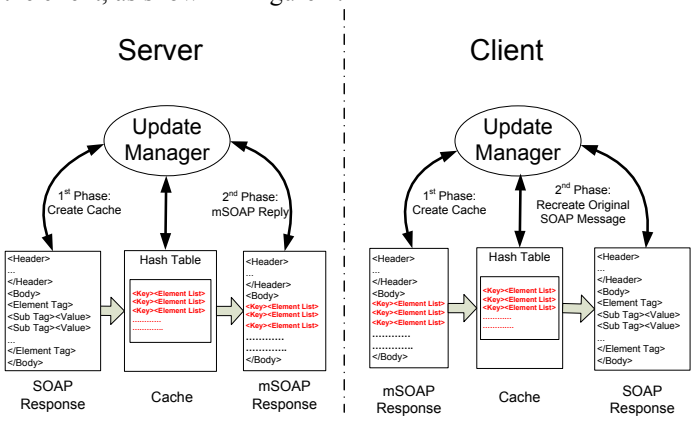

Figure 1 – Client and Server-side Caches

The caches store the latest reply by the web-service and are managed by the client and server-side *Update Managers*. When the web service is executed for the first time, before the reply is sent to the client, the server-side Update Manager creates a cache for the web service and stores in it the reply. When the client receives the first reply from the web service, before the reply is forwarded to the client application, the client-side Update Manager creates a cache for the web service and stores the content of the reply in it. At this point, the caches are prepared and subsequent invocation of the web service by the application will be performed using the *Differential Updates* method. When the web service is invoked again, the reply message, produced by the web service, is examined by the server-side Update Manager to compare the reply message with the previous one. The Manager modifies the reply message to include only those list elements - tags and values - which have changed since the previous execution of the web service.

Figure 1 shows Update Managers as the architectural software components that maintain the caches and use them in manipulating the exchanged XML messages.

C. Mobile SOAP (mSOAP) Protocol

The mSOAP protocol is an extension to the SOAP protocol to contain information relevant to the Differential Updates method. The activities performed by the update managers are shown in Figure 2. Obviously, Differential Caching can work only if both the client and the server platforms have compatible cooperating caching software and are

communicating using the same protocol. Communication and processing proceed in two phases (see Figure 2).

In the first phase of the protocol, the client and the server need to inform each other that they are prepared to use the Differential Caching and exchange appropriate information/parameters, which are required to initiate the caching and Differential Updates. Both the server and the client cache the content of the first reply message. The second phase proceeds on the second and subsequent invocation of the web service.

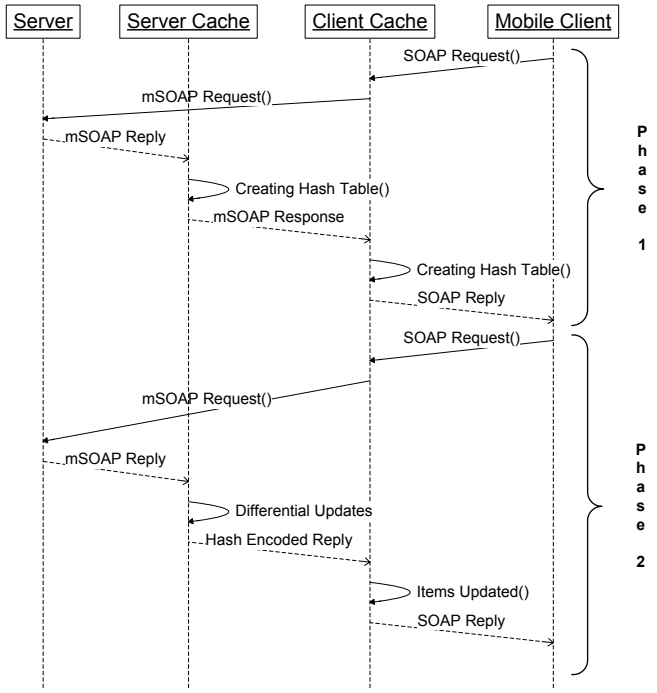

Figure 2 – Activities Performed by the Client-side and Server-side Updated Managers

The protocol begins with the web service request created by the client application for the *first time*. In essence, an application makes a request to the web-service client framework, in which the client-side Update Manager is incorporated, to create and send a SOAP request message. After the SOAP request message is created, the client-side Update Manager is invoked and it modifies the request header to include information about which mSOAP protocol version(s) it is willing to use and also its own version number. Also included is information on any encoding algorithms the Manager is able to support in an exchange of data – encoding will be elaborated upon shortly. In short, an mSOAP request is created and transferred to the server.

When the request is received on the server platform, a handler invokes the Update Manager to processes the request before it is forwarded to the web service. The Manager recognizes the client's request to use the mSOAP protocol and ensures that the Differential Caching algorithm and the mSOAP protocol version requested by the client can be supported and decides to use Differential Caching. The web service is invoked and passed the request. Its execution generates the first SOAP reply. The server's Update Manager then inserts into the SOAP reply a confirmation, which states that the Differential Caching and the protocol version will be used – in essence creating an mSOAP message. Also, the Manager creates the cache and stores the content of the reply message in it. The reply message is then transferred to the client.

Upon reception of the first reply message, the client recognizes that the server has agreed to use Differential Updates. It creates the cache and stores in it the content of the reply message before forwarding it to the application. Subsequent messages between the client-side and server-side Update managers will be exchanged using the mSOAP protocol and the Differential Updates method. It should be noted that the usage of the mSOAP protocol and the Differential caching are transparent to the client application and the web service – they perceive/assume normal SOAP protocol.

III. Differential Updates and Encoding

In this section, we overview the design of the Differential Caches and discuss the role of encoding in a message exchange. We discuss the case when the cache is not shared, i.e., we have a single pairing of a client-side cache with a server-side cache for each client application that repeatedly invoking a specific web service. An application is identified by a unique IP address (or, to be more precise, by a unique pair of IP address and the port number of a TCP protocol stack). A shared cache is described in Section IV.

A. Differential Caching and Updates Mechanism

Recall that the Differential Cache consists of a pair of caches, one on the server and one on the client. An application request for a web service and a web service's reply are XML messages that contain information expressed as data elements consisting of, at the lowest levels of the data hierarchy, of tags and values but no further sub-elements. We consult the WSDL for the description of the replies and identify web services that may benefit from Differential Updates – web services that have lists in their replies. Thus, only web services for which WSDL indicates that a list of elements is returned are considered for utilization of Differential Updates. This examination of the web services' WSDLs is done prior to operational processing of web services replies.

1) Server-side Cache

When the web service generates a reply to the application's first request for the web service, we cache portions of the XML reply message. In essence, we store in the cache a list of elements. Together with a list we also store its XML path identifying where in the XML message the list appears. Figure 3 shows a portion of a reply message from a web service that sends to the invoking application a list of elements. Furthermore, the cache is organized as a hash table facilitating fast storage and retrieval of lists (with their list elements). To store/retrieve a list, hashing on the XML path of a list item is used. Finally, we also modify the reply message by inserting special codes that identify those lists, in the XML message, that have been cached together with the number of elements in

the list. The special codes are inserted into the message by the server Update Manager and removed by the client Update Manager.

On subsequent invocations of the web service by the application, the server Update Manager compares the lists of the web service's reply XML message to those stored in the cache. Any element, of a list, which has the same value as in the previous reply stored in the cache, is removed from the message. Of course, such an element is re-inserted on the client-side once the reply is delivered to the client but before it is passed to the application. In this way the size of the reply messages is reduced and thus communication delays are reduced.

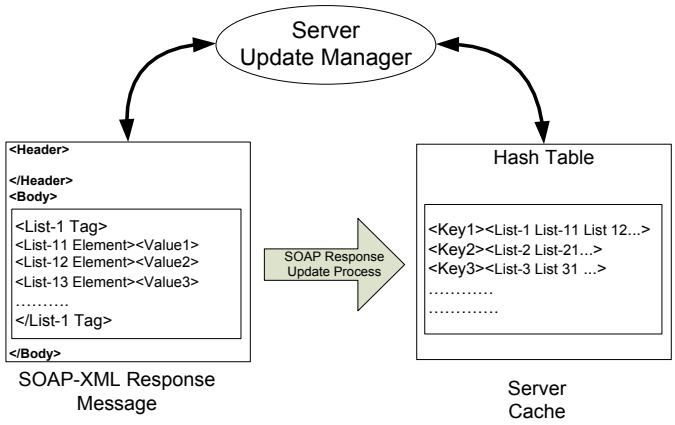

Figure 3 – Creating Cache to Store Content of the Reply

The above description is straight-forward when dealing with a static list of elements. Complications, however, arise if new elements are inserted in a list, or if elements of a list are removed, by the web service. When a new element is inserted in the list by the web service, the Update Manager on the server detects it when it compares the new reply message with the one in the cache. It inserts in the message a special code indicating a newly inserted list element together with an index of the preceding element of the list. Similarly, if an element of a list is removed by the web service (i.e., it no longer appears in the current reply), the Update Manager inserts into the reply message a special code together with the index of the element that has been removed. The code and index are in binary and hence short in comparison to the size of XML tags.

2) Client-side Cache

The processing of a web service reply on the client-side is complementary to that of the reply's processing on the server. When the first reply is received, it is cached in the client cache in a similar manner as on the server (see Figure 3). However, processing of the message is simpler as the cached lists are identified in the message by special codes. As on the server, the cache is organized as a hash table with the lists, extracted from the reply, being the hash table entries. Hashing is on the list's path in the XML message. When the web service is invoked again, subsequent replies from the web service are modified by the client Update Manager to restore the original message using the cache. The received reply message is searched for special codes that identify the cached list items. Elements of the list that have been removed on the server side are re-inserted from the client's cache – see Figure 4. The client Update Manager also recognizes the special codes, inserted in the message by the server Update Manager, that identify those elements of the list that were, in comparison to the previous reply, either inserted or removed by the web service.

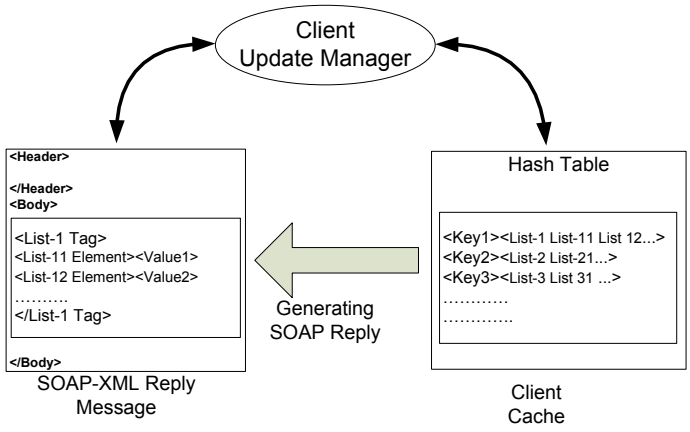

Figure 4 – Regenerating the Original Response Message

3) Application's New Requests

Clearly, an application may request from the web service new information, such that the web service's reply does not contain any, or not many, of the data elements of the previous reply. For instance, a mobile application may ask for a schedule of Bus #1 repeatedly but then it may ask (at the user's request) for a schedule for Bus #2. In such a case, the list of bus stops and times would be completely different in the new reply (for Bus #2) in comparison to the previous reply (for Bus #1). The server Update Manager keeps track of the number of changes in elements of a list between the previous and current reply and if the number of changes exceeds some threshold, then the cached reply is purged. The reply message is also tagged to inform the client Update Manager to purge its cache of the old reply.

B. Encoding

A disadvantage of using the XML format for messages is that XML is verbose and thus leading to messages that have large sizes. Various encoding techniques (e.g., (Girardot, 2000), (Devaram, 2003), (Werner, 2004) and (Naresh, 2005)) have been proposed and are being used to alleviate this problem. One of the techniques relies on exchanged messages having a static set of XML tags that are known to both the web service and the application. The tags are encoded and it is the codes that are transferred in XML messages instead of the tags themselves.

We use this encoding technique in our Differential Caching method. Recall the two-phases of the mSOAP protocol. The first phase consists of an exchange of messages, between the client and the server Update Managers, for notification that Differential Caching is used and of creation of the pair of caches. In the first phase, information is also exchanged about

which encoding technique, if any, is going to be used. In the first mSOAP message, in addition to the request for using Differential Updates method, the client Update Manager also includes a request for encoding to be used together with which encoding methods it can support. The server-side Update Manager, when processing the first reply from the web service, creates an encoding scheme for the tags of cached items and it includes this encoding information in the mSOAP reply: For each tag, the Manager also includes its code. When the client-side Update Manager receives the first reply message, for each tag it also finds its corresponding code and thus builds its encoding table. The server-side Update Manager modifies subsequent reply messages, produced by the web service, by replacing any tags with their codes from the encoding table.

In short, in the first phase discussed in the previous section, the encoding information is transferred from the server to the client. In the second phase, before a reply message is sent from the server to the client, the Update Manager on the server replaces XML tags by their codes, while on the client-side, when the Update Manager receives the reply message it replaces the codes with the XML tags.

IV. Shared Cache

Clearly, there are many applications on distinct systems that invoke **a** web service and having on the server a cache for each application (that invokes that web service) results in many server-side caches potentially storing the same data returned by the web service. In this section we describe a single shared cache.

A. Assumptions

Recall that we store lists, consisting of elements, in the cache: We consult the WSDL to identify a list, as an XML data element that contains a list of elements. For the shared cache we make an additional assumption in that the web service's reply does not contain a list that in itself may be repeated. For instance, the reply does not contain a list of bus schedules, for various buses identified by numbers, such that there are many lists in the reply, one for each bus schedule.

B. Shared Cache Architecture

The following issues arise when considering a cache that is shared:

a. How to determine, efficiently, whether or not a value for an element of a list has changed when compared to the previous value returned to the specific application.

b. How to determine which of the cached data elements of a list were or were not included in the previous reply to a specific application.

An efficient solution for the first issue is to use timestamps that are assigned by the server Update Manager. When the web service is invoked it results in a reply message that is examined by the server-side Update Manager, which also creates a timestamp of the web service's reply. If the value of an element in the reply's list is not the same as the corresponding cached value, then the Update Manager replaces the cached value with the new one, from the reply, together with the reply's timestamp value. Consequently, each element of a list stored in the cache has a timestamp – which is the time of the reply message from which the element came.

The second issue is resolved, on the server by the Update Manager, by keeping, for each application that invoked a web service, a list of references (actually hash-keys) to the cached list's elements that appeared in the last reply of the web service to that application.

1) Server Update Manager

The cache contains a hash table that stores elements of any list returned by web services in their previous invocations by applications. The element's hash-key is created using a function with arguments being the element's tags (but not the values) and the web service's unique ID. Each element (of a list) stored in the table contains, in addition to its tags and the value, also the timestamp of the reply (from a web service) from which the element came.

For each application that invokes a specific web service, there is an information object describing the most recent reply returned to that application by the web service. It contains a timestamp of the reply, the web service's ID, the application ID, and a list of hash-keys. The hash-keys identify the elements, of the list that appeared in the most recent reply (to the application by that specific web service). For fast access, these information objects are also stored in a hash table with hashing being done on the unique combination of the application's ID and the web services ID.

Consider now the case when a web service was invoked by an application and produced a reply that is now processed by the server-side Update Manager. Using the current time, the Manager creates a timestamp – it is the timestamp of the reply. The Manager searches the cache for the most recent reply of the web service for that application; more specifically it accesses, using the combination of the application ID and the web service ID, the hash table of the information objects. If there is no information stored in the cache on previous replies to the application by the web service, a new information object is created and stored. It contains the timestamp of the current reply and also a list of hash-keys that identify the elements of the list contained in the reply message – these elements of the list in the reply are inserted into the cache, i.e., they are inserted into the hash table, contained in the cache, using their hash-keys.

If, on the other hand, the information object is found – that means that the cache contains information on the previous reply by the web service to the application. The information object is retrieved – it contains the timestamp of the previous reply and also a list of the hash-keys that identify the elements, of the list, which were in the previous reply and are stored in the cache. The server Update Manager must compare the currently examined reply with the previous reply for the following cases:

a. New list element appears in the reply: Each such new element of the list is inserted into the cache (and also remains in the reply message).

b. List element in the previous reply no longer appears in the new reply: For any element, of the list, which appears in the previous reply but not in the new one, a special code is inserted in the reply message to inform the client-side Update Manager of the case.
c. List element appears in both the new and previous reply: The Manager needs to determine whether the value of the element of the list in the current reply is the same as in the previous reply. First, it compares the values of the current and previous elements. If they are different, the new element, appearing in the reply message, is stored in the cache together with the reply's timestamp. If the values are same, the Manager compares the timestamp of the previous reply to the application (timestamp obtained from the information object) to the timestamp of the element of the list stored in the cache. If the element's timestamp is "older" than the timestamp of the previous message then the value has not changed and the element is removed from the reply message (the element will be re-inserted into the message by the client Update Manager). Otherwise the element remains in the reply message.

The above organization facilitates storing of only one copy of an element of a list in the cache, as opposed to a copy of an element for each application that invokes the web-service. We do need to keep track of which elements of the list have been most-recently received by an application but this is done using hash-keys and thus reducing the storage size. The organization also facilitates fast look up of the information objects and search for elements of lists.

2) Client Update Manager

Processing of the reply message by the client Update Manager does not change in comparison to the case when the server did not have a shared cache.

C. Comments

Several issues, relating to the shared cache, have not been addressed. Some are the cache management issues. For instance, the cache needs to be examined for information objects and elements of lists that have not appeared in replies for a sufficiently long time so that they should be purged.

However, there is another issue that is critical for correct execution and that may also impact performance – the issue of synchronization of access to the cache. Web services are executed concurrently by threads. As a reply is examined by the server's Update Manager code, access to the caches data structure needs to be synchronized – this may affect the performance. We have not examined the potential impact of synchronization on the over-all performance at this time.

V. Complexity, Scaliability, Fault Tolerance

Complexity: Recall that the server has an individual cache for each application. Thus, caching on the server is, in **space** requirements, directly proportional to the sum, over all applications, of the number of cached items per **each** application. A caching operation on the server, executed by a web service invocation, is directly proportional, in **time**, to the number of items cached for that application. The time and space complexity for the *caching on the **client*** is directly proportional to the number of cached items.

Scalability: The scalability is not a significant issue assuming that the cached DB items can be supported by (can fit on) a server. Applications can be partitioned using their IDs and thus each application can be assigned to one of many servers. Requests from applications are directed to "their" servers using their IDs.

Fault Tolerance: Timestamps can be used to support fault tolerance in this simple request-response environment, which is less complex than a DB recovery environment. The server Update Manager inserts in the reply message a timestamp that is then stored by the client Update Manager in its cache. In a request message for a web service, the client Manager inserts the time-stamp of the cached reply. If the client crashes, upon recovery its cache will be empty with the time-stamp being set to zero (oldest timestamp). Consequently, the client time stamp in the request message to the server will force the server to send all of the data. If the server crashes, the cache will cold-start and the server will send all of the requested data in a response message to the application. The server will process subsequent requests normally utilizing the cache.

As in other work, we rely on TCP/IP for reliable delivery of messages (no duplicates and delivered in the order they were sent). There are no difficulties in case of a time-out and a repeated request made by the client application as timestamps are used to ensure that the client does not see out-dated data. In case of network partitioning: When the server is inaccessible, data can be supplied from the local cache, depending on the adopted model of data consistency (e.g., eventual consistency) while the network partitioning issue is being resolved on another level; however, this is another use of the cache and is consider to be out of scope for this paper.

VI. Experiment Setup

We have implemented a prototype as a proof of concept and performed experiments in order to explore potential benefits of using Differential Updates in reducing message sizes and thus delays. We also measured overhead. The prototype was for the non-shared cache organization described in Section III, i.e., when the server has a separate cache for each application invoking a web service.

We first overview the set-up and experiments, then we describe the platforms, instrumentation of the Update Managers on the server and the client, and finally describe the client application and the web service it invokes. The subsequent section reports results of experimentation.

A. Experimentation Overview

We created a simple application that repeatedly invokes a web service asking it to provide information on rating of stocks. The application invokes the web service while providing it a list of stocks that are of interest. The web-service accesses a DB system to retrieve the requested information about each stock, which is a rating of the stock, and then returns this information to the application – see Figure 5. We measure the delays due to various activities,

such as data transfer and Update Managers' overhead delay. The application has a number of input parameters that govern the list of stocks and how frequently the web service is invoked. One of the key parameters is the percentage of data/ratings returned by the web service that have changed from one invocation by the application to another – i.e., percentage of the list of stocks, which are returned by the web service, for which the ratings have changed since the previous invocation of the web service. The application and the web-service are running on distinct computing systems. We have instrumented the Differential Caching method and the mSOAP protocol as described below.

B. Platforms

The client machine running the application and the software to instrument the experiments, was Intel® Core™2 Duo Processor T5670 (2M Cache, 1.80 GHz, 800 MHz FSB).

The web service simply retrieves the requested stocks from a database stored on the MySQL Relational DB System. The DB was housed on the same platform/system as the web server and the web service itself. The server platform was Intel® Pentium® 4 Processor 2.80 GHz, 512K Cache, 400 MHz FSB. The web service accesses the DB through JDBC drivers used for Connection Bridge. There was no other load generated on the web server or the DB system besides the load generated by our experimental software. The available bandwidth for downloading the data from the server was 15 Mb/s, while the number of network nodes between the server and the client was 3. The available bandwidth was relatively steady as data was transferred only through local networks, one of which was a wireless network, with fixed routing tables. There was minimal interference from network activity generated by software outside of our experimentation.

C. Implementation of the Server and Client Update Managers

Apache Axis 2 1.5 Framework was used on the server for hosting the web services. The framework uses SAX based parsing for serialization and de-serialization of SOAP-XML messages. Web Services are created through Java class by defining appropriate functions and making the object of the class Serializable. mSOAP was implemented on the server through including a wrapper over the top of a base web services class with inclusion of a server cache. Requests were first sent to the wrapper class that invokes the base web services for execution. The wrapper class serves as another service on top of base web services. There were no security and privacy rules applied inside the framework and full control was given to the wrapper class.

On the client, a cache is included to store previously received response messages. A wrapper class is used on top of web service client in order to implement client-side Update Manager and required functions for the client cache. The client application was implemented by using Sun J2SE 5.0 JDK and the web service client was compliant with JSR-109.

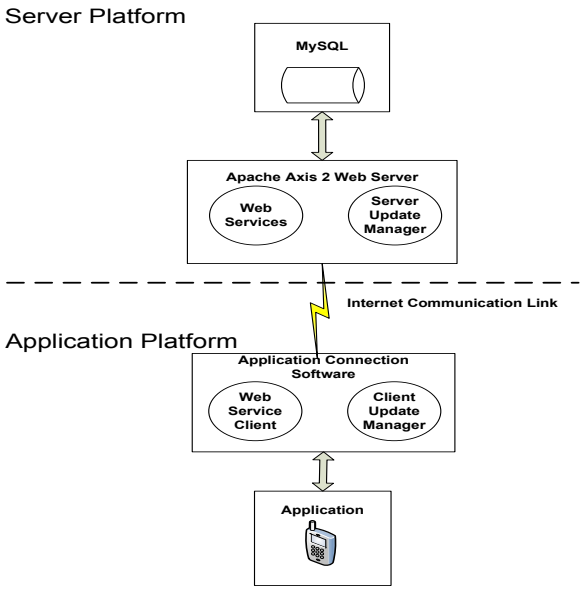

Figure 5 – Experimentation Setup

VII. EVALUATION

For comparison purposes we instrumented five different schemes for an application to obtain stock ratings, schemes labelled as SOAP, Encoding, Differential Updates, mSOAP, and Binary.

1. SOAP: This is a regular invocation of the web service without any of our optimization techniques.
2. Encoding: The mSOAP protocol is used for encoding but without the Differential Updates. On the first invocation of the web service, included in the reply is encoding for XML tags that are used to encode the web service reply messages. The client receives the encoding information in the first reply. It reconstructs the tags in the subsequent XML messages before forwarding them to the invoking application. Encoding and decoding are performed by the respective server-side and client-side Update Managers. However, the Differential Updates are not used.
3. Differential Updates: Differential Updates are in use by the server and the client-side Update Managers but there is no encoding of XML tags.
4. mSOAP: Both Encoding and Differential Updates are in use.
5. Binary: No optimization is performed. As the name implies, the XML messaging is not used and instead of invoking a web service, the application uses the *Remote Method Interface (RMI)* to invoke a method on the server remotely. The method on the server retrieves the stock rating data from the DB and returns them directly to the application. Thus the web server and web service overheads are avoided.

The application is repetitively asking for ratings on the same 195 stocks that are stored in the DB. One of the

parameters governing the experiments is expressed in percentages and governs the ***variability*** of the data values contained in the reply from the web service. Variability of 60% means that 60% of the stock ratings have values that are different from those of the previous invocation of the web service. It should be noted that, exploring the variability of data represents not only the case when the values, retrieved from the DB, change between invocations of the web service, but also the case when the application asks for stock ratings for a list of stocks that varies between invocations of the web service.

Recall that, with the exception of the communication delay, the systems are isolated and the only load is due to our experimentation. For each point presented in the graphs we made several runs and report the average. Because of the isolation of the systems, with the exception of some minor variations in the network delays, little variation was observed in repeated runs.

A. Average Overall Delays

Overall average delays for each of the five methods are shown in Figure 6. It should be noted that the shown delays are averages for variability of data records between 0%-100% made in 10% increments. Influence of the different percentile variability of records on the performance is shown in Figure 7 and will be discussed shortly.

Figure 6 shows that the highest delay, about 800 ms, is for the invocation of the web service using the normal SOAP protocol. If encoding is used to reduce the size of XML tags, there is an improvement in delay of almost 200 ms to about 600 ms. When Differential Updates are used, but without encoding, the delay is about 400 ms. When Differential Updates are combined with encoding in the mSOAP protocol, the average delay is reduced to about 350 ms. The smallest delay, of about 10 ms, is for the binary method, i.e., when the application uses RMI to invoke a method on the server to retrieve and return the stock ratings.

To generalize, two observations, already made by other researchers, are confirmed:

1. Minimization of the size of XML messages is beneficial as it reduces delays. We utilize encoding of XML tags as one of the minimization methods that has been discussed in a number of research papers (e.g., (Werner, 2004), (Apte, 2005), and (Suzumura, 2005)). However, we also propose a new method, Differential Updates, to further minimize the XML message size in certain scenarios.
2. In comparison to applications invoking methods directly to perform the required services, using RMI in our case, web services incur high overhead delay – thus interoperability of using web services comes at a steep price (e.g., (Devaram, 2003), (Ion, 2007), (Liu, 2007), and (Scholz, 2008)).

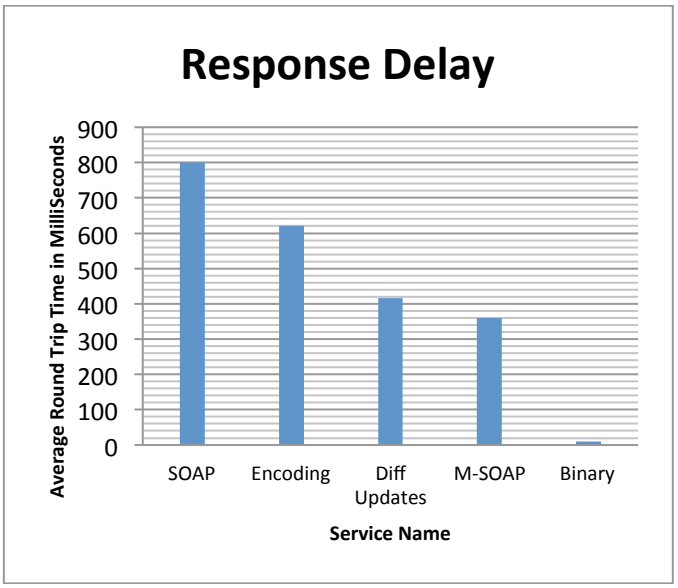

Figure 6 – Overall Delays

B. Variability of Data

The Differential Updates method reduces the size of XML responses from a web service if the response data is relatively static between invocations. We changed the variability of data returned by the web services and observed the effect on the response time. More precisely, we varied the percentage of stock ratings that were changed from one invocation of the web service to another. The observed delays for the five methods are shown in Figure 7. To repeat, the variability of 60% means that 60% of the stock ratings are different from those returned by the previous invocation of the web service.

The figure shows that the variability of data does not affect the SOAP and RMI methods. This is because these methods produce messages of fixed size if the number of stock ratings retrieved does not change. The Encoding method is similar in that, after the response message to the first invocation, which includes the encoding information, the message size does not change. Subsequent responses from the web service are of the same size, but are smaller than in SOAP as the XML tags are replaced in response messages by their codes. Encoding does not depend on the variability of the data in response messages in our case because the length of codes replacing the tags does not change.

The Differential Updates method, of course, depends on the data variability. Smaller variability means that more data elements are removed, by the server-side Update Manager from the response message, as they will be re-inserted on the client-side from the cache. Thus less variability means smaller messages sizes and lower delays. The same also applies for the mSOAP method, which, in addition to Differential Updates, also utilizes Encoding.

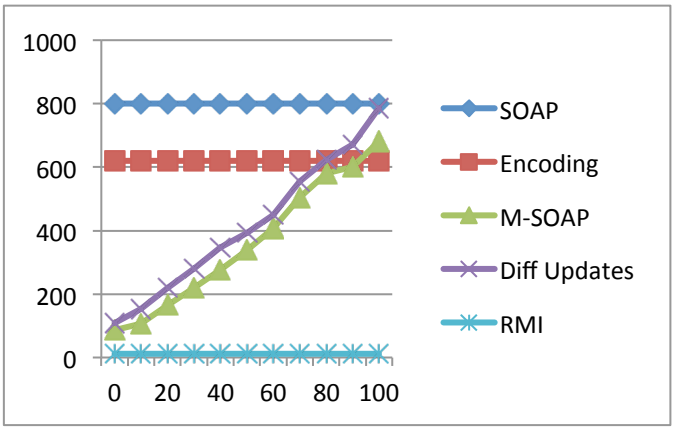

Figure 7 – Delay (horizontal axis in ms) vs. Data Variability (vertical axis in %-increments)

The results show that Differential Updates perform over 100% better than Encoding when the data variability is in the range of between 30% and 40% and over 300% better when the variability is 20% or less. Of course, both Encoding and Differential Updates should be used because, when combined, they lead to the best reductions in delays.

C. Overhead Delays

In this subsection, we report on the overhead delays of our mSOAP protocol. The mSOAP protocol includes overhead due to the following activities:

Differential Updates and Encoding at Server in Phase 1: In Phase 1 (when the web service is invoked for the first time), the server-side Update Manager creates a cache after receiving the reply message from the web service, identifies to-be-cached items in the reply message, and stores them in the cache. It also creates the encoding table and inserts, in the message, next to XML each tag its code – thus conveying to the client Update Manager the encoding information.

Differential Updates at Server in Phase 2: In Phase 2 (2nd and subsequent invocation of the web service), the server-side Update Manager compares the reply message generated by the web service with the content of the cache. For any list element in the message that has a value that matches the one that is cached, it is removed from the response message.

Encoding at Server in Phase 2: In Phase 2, after the Differential Updates method above is used to reduce the size of the reply message, the server-side Update Manager uses the coding table to replace any XML tags in the message with their codes.

Differential Updates and Encoding at Client in Phase 1: In Phase 1 (when the reply message from the first invocation of the web service is received), the client-side Update Manager creates a cache, identifies cached items in the reply message, and stores them in the cache. It retrieves the encoding information from the reply message and stores it in its data structures. It removes from the message any mSOAP protocol information.

Differential Updates at Client in Phase 2: In Phase 2 (receiving the reply messages from the 2nd and subsequent invocation of the web service), the client-side Update Manager compares the received reply message with the content of the cache. For any list element in that cache that is not appearing in the message, it is inserted in the message.

Decoding at Client in Phase 2: In Phase 2, the client-side Update Manager uses the Encoding table to replace any codes in the response message with their corresponding XML tags.

mSOAP Network: Delay of transferring the mSOAP message over the network between the client and the server is also shown for comparison purposes.

Processing delays for the activities in the Phase 1 of the mSOAP protocol, on the server and the client, are shown in Figure 8. The figure also shows the network delay due for transferring Phase 1 response message, i.e., the first response message, from the web service to the client. The transfer delay is about the same as for the regular SOAP protocol, shown in Figure 7 – inclusion of the Encoding information does not have a significant impact on delaying the first message. In a list of stock ratings stored in the web service reply message, total number of unique tags is smaller in comparison to the total number of tags as tags are repeated in list elements. The processing delays at the client and the server are below 100 ms and are low relative to the data transfer delay. In summary, in its Phase 1, the mSOAP protocol has low overhead processing delays and low impact on data transfer delay.

The overhead delays and data transfer delays in the mSOAP's Phase 2 are shown in Figure 9, which appears at the end of this paper, as a function of the variability of data.

First we discuss the delays due to Differential Updates. As the variability of the data increases, the overhead delays of Update Managers, at both the client and the server, do increase slightly. In both cases the Update Manager manipulates each element of a list being processed. On the server, the Update Manager determines whether a list element can be removed if its value has not changed from the previous invocation of the web service. On the client, the Update Manager determines whether a list element in the cache needs to be inserted in the message. The slight increase due to variability is that our coding performs slightly more work if the value of a list element has changed from the previous invocation of the web service.

When considering overhead delays due to encoding on the server and decoding on the client, Figure 9 shows that they do not vary. The Encoding method has been instrumented without the usage of Differential Updates and, consequently, all XML tags are encoded on the server and decoded on the client. It should be noted that incorporating encoding/decoding of XML tags within the Differential Method is simple and with negligible execution delay. As list elements are processed by either of the client or server Update Managers in the Differential Updates method,

encoding/decoding simply causes a direct table look-up and hence incurs minimal delay.

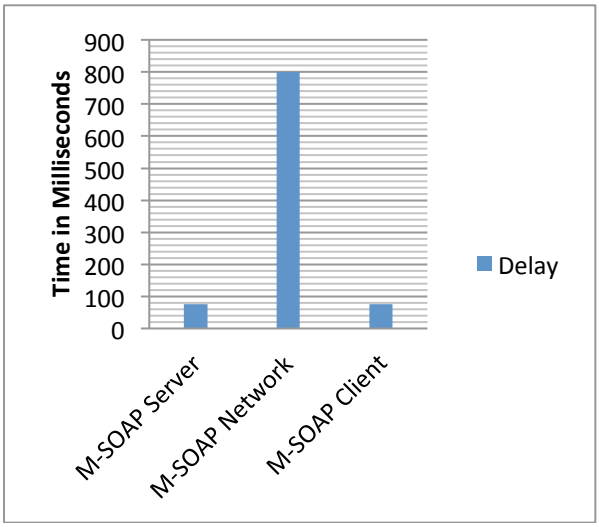

Figure 8 – mSOAP Overhead Delay in Phase 1

The figure also includes, for comparison purposes, the network delay of transferring an mSOAP reply message – transferring the reply message is not overhead as the message is transferred in any case. From the figure it is clear that the network overhead delays dominate the overhead delays. We expected that, in a realistic environment of mobile devices invoking web services, the network delays would be far more dominant than in our experimental set up. Consequently, reducing the size of XML messages is highly beneficial and comes at a low overhead cost. Reducing the size of messages exchanged by a mobile device has the benefit, in addition to reducing delays, of also reducing the power consumption due to communication. However, reductions of power consumption or usage of caches for the purposes of availability in case of communication interruptions are out of scope of this paper.

VIII. Related Work

There has been much research on the usage of compression and encoding techniques applied to reducing the size of XML documents and XML-based communication. For instance, one of the most popular methods is Gzip that enables compression of XML message to reduce its size (Deutsch, 1996)).

In (Natchetoi, 2007), XML compression uses both context and acceptable loss (of unnecessary task data) to produce more efficient mobile communications. New concepts include a scheme for contextual dictionary management (i.e. dictionary construction, update, and transfer) combined with a separation of dictionary and data messages.

Encoding of XML tags into binary codes and transferring the codes instead of tags has been proposed in (Naresh, 2005) together with formally incorporating the encoding in the communication protocol, which is called the Wireless SOAP. We incorporate encoding into our mSOAP protocol in addition to using Differential Updates/Caches. Various XML encoding methods were proposed, such as XMill, Millau DDT, WBXML, WSOAP and TDXML ((Girardot, 2000), (Ng, 2006a, 2006b)). XMill and TDXML were found to be good encoding schemes when applied for SOAP-XML communication (Ng, 2006a); however, the reduction in the message sizes of exchanged documents is at the cost of increased processing requirements and complexity on the client when reconstructing the original form. TDXML describes the SOAP messages using indexing of tags (Ng 2006b). This is useful when replication of tag names occurs within the same message as using tag indices instead of tags reduces the message size.

Differential Encoding and Differential de-serialization are two approaches used to obtain a difference document from a previously sent SOAP message ((Werner, 2004), (Suzumura, 2005)). The basic assumption underlying these two approaches is that the majority of SOAP envelope remains the same when communicating between the client and the server. Before transferring a message, a difference is calculated, between the previously transferred message and the message to be transmitted, and it is this difference that is transferred. The receiver reconstructs the original message from the received difference and the cached message that was previously received. There are a number of key distinctions between their methods and our Differential Updates method. In their methods, the size of the difference document is independent of how much data has actually changed and, furthermore, slight modifications made in random parts of the message may result in the size of the difference document that is almost equal to the size of the original SOAP message and thus achieving low reduction in the size of transferred data. This a fundamental difference when compared to our method, in which a few changes in the reply message would result in high reduction in the size of transferred data. Another distinction is that their methods concentrate on reduction of message sizes in the whole SOAP message envelope while we concentrate on reduction of the Body of the SOAP envelope.

Caching of data for web service communication has also been researched for the use with mobile devices. An example of a useful approach is presented in (Xin, 2007), in which the authors used dual side caching in web servers and Personal Digital Assistants to improve availability in face of loss of connections. Their basic technique is to cache the SOAP response messages in the client so that the response messages would be available in case of a connection loss or fluctuation in bandwidth. We also cache response messages from a web service but for the purposes of reducing the size of responses from the web services when it is invoked again. Although we have not addressed the issue of using our cache for availability in case of lost connections, it could be used for that purpose.

There is much research on software caches in client-server or n-tier architectures, with many objectives ranging from providing transactional guarantees, through providing consistency based on relaxed consistency models, to providing availability (e.g., (Garrod, 2008), (Haas, 1999), (Oh, 2005) and (Pitoura, 2007), just to name a few). It should be noted that although the hardware for mobile devices is improving at

a tremendous rate, mobile devices still have limitations in terms of memory, processing power, communication bandwidth, and, in particular, power consumption. Consequently, caches targeted to mobile devices, in general, are smaller and simpler than caches targeted to servers or desktops. This is also the case in our proposal. The client cache is limited in size and the work of the client's Update Manager is not demanding.

IX. Summary and Conclusions

We created Differential Caches and the Differential Updates to reduce the size of response messages returned by a repeatedly invoked web service. A novel cache-based system to speedup the transfer of data is described. The communication between the client and the server is through the mSOAP protocol, which is a transparent extension of the SOAP protocol. We reduce the size of reply messages, from the web service to a client application, by removing from the messages data elements that are the same as in the previous response message. When the message is received by the client, the missing data elements are re-inserted into the message from the cache before the reply message is forwarded to the application.

The advantage of our proposed method is that it is transparent to the web service and application developers as they need no knowledge of it. The method is provided automatically, without affecting the functionality of the web services or applications.

We described two cache designs. In one, the server maintains a cache for each application invoking a web service. The second design is based on a shared cache, which utilizes timestamps for replies and for cached data in order to ensure correctness of reconstruction of reply messages. The mSOAP protocol supports not only our Differential Caches/Updates method but also other optimization techniques, such as encoding, that can be used to reduce communication delays.

We created a research prototype and performed experiments in order to evaluate the trade-off between the potential benefits and overhead. The processing overhead on the server and the client is more than outweighed by the potential benefits of reducing the communication delay. The research prototype included the mSOAP protocol that supported encoding to reduce the size of exchange messages in addition to our method. Experiments show that a speedup of up to 800% is possible using our method in comparison to SOAP communication, depending on the data variability. Furthermore, experiments also show that Differential Updates perform over 100% better than Encoding when the data variability is in the range of between 30% and 40% and over 300% better when the variability is 20% or less.

Our method, by design, does not affect the development and implementation of web services or applications. However, in future, it may be advantageous in some cases to include Differential Caches within web services. Besides further reducing delays, data stored in the server cache may, in certain situations, be exploited to avoid retrieval from the data stores/DBs. Additionally, synchronization issues to the shared cache need to be addressed. A further expansion of the mSOAP protocol will be to incorporate the usage of cached data on the client for availability when connection between the client and the server is not available.

References

Apte, N., Deutsch, K., & Jain, R. (2005). Wireless SOAP: optimizations for mobile wireless web services. Special interest tracks and posters of the 14th international conference on World Wide Web (pp. 1178-1179). ACM.

Axis-dev (2010). Axis User's Guide. Retrieved 4 1, 2010, from Web Services Apache Axis: http://ws.apache.org/axis/java/user-guide.html

Cheng, S.-T., Liu, J.-P., Kao, J.-L., & Chen, C.-M. (2002). A New Framework for Mobile Web Services. Symposium on Applications and the Internet Workshops (SAINT 2002 Workshops), 2002 , 218.

Deutsch, P. (1996). GZIP file format specification version 4.3. RFC 1952, Aladdin Enterprises, May 1996. http://www.ietf.org/rfc/rfc1952.txt.

Devaram, K., & Andresen, D. (2003). SOAP optimization via client-side caching. Proceedings of the First International Conference on Web Services (ICWS 2003) (pp. 520-524). Citeseer.

Garrod, C., Manjhi, A., Ailamaki, A., Maggs, B., Mowry, T., Olston, C., et al. (2008). Scalable query result caching for web applications. Proceedings of the VLDB Endowment , 1 (1), 550-561.

Girardot, M., & Sundaresan, N. (2000). Millau: an encoding format for efficient representation and exchange of XML over the Web. Computer Networks, Elsevier , 33 (1-6), 747-765.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., et al. (2007, April 27). SOAP Version 1.2 Part 1. Retrieved February 19, 2010, from W3C Recommendation: http://www.w3.org/TR/soap12-part1/

Haas, L., Kossmann, D., & Ursu, I. (1999). Loading a cache with query results. VLDB 1999 (pp. 351-362). Citeseer.

Ion, I., Caracas, A., & Hopfner, H. (2007). MTrainSchedule: Combining Web Services and Data Caching on Mobile Devices. Datenbank Spektrum , 21, 51-53.

Liu, F., Chou, W., Li, L., & Li, J. (2004). WSIP--Web Service SIP Endpoint for Converged Multimedia/Multimodal Communication over IP. Proceedings of the IEEE International Conference on Web Services, 2004 (pp. 690-699). IEEE Computer Society.

Liu, X., & Deters, R. (2007). An efficient dual caching strategy for web service-enabled PDAs. Proceedings of the 2007 ACM symposium on Applied computing (pp. 788-794). ACM.

Natchetoi, Y., Wu, H., Babin, G., and Dagtas, S. (2007). EXEM: Efficient XML Data Exchange Management for Mobile Applications. *Information Systems Frontiers,* Vol. 9, 2007, pp. 439-448.

Ng, A. (2006a). Optimising Web Services Performance with Table Driven XML. Australian Software Engineering Conference (ASWEC'06) (pp. 100-112). IEEE Computer Society.

Ng, W., Lam, W., & Cheng, J. (2006b). Comparative analysis of XML compression technologies. World Wide Web , 9 (1), 5-33.

Oh, S., & Fox, G. C. (2005). HHFR: A new architecture for Mobile Web Services Principles and Implementations. Community Grids Technical Paper .

Pitoura, E., & Chrysanthis, P. (2007). Caching and replication in mobile data management. IEEE Data Engineering Bulletin, Citeseer , 30 (3), 13-20.

Scholz, A., Buckl, C., Kemper, A., Heuer, J., & Winter, M. (2008). WS-AMUSE-web service architecture for multimedia services. Proceedings of the 30th international conference on Software engineering} (pp. 703-712). ACM.

Srirama, S. N., Jarke, M., & Prinz, W. (2006). Mobile web service provisioning. Advanced International Conference on Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services, (pp. 120-120).

Suzumura, T., Takase, T., & Tatsubori, M. (2005). Optimizing Web Services Performance by Differential Deserialization. 2005 IEEE International Conference on Web Services, 2005. ICWS 2005. Proceedings, (pp. 185 - 192).

Varshney, U., & Vetter, R. (2002). Mobile commerce: framework, applications and networking support. Mobile Networks and Applications , 7, 185-198.

Werner, C., Buschmann, C., & Fischer, S. (2004). Compressing SOAP messages by using differential encoding. IEEE International Conference on Web Services, 2004. Proceedings (pp. 540-547). IEEE Computer Society.

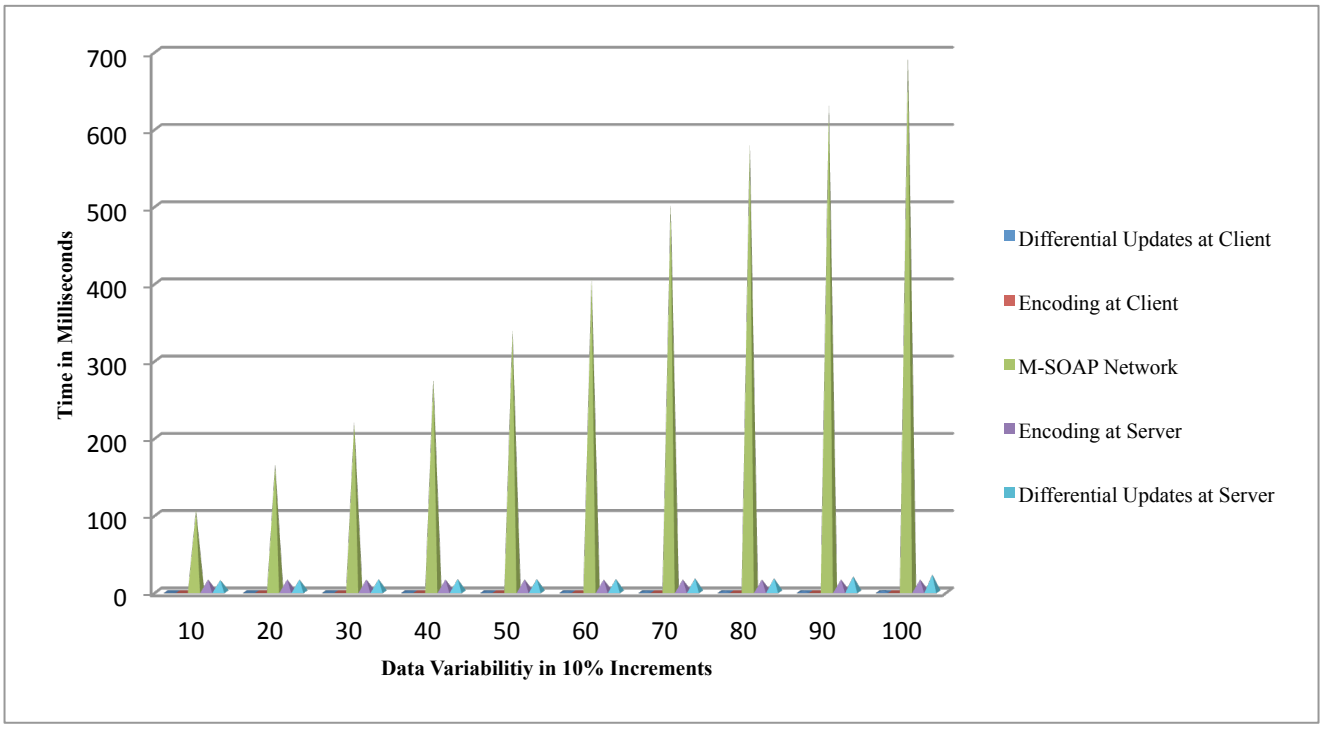

Figure 9 – Overhead Delays in Phase 2