A Review of SOAP Performance Optimization Techniques to Improve Communication in Web Services in Loosely Coupled Systems

Mutange Kennedy Senagi[1], Okeyo George[2] and Cheruiyot Wilson[3], Sati Arthur[4] and Kalunda Jades[5]

**[1,4,5] Information Technology, Dedan Kimathi University of Technology
Nyeri, Kenya**

**[2,3] Computing Department, Jomo Kenyatta University of Agriculture and Technology
Nairobi, Kenya**

Abstract

Web services (WS) implements Service-Oriented Architecture (SOA). WS extends World Wide Web (WWW) infrastructure. This provides a means of integrating software applications in loosely coupled distributed systems. WS communication is facilitated by Simple Object Access Protocol (SOAP). SOAP offers a simple and lightweight mechanism for exchanging structured and typed information among peers in a decentralized, distributed computing environment. However, SOAP's transmitted data is represented in XML. XML documents are huge in size and verbose (highly redundant), and processing of XML information and its conversion to and fro memory data types are some of the major hindrance in performance for high performance applications. This survey paper gives an insight of previous researchers' contributions on techniques used in optimizing SOAP in communication in WS in terms of bandwidth utilization and throughput. To optimize SOAP, several techniques covered include: client side caching, differential serialization, SOAP binding, compression, server side caching, and differential deserialization.

Keywords: *SOA, web services, SOAP, XML, WSDL, and SOAP performance evaluation.*

1. Introduction

SOAP is a protocol that exists in the web service (WS) architecture. SOAP was coined in 1998 and adopted by World Wide Web Consortium (W3C) in 2000. SOAP does packaging and exchange of messages in loosely coupled systems. SOAP messages are XML based; SOAP messages are packaged in XML documents [1]. In messaging, SOAP depends on several ubiquitous protocol and data formats. Among them being XML and HTTP. However, XML has a verbose and huge structure; it has redundant textual characteristics and uses tags to delimit data. Therefore, processing and conversion of XML data to and fro memory data types are one of its major performances undoing in high performance applications [2]. In this survey paper, we will address SOAP optimization techniques addressed by researchers and how they improved SOAP performance in communication in WS in loosely coupled systems.

This paper is organized as follows. In section 1.1 we will discuss what a service is in the context of SOA. We will then discuss web services stack in section 1.2. In section 1.3, we discuss SOAP message structure. In section 1.4 we discuss why this survey paper is interested in SOAP. In section 1.5 we discuss SOAP some of the performance evaluation techniques and some of the tools used in evaluating various performance metrics. In section 2 we discuss various SOAP optimization techniques classified as client side, communication channel, and server side. Section 3 concludes this survey research paper.

1.1 Services in Service Oriented Architecture

SOA is a loosely coupled architecture designed to meet various business needs of organizations. There has been paradigm shift from Object Oriented Systems Analysis Design (OOSAD) in the 1980's to Component-Based Development (CBD) in the 1990's and ultimately we have SOA. This is a transmutation from the remote invocation of objects in OOSAD to message passing between services in SOA. Good software engineering practices recommend separation of business component from the user interface component as opposed to the traditional approach in which both were a single entity. Nonetheless, with SOA, particularly WS, business functionality is implemented through exposure of services that can be consumed by heterogeneous applications outside the control of the system [3].

A service is a well-defined business functionality that can be consumed by a different application [4]. SOA contains a set of linked services that can be accessible over the network or internet [5]. As shown in Figure 1.1, SOA comprises of three main components: service consumer, service provider, and service registry. The service consumer is an application, a module or another service that requires a service. The service consumer initiates enquiry from the registry, binds the service over the transport component and executes the service function. It executes the service as per the interface contract. The service provider, on the other hand, is an addressable network entity that accepts and executes requests from the consumer. The service provider publishes its services and interface contract to the service registry; so that the service consumer can discover and access the services. The last component, the service registry, is the enabler of service discovery. The service registry contains a repository of available services. It allows interested service consumers to look up for services of available service providers. Operations in SOA include: publish, find and bind and invoke. Publish involves a service publisher making a service accessible. Find operation involves a service consumer querying the service registry for a service that meets its benchmarked needs. Binding and invoke operations involves the service consumer retrieving a service description from the service registry and entreating the service as per the service description [6].

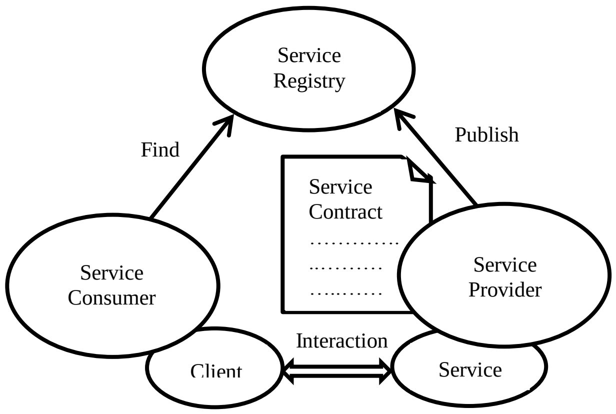


Fig. 1.1: Service-Oriented Architecture roles. Adopted from [16]

SOA is being adopted by many programmers as a way of integrating heterogeneous software systems and providing different services thus building dynamic systems that are loosely coupled [7]. Some of the technologies that implement SOA include: Common Object Request Broker Architecture (CORBA) [8], Java Remote Method Invocation (RMI) [9], Component Object Model (COM) [10] and web services [11]. Java RMI, CORBA and COM are used in developing highly coupled systems where client and server are dependent on each other thus monolithic in nature [2] [12] [13]. WS develop loosely coupled systems that are platform independent and work in heterogeneous systems [2] [13]. Moreover, WS is the most recommended technology for realizing SOA because of its ease of use, modularity, multiple vendor support, compose-ability, low cost, and commonality with the SOA model [14].

1.2 Web service

A WS is a software system designed to underpin interoperability of machines within a network [11]. This has led to tremendous rise in the usage of WS [15]. The main goal of WS is to have a standard way of exchanging information between applications [13]. Its roles complement those of SOA as shown in Figure 1.1. WS allow interoperability between heterogeneous systems though it’s quite complex. Nonetheless, Snell et al. (2001) provides a simplified and elaborate WS stack as shown in figure 1.2.

The WS has five layers namely: discovery, description, packaging, transport, and network. Each layer plays a specific role. In the discovery layer, discovery is performed by end users. It’s the act of locating a resources description from the service registry thus providing an easy publish/find functionality. Service discovery in WS is handled by Universal Description Discovery and Integration (UDDI). The description layer describes the public interface of a WS. Description in WS is done by Web Services Description Language (WSDL), Resource Description Framework (RDF) or DARPA Agent Markup Language (DAML). RDF and DARPA are rich but complex in describing WS than WSDL. Packaging layer does the packaging of messages before relaying them in the network. The message is usually in a format that all parties understand in the heterogeneous environment. SOAP or Representational State Transfer (REST) is responsible for the packaging of messages in WS. Transport layer transports data and enables application-application communication on top of the network. Some of the protocols involved include HTTP, TCP, SMTP, and Jabber. The network layer provides critical basic communication, addressing, and routing capabilities [16].

WS implementation can be done in SOAP or REST. SOAP adopts XML-based messages that are huge in size and parsing which are computationally expensive. In REST architecture, resources (data and functionality) are accessed using web Uniform Resource Identifiers (URIs) via simple well-defined HTTP operations [17]. SOAP causes relatively high network traffic, high latency and processing delays as compared to REST [13] [17]. In systems with limited resources like Mobile Operating Systems (MOS), REST is preferred to SOAP [18] [19] [20]

[21]. Nevertheless, SOAP can also be implemented in MOS [22]. SOAP enjoys the benefit of being more secure than REST. SOAP and REST have their own strengths and weaknesses, therefore, the choice to use one depends on application complexity, requirements and constrains etc. The software developer should choose wisely the technology to adopt [23] [24]. Notwithstanding, researchers are working around the clock to improve SOAP performance as discussed in details in section 2.

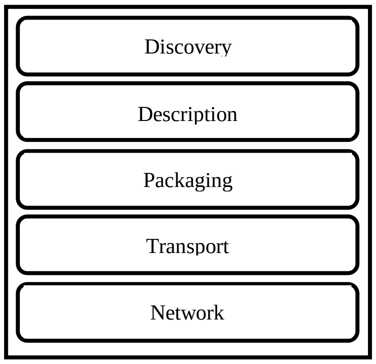


Fig. 1.2: The web service technology stack. Adopted from [16]

1.3 SOAP (Simple Object Access Protocol)

SOAP is a standardized de facto XML-based protocol for packaging, services invocation and exchanging messages in distributed systems aided by WS interfaces [12]. As shown in Figure 1.3, SOAP structure has four regions namely: envelope, header, body, and fault. SOAP message structure has four regions. The SOAP envelope <Envelope> is the root element in every SOAP message, and contains two child elements, an optional <Header> and a mandatory <Body>. The SOAP header <Header> is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is to be processed by SOAP nodes along the message path. The SOAP body <Body> is a mandatory sub-element of the SOAP envelope, which contains information intended for the fundamental recipient of the message. The SOAP fault <Fault> is a sub-element of the SOAP body, which is used for reporting errors [25].

SOAP protocol relies on HTTP or HTTPS in communication. However, SOAP can still ride on SMTP, and other compatible transfer protocols. The advantage of riding on HTTP is that, it is: firewall friendly, an open standard, and a universally accepted transfer protocol. SOAP messages are encapsulated within HTTP. HTTP is a universal standard in the WWW. The SOAP XML document is embedded in HTTP. Firewalls by default allow traffic through port 80 which HTTP uses in communication. This gives SOAP the power to be platform independent [26]. SOAP request and responses are via HTTP. SOAP uses the HTTP GET method for requests and HTTP POST method for both request and response. HTTP explores TCP/IP protocol for network transport because of reliability. Some researchers have explored HTTP binding on UDP which proved to improve performance although it is less reliable [27].

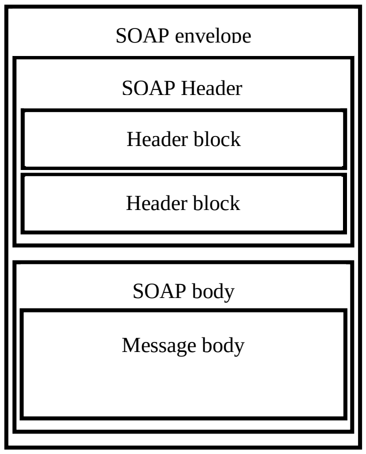


Fig. 1.3: SOAP message structure. Adopted from [26]

1.4 Why SOAP?

In SOA, WS provide a comprehensive solution for representing, discovering and invoking services in distributed systems. At the core of the WS, lie various XML-based standards including SOAP. SOAP is a protocol that ensures WS extensibility, robustness and interoperability between heterogeneous systems. However, SOAP has basically two major performance-related drawbacks:

- XML structure is huge and verbose (highly redundant) which results to high network traffic thus poor utilization of bandwidth and relatively high response time.
- Conversion of XML data to and fro memory data types cause a high computational burden leading to high latency and poor memory utilization.

1.5 SOAP Performance Evaluation Parameters and Tools

SOAP is a service oriented technology which cannot run away from Quality of Service (QoS). QoS can be defined as a set of techniques geared towards managing of resources [28]. Moreover QoS is a set of perceivable characteristics expressed in a user-friendly language with quantifiable parameters that may be subjective or objective. Software performance evaluation is an area of interest in QoS; which is also a field of concern in software engineering [29].

Performance evaluation emerged in the 1970's and 1980's as an important component in computer science. It involves use of accepted methods in measuring computer systems. The field of computer systems and software engineering focuses on specific components in computer science. These components can be evaluated in terms of their effectiveness. SOAP performance metrics involves a standard measure of SOAP performance indices. In the OSI (Open Standards Interconnection) model of communication in a network, SOAP metrics can be evaluated at multiple layers in the protocol stack, for instance, IP packets round trip time (network layer), and channel utilization (transport layer). There are other various performance evaluation parameters which include: throughput, good put, packet loss rate, and MAC layer retries [29] [30] [31] [32]. This research survey is interested in SOAP performance which is widely measured in terms of:

- Round trip time (responsiveness): It's the time required to traverse a network and coming back. Round trip time is measured in milliseconds (ms).
- Bandwidth/channel utilization. It measures utilization of a channel. It is the amount of data transmitted or received at a given time. Its measured in megabytes per second (mbps)
- Throughput: Measures the packets that are flowing out (e.g. requests) of a node/client. In WS, it can be measured in either megabytes per second (mbps) or requests per second (req/sec). It's usually measured at the server side [30].

Researchers in [32] did a study of web services testing tools using soapUI [33], JMeter [34] and Storm [35] in terms of their architecture, features, interoperability, software requirements, and usability. Moreover, they did a comparison of their throughput, response time and usability; only JMeter and soapUI support testing of throughput. Nonetheless, soapUI outperformed JMeter and Storm thus can be regarded as fastest tool in terms of response time, JMeter had better throughput than soapUI, and Storm had a very simple and easy to use interface. It was observed that response time values taken at 6:00 AM are most optimal [32].

Apache Bench can be used to test various metrics among them: throughput and response time [36]. Web services performance testing tools is a rich area of study that can exploited. Nevertheless, there are many vendors in the internet who have come up with tools that can test web services. Some of these tools include: Fiddler [37], NetMon [38], Wire Shark [39], and NeoLoad [40]. Software testing in complex systems can be very involving. Software performance profiling can be very essential in determining a software performance in terms of memory utilization, execution time etc. [41] [42]

The goal of this survey research is to provide a survey of techniques that can improve SOAP performance which are covered in the following section. The classifications are thematic: client side, communication channel and server side.

2. SOAP Performance Techniques

The dependence of SOAP on XML in messaging is the major hindrance in performance for high performance applications. Several researchers have made contributions on how to optimize SOAP performance in web services communication. This section covers various SOAP optimization techniques that are thematically classified as: client side in section 2.1, communication channel in section 2.2 and server side in section 2.3.

2.1 Client Side

A client is a computer that sends requests to the server; normally it is the end users computer. All the computing operations involved in client computer are said to exist in the Client Side. In this section we discuss client side caching technique in 2.1.1 and differential serialization (DS) discussed in 2.1.2 as client side optimization techniques.

2.1.1 Client Side Caching

Client side caching is the storage of data in the client side. Client side caching is a technique of improving SOAP performance in terms of improving response time. SOAP client side caching has been supported by several researchers [2] [43] [44] [45] [46]. Caching has been embraced solely to improve the amount of traffic and latency between the service and underlying data providers [2] [43] [46]. Client caching can store data temporarily within the internet browser or by a JavaScript data structure [46].

Data in SOA is categorized as server state and service data. Service state is data that concerns the state of the business process/service while service result is data that is delivered by the business process/service back to the presentation layer. Moreover, caching can be categorized as: client side caching, proxy caching, reverse proxy caching, and web server caching. Figure 2.1 shows different types of caching.

In client side caching, data is stored by the client side browser temporarily on the local disk or browser's internal memory. Its advantage is that data cached on the local client can be easily accessed and reduces network traffic while its disadvantage is that cached data in the client is browser dependent and is not shareable. Proxy caching uses a proxy server that stores cached data between the client and the web server. This data cached in proxy server can be shared among clients thus leverages the weakness identified in client side caching. Its advantage is that it fulfills all requests from web page without sending them out to the actual web server over the internet, resulting in faster access and reduced traffic. Its disadvantages include deployment and infrastructure overhead to maintain the proxy servers. In reverse proxy caching, the proxy is placed in from of the web server. The proxy responds to the most frequent request and passes others to the web server. As much it reduces the number of request directed towards the web server, its position in front of the server increases network traffic. In web server caching, the web server stores its own cached data. It improves the performance of a site by decreasing the round trip of data retrieved from database or other servers, reduces server load, and reduces bandwidth consumption [43].

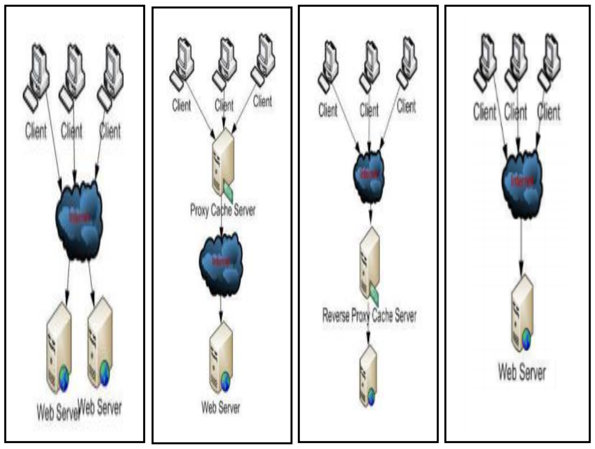


Fig. 2.1: Client side caching, proxy caching, reverse proxy caching and web server caching respectively. Adopted from [43]

After profiling the client side, researchers in [44] note that around 40% of execution time is spent in XML encoding which involves serializing and marshalling the SOAP payload before transmitting it to the server. Similarly, researcher in [43] [46] also noted an improved performance using client side caching. Therefore, clients that send the same request to the server frequently consume a considerable amount of time in encoding XML. To overcome this challenge, caching such request(s) not only saves a considerable amount of execution time in recreating the payload, but also the time involved in trips to fetch data from the sever. The client always checks if the request was previously indexed and cached on the client side before sending it to the server. If the request was cached, it does a simple file I/O operation to fetch the payload from the client side cache [44]. In as much as [44] used RPC-style in WSDL 1.1 binding which is an inefficient SOAP binding style as discussed in section 2.2.1, an evaluation of SOAP caching on the client side showed an improved performance by a remarkable 800%. This resulted to better performance than the traditional binary Java RMI which outperformed SOAP as discussed in [47]. Figure 2.2 shows the round trip results of SOAP with Java RMI.

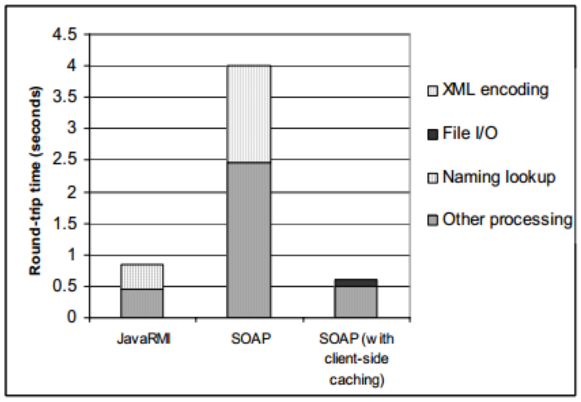


Fig. 2.2: Comparison of SOAP (with client-side caching) with Java RMI and the traditional SOAP. Adopted from [44]

Some of the challenges involved in client side caching include: how frequent the data needs to be updated, the data being user specific or application-wide, and what mechanism to use to indicate that the cache needs updating [43] [44] [46] [48]. Researchers in [46] [44] noted that proper indexing and time stamping can be used to verify its validity. Consequently, [44] notes that data in the client side can be updated not only by deleting and renewing data period of time, but also by updating last modified timestamps by use of a cache provider. Updating last modified timestamps is much better because it imposes less overhead as compared to reloading the entire data set. Research in [46] suggests hybrid reverse caching strategy in web caching. Hybrid reverse caching caches data structures rather than static values. This caching can be built on unified data stores to eliminate redundant and duplicate data. Client side caching performance can be enhanced further by doing more research on caching algorithms that can further improve fetching and serialization of XML data.

2.1.2 Differential Serialization

Differential serialization (DS) avoids serializing of the whole message structure. Serialization of sent/outgoing messages involves conversion of in-memory data types to SOAP XML-based ASCII string formats, and then packing this data into message buffer; this counts as one of the major performance bottlenecks of SOAP performance as it accounts for 90% of end-to-end message time. The client that sends the requests is called the bSOAP. In DS, once a serialized message has been sent by a SOAP communication end point, the client saves the message so that it can be reused by future subsequent messages as a template. Subsequent messages that have the same structure or are identical can reuse the structure and avoid the serialization overhead involved in regenerating the structures from scratch. This works best if the same client sends a stream of similar messages. This technique improves response time [2] [15] [49] [50] [51] [30].

The steps to follow to make DS a success is: tracking data changes and overwriting only those values that have been changed since the last sent message, expanding the serialized message to accommodate large serialized values, storing the message in chunks and padding them with white spaces to reduce the cost of expansion, and overlaying the same memory region with different portions of the same outgoing message to reduce memory consumption. The steps outlined above demonstrated a best case performance of ten times faster. The study also showed that send times reduced by a factor of five only when parts of the message were to be re-serialized. In designing the DS, when comparing the outgoing message to the saved templates the different matching possibilities are [49]:

- Message content matching: This entails the entirely sent message being exactly the as the one sent from the client earlier; the client sends the message as it is.
- Perfect structure match: This entails the message having the same structure and size as an earlier message but having values of some field that have changed. In this case, the serialized message is replaced with the changed values only.
- Partial structure match: This entails the message having a structure but a change in size of the message as compared to an earlier message. Also, some of the values may not have matched. Unlike in memory base types, the serialized message template may be expanded or contracted to meet the requirements of the new message.
- First time send: This phase encounters the normal overheads involved in creating a serialized message from scratch, checking whether it exists amongst the saved templates and saving a pointer to it after it has been created.

From the different matching possibilities, researchers in [49] notes that partial structure match can be avoided using several techniques which are: stuffing, shifting, chunking, and stealing.

- Shifting: This involves expanding the message in memory when the serialized form of a new message exceeds its field width as shown in Figure 2.3. It involves shifting bytes in the template to make room for the new values then updating Data Update Tracking (DUT) Table accordingly. This is expensive because it entails memory moves, possibly memory reallocation, and updating DUT table.

```
…</w><x xsi:type='xsd:int'>1.2</x><y xsi:type=….
becomes
…</w><x xsi:type='xsd:int'>1.23456</x><y xsi:type=….
```

Fig. 2.3: Shifting technique in Differential Serialization. Adopted from [49]

- Stuffing: This involves adding extra white spaces in the serialized message to accommodate potential future updates that would otherwise require expansion as shown in Figure 2.4. The white spaces can be explicitly created when the template is created or after serializing a value that requires less space. This technique can avoid shifting which is an expensive technique.

```
…<y xsi:type='xsd:int'>678</y><z
xsi:type=…
can be represented as
…<y xsi:type='xsd:int'>678</y>□□□□<z
xsi:type=…
```

Fig. 2.4: Stuffing technique in Differential Serialization. Adopted from [49]

- Stealing: This reduces the costs of increasing field size by stealing extra spaces from neighboring fields instead of shifting entire portions of memory chunks. This technique is actually less expensive than shifting. Performance of stealing depends upon the Halting Criteria (tell when to stop stealing) and direction (tell left, right or back-and-forth of memory chunks).

```
…'>678</y><z xsi:type='xsd:double'>1.166</val>□□□□□
y can steal from z to yield…
…'>677.345</y><z xsi:type='xsd:double'>1.166</val>□
```

Fig. 2.5: Stealing technique in Differential Serialization. Adopted from [47]

- Chunking: This involves storing messages in potential non-contiguous memory chunks to limit the impact of the expensive Shifting.

Clients that send the same message frequently can maximize the advantage of DS in improving performance of that system. DUT Table comes in handy to track whether a program has changed data items in new messages since the last serialized SOAP messages. DS had an impact of up to 17% improvement [49]. However, in as much as [52] proposed an optimized version called XSOAP that used a new XML parser specialized for SOAP arrays, [53] notes that, in scientific grid computing (an area of high performance computing that is adopting the web service architecture), sending scientific data e.g. large arrays of floating point numbers and complex data types via standard implementation of SOAP is expensive.

2.2 Communication Channel

The SOAP XML document is embedded in HTTP as the default transport protocol. SOAP messages can be transported in SMPT and FTP among other protocols. HTTP uses port 80 as the default communication port. By default SOAP uses HTTP-GET or HTTP-POST protocol to communicate in WS [26] [27]. The wire format of data in communication channel affects SOAP performance [44] [54]. In this section we discuss SOAP binding style in section 2.2.1 and compression techniques in section 2.2.2 as techniques that improve SOAP communication. These techniques are discussed as follows.

2.2.1 SOAP Binding Style

In section 1.2, we discussed the web service stack which contains the description layer. The description layer is responsible for describing the public interface of a specific web services. Services are exposed on the public interface of a web service. Service description (location and methods exposed) is handled by Web Service Description Language (WSDL). Other approaches include the W3C's Resource Description Framework (RDF) and DARPA Agent Markup Language (DAML) which provide a much rich capability but very complex to describe web services than WSDL [7].

WSDL is a model that provides an XML format for describing WS in the web community; this ensures interoperability in heterogeneous systems. As per WSDL 1.1 [55] standards, the document structure of the XML has two sections abstract and concrete. The abstract elements (Type, Message, and PortType) defines WS interface while the concrete section (Binding and Service) describes how abstract interface maps messages on the wire [7] [55] [56]. All the WSDL 1.1 elements include:

- Type: This is a container for the schema type definitions.
- Message: This defines an abstract message that serves as the input/output of an operation. An operation is a message exchange; a focal point of a service interaction
- PortTypes: It's also known as Interfaces. It's an abstract set underpinned by one or more endpoints. It describes a function signature (operation name, input parameters, and output parameters) in a Message. An endpoint defines a combination of an address and a binding e.g. URI
- Bindings: This is a concrete protocol and data format specification for a particular PortType.
- Services: This is a collection of related network endpoints. An endpoint is a port

This research is interested on the binding element. Binding defines the message format and protocol details for operations and messages as defined by a particular PortType. In WSDL 1.1, binding has two attributes which include: style and use. The default style of the service is either RPC or document and the default transport protocol (HTTP) while in the communication channel [55]. The styles are discussed as follows:

- Document-style (previously called message-style) in SOAP dictates that the body contains an XML document, and the message part specifies the XML elements.
- RPC style in SOAP dictates that the body contains an XML representation of a remote procedure being invoked and the message parts representing the parameters to the method.

The use attribute specify the encoding to be used to translate the abstract message parts to concrete representations. It has two possible values are encoded or literal.

- In encoded, abstract definitions are translated to a concrete format by using the SOAP encoding rules.
- In literal, the abstract type definitions turn to be the concrete definitions, that is, you can simply inspect the XML Schema type definitions to validate the concrete message format.

The style and use attributes forms four possible combinations called binding styles, common once being RPC-encode as shown in Figure 2.7 and document-literal as shown in Figure 2.8 [7].

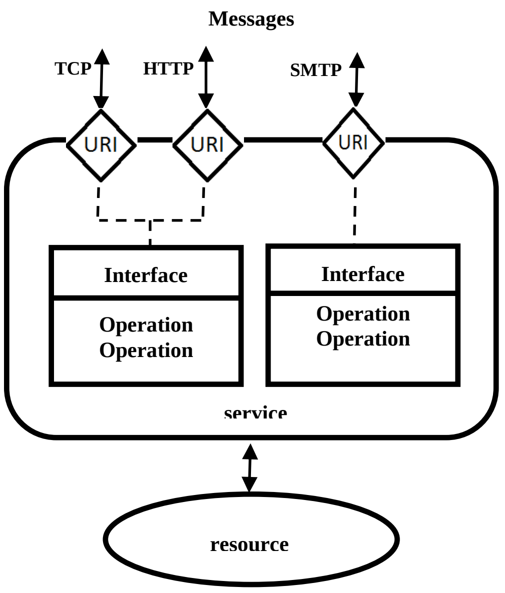


Fig. 2.6: WSDL Interface and bindings. Adopted from [7]

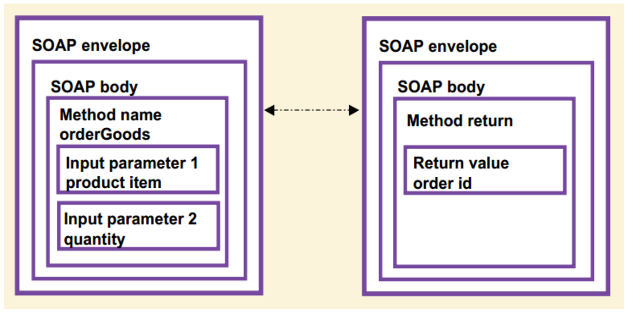


Fig. 2.7: SOAP document-literal call. Adopted from [26]

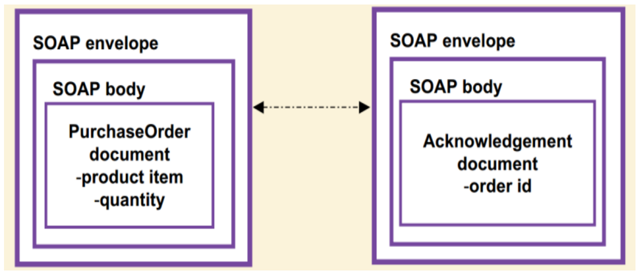


Fig. 2.8: SOAP RPC-encoding call. Adopted from [26]

Several researchers have done research on the different binding styles and their effects on performance on SOAP in communication. RPC-encode have more overheads than document-literal [7]. As much as document style had its own short comings, [7] [57] recommended adoption of document-literal over RPC style in a bid to improve performance. Java which had document style (MTOM technology enabled) showed that, web service using RPC style requires 15% more time as compared to document-literal style [58]. Moreover, [59] notes that test client-side experiments built on document- literal encoding style was faster than previous implementation using RPC. The research findings in [58] notes that RPC style requires 15% more than document- literal as shown in figure 2.9.

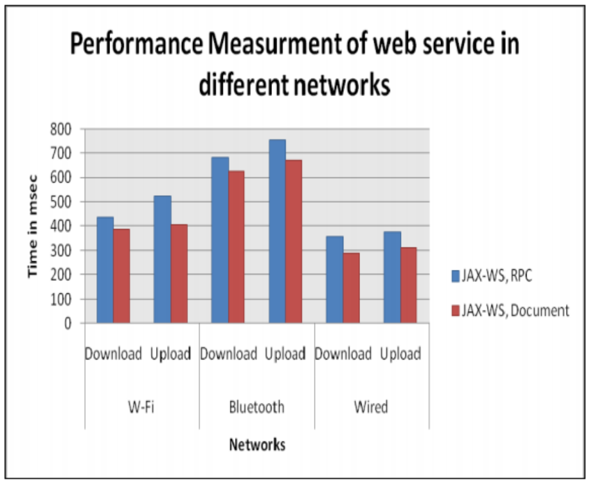


Fig. 2.9: Performance measurement of Web service in different networks. Adopted from [58]

WSDL 2.0 [60] is a later version of WSDL 1.1. WSDL 2.0 comes with certain features and language elements changed and expanded. For example the definitions element is renamed to descriptions, portType element is renamed to interface, port element is renamed to endpoint, and message element is discarded. The message element defined RPC (parameter driven) and message (document type) communication. Due to the limited expressive powers of RPC in message element, WSDL 2.0 discards it altogether and simply allows an operation to reference a type (such as an XML schema element) directly [61].

WSDL 2.0 component model is a set of components with attached properties which collectively describe a WS. Components in WSDL 2.0 are typed collections of properties that correspond to different aspects of WS. Components in WSDL 2.0 are serializable in XML 1.0 format but are independent of any particular serialization of the component model. WSDL 2.0 components include: description, element, type, interface, interface fault, interface operation, interface message reference, interface fault reference, binding, binding fault, binding operation, binding message reference, binding fault reference, service, endpoint, and extension component [52]. WSDL 1.1 the predecessor of WSDL 2.0 has lots of restructuring. Some of the new alterations might be beneficial upon its full understanding [61].

2.2.2 Compression

Several researches have supported compression as a promising solution to improving the huge verbose XML messages in SOAP [2] [22] [43] [44] [59] [62] [63]. Compression improves bandwidth utilization and response time of SOAP messages. Compression has its tradeoffs e.g. extra compression processing time. In the research [44] these tradeoffs were not beneficial. However, recently with the increased hardware processing capabilities, these tradeoffs are beneficial as is not as costly as increasing bandwidth which is widely under constrains [22]. Different compression algorithms have different compression ratio and different compression time for the same XML file(s) [45]. An attempt in [44] to compact XML tags to reduce the length of the XML tags names had negligible effect on encoding. The research [44] further suggested that other than the data an XML message contained, the major cost of the XML encoding/decoding is in its structural complexity and syntactic\ elements. Research in [63] [59] notes that XML files are highly redundant thus lossless compression algorithm works out best to achieve better compression ratios. Lossless compression algorithm exploits statistical redundancy to represent sender's data more concisely without errors. However, [63] [59] notes that lossless compression will not work for high entropy (high disordered) data e.g. already compressed data, random data or encrypted data as it will result in expansion rather than compression. Lossless compression algorithms include Gzip, Bzip2, Fast Infoset (FI), Efficient XML Interchange (EXI) etc. WS-security performance can be an interesting area to explore.
Experiments set up in [63] to compare the best compression algorithm between EXI, FI, and Gzip. It was indicated that FI performed the poorest. EXI showed slightly better compression ratios and response time than Gzip. However, [63] recommended Gzip compression algorithm in disadvantaged network. EXI [64] showed promising better performance outcomes than Gzip although it is still under open source test and test [63], a commercial version is yet to be released.

In [62] they focus on compression on textual data. They used an algorithm that works in three steps: removal of white spaces, compressing data to UpperCamelCase then decompress compressed data. Experiment in [62] had significant performance gains of up to 22% in bandwidth utilization. The algorithm works in small and large sizes of messages. Nevertheless, experimental results in [62] show that use of Gzip compression algorithm further improves bandwidth utilization as data integrity is observed. In multimedia data, a detailed analysis of multimedia streaming and compression is tackled in [65] [66]. This survey paper is interested in textual data compression.

Researchers in [15] did an evaluation of performance of Gzip and Bzip2 compressors by doing a comparison against three XML compressors (XMILL, xmlppm and XBXML). The methodology involved building an XML tree and converting it into a binary tree then encoding the XML tags by Fixed Length and Huffman techniques. This eventually removes all the closing tags thus saving the opening tags and data leaves of the created tree hence reduces the size of the messages sent and received. Nevertheless, in the experiments, out of the 160 messages that were equally divided into four groups in terms of message size categorizes data as small messages (140-800 bytes), medium massages (800-3000 bytes), large messages (3000-20000 bytes), and very large messages (20000-55000 bytes).

In the results of XML (uncompressed), XML (Bzip2), XML (Gzip), XMILL (Bzip2), XMILL (Gzip), XMILL (ppm), xmlppm and wbxml, evaluations shows Gzip compression was more effective than Bzip2 by achieving better compression ratio but XMILL (ppm) outperforms Gzip and Bzip2. The findings are shown in Table 1. Their experiment conclusions reveal that Huffman encoding was the most efficient for large and very large documents while Fixed Length encoding was found to be efficient for small documents. The compression trends observed in Table 1 can be attributed to the fact that look up tables are usually created in lossless compression techniques. Look up tables' aid in mapping of symbols to binary codes during compression process. Therefore, lookup tables in large documents consume a small space as compared to encoded data while in small documents the lookup table tends to be larger than the encoded data. This explains why a high compression ratio is exhibited in large documents as compared to small documents.

The research [2] [22] [63] [62] [63] [59] [44], support the fact that compression has a deep impact in not only reducing the response time, but also the improving bandwidth utilization hence increases the performance of SOAP based applications. Furthermore, [67] categorizes data compression algorithms methods into three:

- General Purpose Compression Algorithm: This include Gzip which is based on Huffman coding, LZ77 which is a substitution compressor and Bzip2 which is an implementation of Burrows-Wheeler block-sorting algorithm

- XML Aware Compression Algorithm: This explores the separation between XML markup and payload. The simplest in this category are substitution-based algorithms that work at the markup level. They include BXML, WBXML, XMILL, XMLPPM, ESAX etc.
- Schema-Aware Compression Algorithm: This defines their schemas in form of XSD or DTD files. They do not encode part of infoset which can be decoded by the receiving party.

Table1: Result compressed size of different SOAP messages using xmill, xbmill, Gzip, and Bzip2 compressors in addition to fixed and variable length encoding. Adopted from [15]

Size(b)	XMILL	XBMILL	Gzip	bzip2	Fix.-Len.	Huf.
146	121	167	116	128	84	89
211	161	193	154	165	120	128
313	208	261	192	214	171	183
593	294	351	278	301	268	285
817	324	399	322	337	306	324
1392	475	556	473	492	483	513
2544	671	784	706	776	667	688
3110	739	829	833	804	772	795
9775	1458	1462	1861	1528	1441	1370
16997	2090	1900	2822	2019	1927	1720
22728	2500	2226	3531	2393	2318	2008
36114	3893	3447	5164	3193	3236	2702
47800	4764	4008	6462	3856	4033	3339
53346	5175	4236	7150	4105	4411	3635

In as much as [67] argues that compression reduces response time, other factors about a compression algorithm need to be considered e.g. encoding and decoding time, the number of messages transferred, average compression time, number of processes involved, network components passing time, and geographical distance. Experiments in [67], prepared in .NET, studies Bzip2 and BXML algorithms considering their response time. From Table 2 and Figure 2.10, it's inferred that despite Bzip2 having a better compression ratio than BXML, BXML has a better response time because it has a lower compression time. Moreover, [67] notes that when using a compression algorithm, if the data size is more than a specific threshold it may increase the response time and improve performance otherwise it degrades the performance.

Table 2: The percentage of compression using Bzip2 and BXML algorithms. Adopted from [67]

Compressing Algorithm	Data Size(KB)				
	.7	32	65	100	130
Bzip2	69	30	16	11	6
BXML	81	64	55	45	46

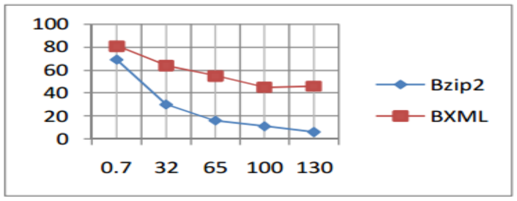


Fig. 2.10: The percentage of compression using Bzip2 and BXML algorithms. Adopted from [67]

Researchers in [22] gave a detailed algorithm on how a client and server communicate exploring text compression technique in a bid to improve performance in web-service-based applications. The algorithm showed text was reduced by 80%, meaning 80% less storage space was saved. Nonetheless, the text data being transferred required less time which translated to high performance for client-server application communication. As much the compression had some tradeoffs like processing time, it resulted to a general better performance of the system. Figure 2.11 shows an implementation of the compression. Text input from the client through the proxy is serialized as text-based SOAP message, compressed then sent to the server. The text is then decompressed, de-serialized then passed to the web service. The web service processes the request and returns the result which is later serialized and compressed before being sent to the client. Lastly, the client collects the text message which is de-serialized and decompressed through the proxy. The total processing time of a request is given in Eq. 1 where tser is time needed to serialize the request in xml format, ttr is the time needed to actually transfer the serialized request, tdeser is the time needed to de-serialize the xml text and tservproc is the time needed for processing the request and producing the results at the server side [22].

$$T_{total} = 2\cdot(t_{ser} + t_{tr} + t_{deser}) + t_{servproc} + t_{com/dec} \qquad (1)$$

The research done in [45] gives an assessment formula Eq. 2 to calculate T which is the transmission time a WS can be reduced. In the formulae: N is network speed in bytes per second, C is the computing speed of the device in units per second, compression algorithm requires Z computing units, and compression algorithm can compress the SOAP message out E byte. If the result of T is a positive value then the transmission performance of WS is improved. If it's a negative value, then it means that the compression algorithm does not improve the transmission performance of the algorithm.

$$T\text{ (seconds)} = (E \div N) - (Z \div C) \qquad (2)$$

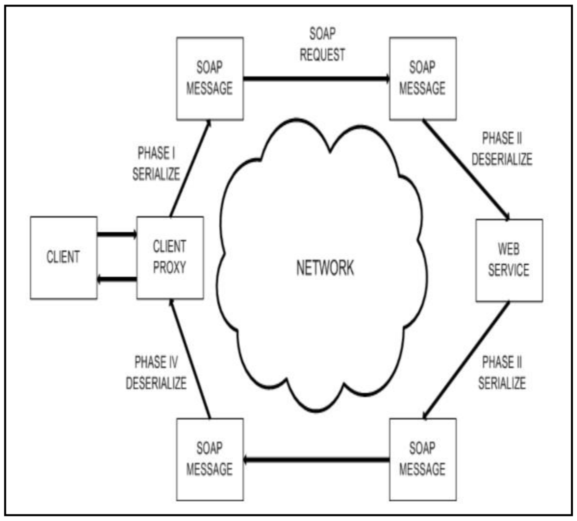


Fig. 2.11: Implementation of Compression and Decompression after serialization and before deserialization. Adopted from [22]

In as much as there is a tradeoff between compression and CPU usage, this can be resolved by use of powerful server that uses multi-processing cores [14]. Moreover Moores's Law shows the doubling of transistors in integrated circuits (IC) in computer hardware in a span of approximately two years. This show processing speeds will not be a limiting factor for compression in the near future.

Gzip compression algorithm has been adopted by web browsers [72] and web servers [73] [74] as a way of compressing data in client-server communication model. Web browsers can decompress and render Gzip files compressed by web servers. This has improved bandwidth utilization and response time of files fetched to and fro the web server.

2.3 Server side

A server is a computer that receives and processes requests from client(s). Normally a server computer is has high hardware specifications in order to process its client's requests efficiently. All the computing operations involved in server computer are said to exist in the Server Side. In this section we will discuss some of the techniques involved in the server side operations in relation to SOAP which are: server side caching technique discussed in section 2.3.1 and differential deserialization (DDS) discussed in section 2.3.2. These techniques are discussed as follows.

2.3.1 Server Side Caching

Server side caching is the storage of data on the server side. Server side caching improves response time [2] [45] [46]. As discussed in client side caching in section 2.1.1, server side caching is slightly different as data is temporarily stored in serialized objects [46].

In [45], cache is categorized in two methods: message body and template cache.

- Message body: This involves storing all SOAP message body calls. According to the client's request, the cache identifies each message body with two parameters: unique ID and Time To Live (TTL). If the requested XML message exists in cache and the TTL is valid, then the message is fetched from the cache and returned accordingly. Otherwise it's fetched in the server.
- Template cache: In this method, it's argued that in a service process, clients request the same elements but with different real time values. With this technique, the elements can form the template as the real time values are dynamically interchanged. This avoids reconstruction/destruction of the template message. Just like the message body technique functions, unique IDs are generated for message bodies and validated against when a client makes requests vis-à-vis its TTL in the template cache. This is managed by the template cache management module. Template cache structure is as shown in figure 2.12.

Results of optimized message and template cache are as shown in figure 2.13. Template caching is seen to have better performance than message body caching technique.

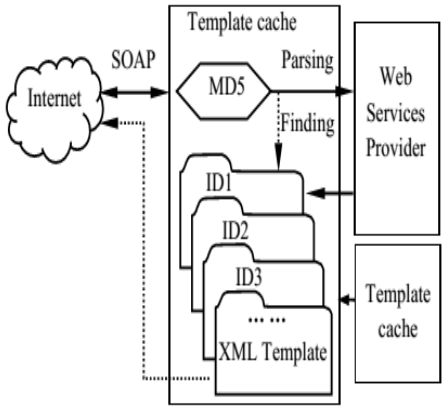


Fig. 2.12: Structure of Template Cache. Adopted from [45]

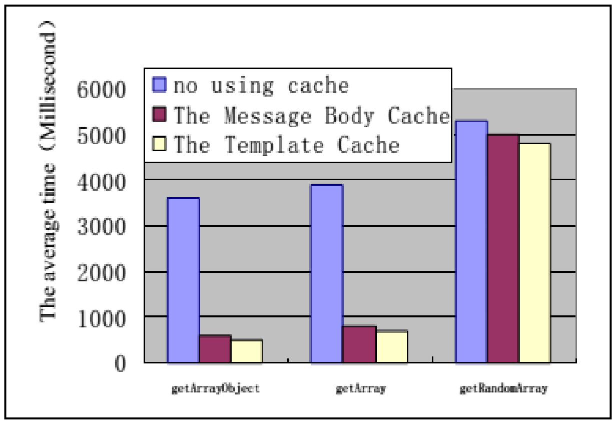
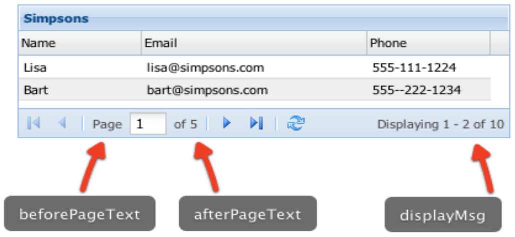


Fig. 2.13: Test results for template and message body caching. Adopted from [45]

Server side data chunking is one technique that is typically handled on the server side. As the number of records to be loaded on the browser increases, the time required to load the data increases. Data chunking comes in as a very important technique for the response from the server to return a sustainable amount of data to the client. In Data chunking, the client specifies the range of data in the request; though this is handled programmatically. The server then composes the chunk and returns it via the response method. This improves performance of loading thousands of data, by loading chunks or bits [43] [68]. Among other JavaScript libraries, Ext. JavaScript 4.1 [68] has adopted this technique as paging which is quite essential in modeling controls e.g. grid as shown in Figure 2.14.

Similarly as discussed in section 2.1.1 (client side caching), the challenge of keeping the cache up-to-date is also a major problem in server side caching [46]. Nevertheless, research in [46] [48] proposes use of database-aided caching technique in the web server. Database caching comes with many advantages including: modifications can be done easily, it has the ability to augment data with metadata, it eliminates the need to parse an entire XML structure which is computationally expensive, and an extra column can be used to store the aforementioned last modified timestamp value for each record easily. A suitable predicate value can be stored in replacement of last modified timestamp value to allow quick comparison between requests. Session management between multiple clients and proxy web server is a rich area of study that can be exploited further [46] [48].



Name	Email	Phone
Lisa	lisa@simpsons.com	555-111-1224
Bart	bart@simpsons.com	555--222-1234

Fig. 2.14: Data chunking/paging example in Ext. Js. Adopted from [68]

2.3.2 Differential deserialization

Differential Deserialization (DDS) works in the server/receiver side. DDS technique has been supported by other researchers [2] [30] [49] [50] [53]. DDS works best if similar messages are sent by different clients [2] [49]. DDS is somehow similar to Differential Serialization (DS). DDS and DS take advantage of sequence of similar messages to avoid the expensive SOAP message de-serialization/serialization process respectively. Neither of them changes the SOAP protocol, the SOAP message nor SOAP message wire format. Both implementations remain independent and interoperable with other SOAP implementations [49]. Moreover, researchers in [51] note that DDS is more promising implementation technique than DS because DS works if the same client sends a stream of similar messages whereas DDS can avoid deserialization of similar messages sent by multiple clients. The speed of the server among other factors determines performance in DDS. Some of the differences between DDS and DS are captured in table 3.

Table 3: Comparison between differential deserialization and differential serialization

Differential deserialization	**Differential serialization**
Works in the server side	Works in the client side
Targets incoming messages	Targets outgoing messages
Deserialization process involves converting of SOAP XML-messages to application object	Serialization process involves conversion of in-memory data types to SOAP XML ASCII format
Uses parse checkpointing the state of the deserializer for incoming messages	Uses Data Update Tracking (DUT) Table to track whether a program has changed data items in new messages since they were last serialized SOAP messages in outgoing message

Deserialization is an expensive process that involves conversion of SOAP XML-messages to application object. Deserialization involves a series of undertakings which include fetching an appropriate deserializer from the type mapping registry, and constructing a Java object from an

XML message. To be precise and relatively simple, the process of de-serializing an XML message into Java objects is as follows [53]:

- Open an XML document that represents the object.
- Reclusively de-serialize the object's members which are encoded as sub-elements after locating an appropriate deserializer from the type mapping system.
- Create a new instance of the Java type, initializing it with deserialized members.
- Return the new Java object.

These undertakings become more complex and expensive when the XML message becomes bigger and deeper. Deserialization can be improved by processing new regions of the XML messages and reuse of the constructed objects deserialized in the past. It even becomes more expensive when handling scientific data stored in arrays, floats, and doubles [53] [50] [69]. Objects are created before they are used and garbage collected after they are finished to be used. The use of more objects affects the performance of garbage-cycling process [53].

Researchers in [71] noted that reuse of the entire object trees works for messages that have exactly the same structure. Researchers in [53] improved on the approach presented in [71]. In [53] it's noted that the fundamental characteristic of processed SOAP-based web services messages is that, their wire format structure has lots of similarity. By exploiting this weakness redundancy can be avoided by using a deserialization mechanism that reuses matching structures/objects from previously deserialized applications objects; deserialization for new regions is only performed on regions that will not have been processed before. Researchers in [53] obtained a 288% maximum performance gain by the use of this technique. However, in large messages on the wire that have repetitive elements like GoogleSearchLargeservice and in cluster-like environments, in such cases, reusing the entire object tree is not the optimal solution because repetition number might differ for each request and requires us to consider issues such as thread safeness and scalability respectively [53].

Nonetheless, DDS primary shortcomings in SOAP message exchange are processing of XML data/content and conversion of strings to in-memory data types; checkpointing is explored further [69]. DDS works by periodically checkpointing the state of SOAP deserializer, which reads and de-serializes incoming message portions, and computing checksum of these SOAP message portions. The checksum is compared against those of the corresponding message portion in the previous message. If the checksums match, the deserializer avoids redoing de-serializing (parsing and converting SOAP message) contents in that region. Essentially, the deserializer runs in two different modes: regular and fast mode. In regular mode, the deserializer reads and processes all the SOAP tags and message content as it creates checkpoints and corresponding message checksum along the way to the end of the SOAP message, whereas, in fast mode, the deserializer considers the sequence of checksum of each disjointed portions of the message and compares them against the sequence of checksums associated with the most recent received message. The deserializer switches between regular and fast mode appropriately.

Fast mode is switched on if the parser state is the same (checksum match) as the one that has been saved in a checkpoint. Regular mode is switched on when there is a checksum mismatch. This indicates a difference in the incoming message and the corresponding previous message. The deserializer switches from fast to regular mode where it reads and converts the message portion's content. Regular mode is actually the normal parsing without DDS optimization. In terms of performance, in fast mode, in the best scenario (when all the message portions are identical, though unrealistic), the normal cost of de-serializing is replaced by the cost of computing and comparing checksums which is generally significantly faster. In regular mode, the worst case scenario (when all the message portions are not identical), the DDS enabled deserializer runs much slower than a normal deserializer because it does the same work plus the added work of calculating checksums and creating parser checkpoints [69]. Further, [69] [70] noted that creating many checkpoints can increase fast mode performance in terms of speed at the expense of checkpoint creation time, check point memory utilization, and checksum calculation and comparison time. Actually checkpointing is memory intensive.

Due to the relatively high memory requirements experienced in [69], [70] introduces a new technique for storing only the difference between successive parser state for messages, this technique is called Differential Checkpointing (DCP). DCP involves only the differences between the consecutive checkpoints as opposed to storing the entire parse states for each checkpoint. DCP optimizes DDS by improving its speed and reducing memory requirements. Despite the fact that DCP reduced memory requirements, it still required significant processing overheads. Moreover, DDS primary shortcomings in its implementation are generating, storing, and using parse checkpoints. Researchers in [69] introduced Lightweight Checkpointing (LCP), a checkpointing approach significantly reduced cost of DCP and DDS techniques.

LCP checkpoints contain very little state information (fewer bytes) created at predefined points within the structure of the message. Each lightweight checkpoint in LCP has a reference to a base checkpoint that contains state information it shares with other lightweight checkpoints. In LCP checkpoints are be created much faster than regular checkpoints, hence requires much less memory and requires less processing overheads. LCP takes only 10% of memory that DCP requires and 3% of the memory original checkpointing algorithm required. For processing time, deserialization with LCP was approximately 36% better than DCP and approximately 52% better than Full Checkpointing (FCP), on average, when approximately half of the message is not changed from the previous message. In FCP, the full parser state is stored with each checkpoint.

Moreover, [50] suggested a Serialization Enhancement Middleware (SEM) Technique that a utilization a combination of DS and DDS techniques to improve response time. SEM is an implementation that runs on the middleware to run on top of any web server. SEM acts as the primary module and takes advantage of similar SOAP requests in a web server. Similarly, SEM avoids redundant serialization stage of SOAP response for request which have completely the same parameters. SEM maintains a trie of incoming parameters for current requests thus processing and serialization of response of same requests is done only once.

3. Conclusion

WS is the most widely used techniques in realizing SOA as compared to the traditionally used CORBA, and Java RMI. SOAP and REST are the only techniques used in implementing SOAP. Despite the fact that REST is a light weight technology and consumes lesser bandwidth, SOAP has proven to be adopted by many software vendors and is more secure than REST. As opposed to SOAP, REST is being adopted mostly in mobile programming. Notwithstanding, programmers are advised to choose wisely on whether to adopt REST or SOAP while developing their applications putting into consideration the entirely on application complexity, requirements, and constrains. Many researchers evaluate SOAP majorly in terms of bandwidth utilization, response time and throughput. WS performance evaluation tools and techniques is a research area that is yet to be explored fully and benchmarked.

Many researchers disagree on whether SOAP is indeed a lightweight protocol. In this survey paper, we uncovered that SOAP is a lightweight mechanism for packaging messages; its dependence on XML is the primary performance drawback. XML documents are not only huge and verbose, but also the processing of XML content and conversion to and fro memory data types, are the major performance hindrances in high performance application. These have led to high network traffic, high latency and processing delays.

An overwhelming number of researchers have made various tremendous contributions in optimizing SOAP in high performance application. W3C sets the standards that software vendors should adhere to in implementing SOAP based WS. Software vendors include Microsoft, IBM etc. As per SOAP's characteristics, researchers have come in with various techniques in optimizing SOAP performance. This survey paper has classified these techniques thematically: client side, communication channel, and server side. The client side has covered client side caching, and differentials serialization. The communication channel has covered SOAP binding styles and compression. Lastly, server side has covered server side caching and differential de-serialization. In SOAP optimization, integrating multiple optimization techniques is a growing trend which indeed has brought better results and conclusions that has shown SOAP even performing much better than traditional Java RMI, and CORBA. Nevertheless, there exist other SOAP optimization techniques that were not covered in this survey paper including: transport protocol, client caching algorithms, compression algorithms comparisons, and SOAP parsing.

References

[1] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte and D. Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. [Cited: August 11, 2013]
[2] Seyyed, Hasan and Roya, Zareh. A Combination Approach for Improvement Web Service Performance. Proceedings of the International MultiConference of Engineers and Computer Scientists, Hong Kong, Vol I
[3] D. Linthicum. Service Oriented Architecture (SOA). http://msdn.microsoft.com/en-us/library/bb833022.aspx. [Cited: August 13, 2013]
[4] H. Qusay. Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). http://www.oracle.com/technetwork/articles/javase/soa-142870.html. [Cited: August 13, 2013]
[5] New to SOA and web services. http://www.ibm.com/developerworks/webservices/newto/index.html#ibm-pcon. [Cited: August 13, 2013]

[6] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, L. Min and T. Newling. Patterns: Service Oriented Architecture and Web Services. Red Books, 2004. Vol. I

[7] P. Bianco, R. Kotermanski and P. Merson. Evaluating a Service-Oriented Architecture. Hanscom, Carnegie Mellon University, 2007

[8] IONA Technologies. CORBA Programmer's Guide, Java. Sun Microsystems, Inc, 2005. Vol. 6.3.

[9] W. Grosso. Java RMI. O'Reilly, 2001

[10] COM: Component Object Model Technologies. http://www.microsoft.com/com/default.mspx. [Cited: August 23, 2013]

[11] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard. Web Services Architecture. http://www.w3.org/TR/ws-arch/#whatis. [Cited: August 13, 2013]

[12] G. Coulouris, J. Dollimore and T. Kindberg. Distributed Systems Concepts and Design. Essex , Pearson Education Limited, 2009, pp. 789. Vol. IV

[13] S. Mumbaikar and P. Padiya. Web Services Based On SOAP and REST Principles. International Journal of Scientific and Research Publications, 2013, Vol. III

[14] H. El-Bakry and N. Mastorakis. Studying the Efficiency of XML Web Services for Real-Time Applications. Proceedings of the 2nd WSEAS International Conference on Sensors, and Signal and Visualization, Imaging and Simulation and Material Science. Wisconsin, USA, pp. 209-219

[15] D. Al-Shammary and I. Khalil. SOAP Web Services Compression using Variable and Fixed Length Coding. Network Computing and Applications, 2010

[16] J. Snell, James, D. Tidwell and P. Kulchenko. Programming Web Services with SOAP. Canada: O'Reilly, 2001, pp. 7. Vol. I

[17] H. Hamad, M. Saad and A. Ramzi. Performance Evaluation of RESTful Web Services for Mobile Devices. International Arab Journal of e-Technology, 2010, Vol. I

[18] F. AlShahwan, K. Moessner and F. Carrez. Evaluation of Distributed SOAP and RESTful Mobile Web Services. International Journal on Advances in Networks and Services, 2010, Vol. III.

[19] H. Hamad, M. Saad and A. Ramzi. Performance Evaluation of RESTful Web Services for Mobile Devices. International Arab Journal of e-Technology, 2010, Vol. I

[20] K. Hameseder, S. Fowler and A. Peterson. Performance Analysis of Ubiquitous Web Systems for SmartPhones. IEEE International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'11), 2011

[21] A. Mullally, N. McKelvey and K. Curran. Performance Comparison of Enterprise Applications on Mobile Operating Systems. TELKOMNIKA, 2011, Vol. IX

[22] I. Ivanovski, S. Gramatikov and D. Trajanov. Enhancing Performance of Web Services in Mobile Applications by SOAP Compression. Proceeding of: Telekomunikacioni Forum Telfor, 2008, Belgrade

[23] A. Gandhi. SOAP vs. REST – The Best Web Service http://greatgandhi.wordpress.com/2010/06/16/soap-vs-rest-%E2%80%93-the-best-webservice/. [Cited: November 16, 2013]

[24] P. Cesare. REST vs. SOAP: Making the Right Architectural Decision. 1st International SOA symposium. Amsterdam, 2008

[25] The structure of a SOAP message. http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=%2Fcom.ibm.cics.ts31.doc%2Fdfhws%2Fconcepts%2Fsoap%2Fdfhws_message.htm. [Cited: August 16, 2013]

[26] M. Papazoglou. Web Services: Principles and Technology. Pearson Education Limited, 2008. Vol. I.

[27] K. Phan. Enhanced SOAP Performance for Low Bandwidth Environments, 2007

[28] Quality of Service (QoS). http://www.cisco.com/en/US/products/ps6558/products_ios_technology_home.html. [Cited: November 21, 2013]

[29] M. E. Gomez. Performance Analysis of Web Applications, 2005

[30] J. Tekli, E. Damiani, R. Chbeir and G. Gianini. Similarity-based SOAP Processing Performance and Enhancement. IEEE, 2011, Vol. 5, pp. 387 - 403

[31] B. Kinicki. Advanced Computer Networks. http://web.cs.wpi.edu/~rek/Nets2/C10/C10.html. [Cited: November 2013, 2013]

[32] H. Shariq, Z. Wang, I. K. Toure and D. Abdoulaye. Web Service Testing Tools: A Comparative Study, 2012

[33] soapUI. http://www.soapui.org/. [Cited: November 21, 2013]

[34] Apache Software Foundation. http://jmeter.apache.org/. [Cited: November 21, 2013]

[35] Storm. http://storm.codeplex.com/. [Cited: November 21, 2013]

[36] Apache Software Foundation. http://httpd.apache.org/docs/2.2/programs/ab.html. [Cited: November 21, 2013]

[37] Terelik – Fiddler. http://fiddler2.com/features. [Cited: November 21, 2013]

[38] Network Monitor Tool and Parsers. www.microsoft.com. [Cited: January 1, 2014]

[39] Wireshark. http://www.wireshark.org. [Cited: January 1, 2014]

[40] NeoLoad Cloud Testing. http://www.neotys.com/introduction/neoload-cloud-testing.html. [Cited: November 21, 2013]

[41] M. Nagy. Software Performance Profiling. ACM, 2008

[42] T. Dorsey. Tools and Techniques for .NET Code Profiling. http://msdn.microsoft.com/en-us/magazine/hh288073.aspx. [Cited: November 21, 2013]

[43] C. Mouli and C. Rajendra. Caching and SOAP Compression Techniques in Service Oriented Architecture. International Journal of Advanced Research in Computer Engineering & Technology, 2012, Vol. I

[44] K. Devaram and D. Andresen. SOAP Optimization via Client-side Caching. Citeseer, Manhattan, 2003

[45] H. Zhai-wei, Z. Hai-xia and G. Guo-hong. A Study on Web Services Performance. Third International Symposium on Electronic Commerce and Security Workshops, 2010

[46] M, Schapranow, J. Krueger, V. Borovskiy, A. Zeier and H. Plattner. Data Loading & Caching Strategies in Service-Oriented Enterprise Applications. Congress on Services, 2009

[47] D. Davis, M. Parasha. Latency Performance of SOAP Implementations. Proceedings of IEEE Cluster Computing and the GRID 2002 (CCGRID'02). Berlin, 2002

[48] J. Tatemura, P. Oliver and A. Sawires. WReX: A Scalable Middleware Architecture to Enable XML Caching for Web-Services: International Federation for Information Processing, 2005

[49] N. Abu-Ghazaleh, M. Lewis and M. Govindaraju. Differential Serialization for Optimized SOAP Performance. International Symposium on High Performance Distributed Computing (HPDC), New York, 2004, pp. 55-64

[50] B. Minaei and P. Saadat. SOAP Serialization Performance Enhancement DESIGN AND IMPLEMENTATION OF A MIDDLEWARE. International Journal of Computer Science and Information Security (IJCSIS), 2009, Vol.6, No. 1, pp. 106-110

[51] N. AbuGhazaleh and M. Lewis. Differential Deserialization for Optimized SOAP Performance. ACM/IEEE conference on Supercomputing. Binghamton 2005, pp. 1-12

[52] K. Chiu, M. Govindaraju and R. Bramley. Investigating the Limits of SOAP Performance for Scientific Computing. Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02). Edinburgh, pp. 246-254

[53] T. S. Takase and Tatsubori. M. Optimizing Web Services Performance by Differential Deserialization. IEEE International Conference on Web Services. New York, 2005, pp. 185- 192

[54] F. Bustamante, G. Eisenhauer, K. Schwan and P. Widener. Efficient Wire Formats for High Performance Computing. IEEE 2000, Georgia

[55] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl. [Cited: November 25, 2013]

[56] http://msdn.microsoft.com/en-us/library/ms996486.aspx. [Cited: November 25, 2013]

[57] Web services performance best practices. http://www14.software.ibm.com/webapp/wsbroker/redirect?version=compass&product=was-express-dist&topic=rwbs_perfbestpractices. [Cited: August 06, 2013]

[58] T. Girish, R. Mudholkar and B. Jadhav. JAX-WS Web Service for Transferring Image. International Journal on Computer Science and Engineering (IJCSE), 2013, pp. 63-69

[59] Alex, Ng, P. Greenfield and C. Shiping. A Study of the Impact of Compression and Binary Encoding on SOAP Performance, 2008, pp. 46-56

[60] Roberto, Chinnici, et al., et al. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/wsdl20/. [Cited: December 24, 2013]

[61] Masoud and Kalali. A Look at WSDL 2.0. http://soa.dzone.com/news/look-wsdl-20. [Cited: December 24, 2013]

[62] T. Wichaiwong, K. Koonsanit and C. Jaruskulchai. A Simple Approach to Optimized Text Compression's Performance. Proceedings of 4th International Conference on Next Generation Web Services Practices, 2008

[63] P. Tomasz, J. Sliwa and M. Amanowicz. Efficiency of compression techniques in SOAP. 2010, pp. 199-211

[64] Liquid Technologies. Liquid XML Compression Library. http://www.liquid-technologies.com/Liquid-Products/XMLCompression/XMLCompressionExample.aspx. [Cited: August 13, 2013]

[65] A. Paya and D. Marinescu. A Cloud Service for Adaptive Digital Music Streaming. Eighth International Conference on Signal Image Technology and Internet Based Systems, 2012

[66] A. Al-Canaan and A. Khoumsi. Multimedia Web Services Performance: Analysis and Quantification

of Binary Data Compression. Journal of Multimedia , 2011, Vol. VI

[67] H. Shirazee, H. Rashidi and H. Homayouni. The Effects of Data Compression on Performance of Service-Oriented Architecture (SOA). International Journal of Emerging Trends & Technology in Co mputer Science (IJETTCS), 2012, Vol. I

[68] Custom Grid Filters Example (local filtering). www.sencha.com. [Cited: December 27, 2013]

[69] N. Abu-Ghazaleh and M. Lewis. Lightweight Checkpointing for Faster SOAP Deserialization. International Conference on Web Services. New York, 2006

[70] N. Abu-Ghazaleh and M. Lewis. Differential Checkpointing for Reducing Memory Requirements in Optimized SOAP Deserialization. Grid ComputingWorkshop – IEEE. New York, 2005

[71] T. Takase, H. Miyashita, M. Tatsubori, and T. Suzumura. An Adaptative, Fast and Safe XML Parser Based on Byte Sequence Memorization. World Wide Web (WWW) Conference, 2005, pp. 692 – 701

[72] R. Constantin. http://www.http-compression.com/. [Cited: January 26, 2014]

[73] http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/25d2170b-09c0-45fd-8da4-898cf9a7d568.mspx?mfr=true

[74] G. Garron. How to enable gzip compression in Apache 2.x. http://www.garron.me/en/linux/enable-gzip-mod_deflate-compression-apache.html

Mutange Kennedy Senagi is a student undertaking Master of Science degree in the field of Software Engineering at Jomo Kenyatta University of Agriculture and Technology. He holds a Bachelor's degree in Information Technology from JKUAT (July 2012). He has undertaken a networking certification course, Cisco Networking Academy (CCNA); CCNA I and CCNA II (2010). He has pursued several mobile programming certification courses: Advanced Android Native Programming at iLab East Africa (October 2013) and Windows Phone Programming at m:Lab East Africa (February 2014). He has worked as an intern in Kimetrica Limited as a Software Developer (2010 - 2011) and as fully employed staff specialized in developing field survey data collecting system (January 2012 - April 2013). He is currently employed by Dedan Kimathi University of Technology as a Teaching Assistant (May 2013 to date).

Dr. George Okeyo received the B.Sc. degree in Mathematics and Computer Science and M.Sc. in Information Systems from the Jomo Kenyatta University of Agriculture and Technology (JKUAT) and the University of Nairobi, Kenya, respectively. He also received the Ph.D. degree in Computer Science from the University of Ulster, United Kingdom. He is a lecturer at the Department of Computing, JKUAT, Kenya. His current research interests include intelligent agents, smart homes, ambient assisted living, activity recognition, mobile and pervasive computing, data analytics, web services, semantic technologies and knowledge representation and reasoning.

Dr. Cheruiyot Wilson has acquired the following degrees: Bachelor of Science in Mathematics and computer science (1994), Masters of Science in Computer Application Technology (2002) and PhD in Computer Science Applications Technology (2012). He is also certified with Microsoft association in the following: Microsoft Certified Database Administrator (MCDBA) and Microsoft Certified Professional (MCP). Dr. Cheruiyot is currently a Senior Lecturer and Deputy Director (Postgraduate Programmes) at the Jomo Kenyatta University of Agriculture and Technology. Previous, he worked as an auditor with the Kenya National Audit Office (KENAO) (1994 - 2003) and the Teachers Service Commission of Kenya (1994 – 1997). Dr. Cheruiyot is an article reviewer with the following organizations: Journal of Petroleum and Gas Engineering, www.academicjournals.org/JGE, Journal of Engineering and Technology Research, www.academicjournals.org/JETR/index.htm. His best paper award was a paper submitted to Springerlink journal of Multimedia systems, titled "Query quality refinement in singular value decomposition to improve genetic algorithms for multimedia data retrieval", whose impact factor was 1.176 and indexed by SCI and EI Compendex. He has published over sixteen papers in different refereed journals. His current research interests are: Multimedia Data Retrieval, Internet of Things, Evolutionary Computation for Optimization, Digital Image Processing and ICT for Development.

Sati Arthur holds a Bachelors of Science degree in Information Technology from Jomo Kenyatta University of Agriculture, Kenya (November 2012). He has also completed CCNA I-IV (2010-2012). He works as an IT consultant with Olivine Technology Ltd (September 2010 to date) and Senior Technologist at Dedan Kimathi University of Technology (March 2014 to date).

Kalunda Jades is a Bachelor of Science in Information Technology first class honors holder from Jomo Kenyatta University of Agriculture and Technology (June 2012) and perusing a Master's degree in Software Engineering in the same university. He is A+ and N+ certified (2010). He worked as an Intern at Safaricom Limited (2012). He is an ICT and research consultant in Information Systems working with Multiple Tech solutions (2010 to date), Project Strategy Risk Management (PSRM) consultants (2010) and Felim Networks (2012 to date). He is currently an academic tutor at the Department of Information Technology at the Dedan Kimathi University of Technology (May 2013 to date).