SOAP PERFORMANCE ENHANCEMENT FOR HIGH VOLUME MESSAGING

ALI BABA DAUDA

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2018

University of Malaya

SOAP PERFORMANCE ENHANCEMENT FOR HIGH VOLUME MESSAGING

ALI BABA DAUDA

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SOFTWARE ENGINEERING

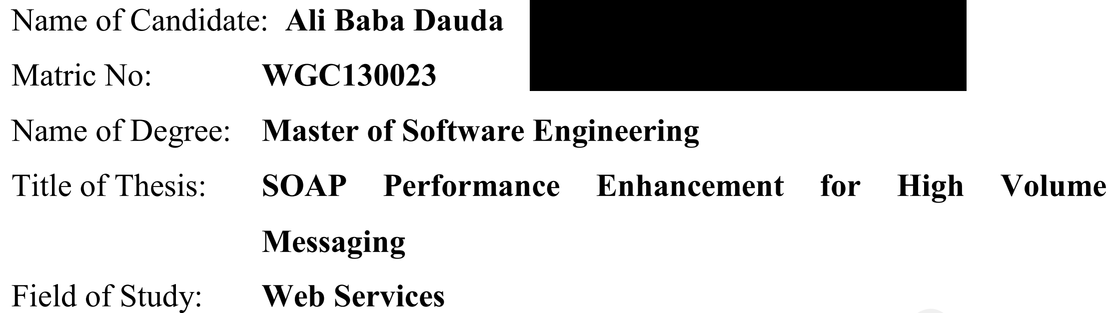
FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2018

University of Malaya

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION



Name of Candidate: **Ali Baba Dauda**

Matric No: **WGC130023**

Name of Degree: **Master of Software Engineering**

Title of Thesis: **SOAP Performance Enhancement for High Volume Messaging**

Field of Study: **Web Services**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every right in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work, I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature 12.11.2018

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation

University of Malaya

ABSTRACT

The emergence of high-volume data exchange like business-to-business and computational sciences that are mission critical and always persist over time have exacted distributed systems and applications to be fast. SOAP is one of the best protocols using XML to exchange message but the XML is too verbose and slows the communication process. To this end, message exchange accumulates overhead and high response time resulting to slow communication and message lost during the transmission. Therefore, reducing the response time and overhead will enhance the communication process. To achieve this aim, LZ77 compression algorithm is modified to encode more symbols. The algorithm is then integrated into two Web services with HTTP and JMS bindings. The HTTP Web service as the benchmark and the JMS as the prototype Web service. For both Web services, the server holds provider, compressor and controller classes and the client contain consumer and decompressor classes. The client invokes the server to establish WSDL contract and communicate via the relevant protocol. Two messages formats, normal and compressed (modified algorithm) with the size ranging 1MB - 22 MB were generated and executed 50 times in both web services. The performance effects of the message formats for the Web services were recorded. The metrics of the Web services used are the payload overhead, server response time, client response time and compression/decompression overhead. The payload overhead, server response time and compression overhead were analyzed at the server side. While client response time and decompression overhead were analyzed at the client side. Average values of the metrics were calculated and the transaction response time is obtained as the sum of response times and the overheads at both endpoints. The metrics were plotted against the message sizes and the effects were analyzed. The findings demonstrated that the compressed JMS binding on SOAP messages recorded low response time and low overhead compared to the compressed HTTP binding. In the

compressed HTTP binding, the internal process at the client side regularly claims memory while creating available space for incoming messages resulted in producing of spikes leading to high overhead. Out of the 50 executions for 12 transactions, compressed HTTP binding delivery failed 6 times, and compressed JMS binding failed 2 times. While for the normal HTTP binding delivery failed 5 times, and normal JMS binding failed 2 times. The overall findings observed that with the modified LZ77 algorithm, SOAP over JMS has proven to be better than the SOAP over HTTP. The SOAP (with the modified compression algorithm) over JMS is a good technique for exchanging high volume messages when low response time and guarantee of delivery are needed in the communication.

University of Malaya

ABSTRAK

Kemunculan pertukaran data jumlah tinggi seperti perniagaan ke perniagaan dan sains pengkomputeran yang misi kritikal dan sentiasa berterusan dari masa ke masa telah mendesak sistem dan aplikasi teragih supaya sentiasa bertindak dengan pantas. SOAP adalah salah satu protokol terbaik yang menggunakan XML untuk pertukaran mesej tetapi XML terlalu meleret dan melambatkan proses komunikasi. Akibatnya, overhead pertukaran mesej berkumpul dan masa tindak balas yang tinggi melambatkan komunikasi dan mesej hilang semasa penghantaran. Oleh itu, mengurangkan masa tindak balas dan overhead akan memperbaiki proses komunikasi. Untuk mencapai matlamat ini, algoritma pemampatan LZ77 diubahsuai untuk mengekodkan lebih banyak simbol. Algoritma ini kemudian diintegrasikan ke dalam dua perkhidmatan Web dengan pengikatan HTTP dan JMS masing-masing. Perkhidmatan Web HTTP digunakan sebagai penanda aras dan JMS sebagai perkhidmatan Web prototaip. Untuk kedua-dua perkhidmatan Web, pelayan memegang kelas pembekal, pemampat dan pengawal manakala pelanggan mengandungi kelas pengguna dan penyahmampatan. Pelanggan memanggil pelayan untuk menubuhkan kontrak WSDL dan berkomunikasi melalui protokol yang berkaitan. Dua format mesej, biasa dan termampat dengan (algoritma diubah suai) bersaiz antara 1MB - 22 MB dihasilkan dan dilaksanakan 50 kali dalam kedua-dua perkhidmatan Web. Kesan format mesej terhadap prestasi perkhidmatan Web direkodkan. Metrik perkhidmatan Web yang digunakan adalah overhead payload, masa tindak balas pelayan, masa tindak balas pelanggan dan overhead mampatan / penyahmampatan. Overhead payload, masa tindak balas pelayan dan overhead mampatan dianalisis di sisi pelayan, manakala masa tindak balas pelanggan dan overhead penyahmampatan dianalisis di sisi pelanggan. Nilai purata metrik dikira dan masa tindak balas transaksi diperolehi sebagai jumlah masa tindak balas dan overhed pada kedua-dua titik hujung. Metrik telah diplot terhadap saiz mesej

University of Malaya

dan kesannya dianalisis. Hasil penemuan menunjukkan bahawa pengikat JMS yang termampat pada mesej SOAP mencatatkan masa tindak balas yang rendah dan overhed rendah berbanding dengan pengikatan HTTP termampat. Dalam pengikatan HTTP yang termampat, proses dalaman di sisi pelanggan selalu menuntut ingatan semasa menyediakan ruang untuk mesej yang masuk mengakibatkan terhasilnya pancang yang membawa kepada overhead yang tinggi. Daripada 50 pelaksanaan dengan 12 transaksi, penghantaran ikatan HTTP termampat gagal 6 kali, dan penghantaran ikatan JMS yang termampat gagal 2 kali. Sementara itu, penghantaran ikatan HTTP biasa gagal 5 kali, dan ikatan JMS biasa gagal 2 kali. Penemuan keseluruhan mendapati bahawa dengan algoritma LZ77 yang diubahsuai, SOAP dengan JMS telah terbukti lebih baik daripada SOAP dengan HTTP. SOAP (dengan algoritma LZ77 yang diubah suai) dengan JMS adalah teknik yang baik untuk pertukaran mesej jumlah tinggi apabila masa tindak balas yang rendah dan jaminan penghantaran diperlukan dalam komunikasi.

University of Malaya

ACKNOWLEDGEMENTS

First and foremost, I thank Almighty Allah for the respite to breath to this moment. Alhamdulillah!

A worthy of thanks to my able Supervisor Associate Professor Dr. Chiew Thiam Kian. For which, without his ardent interest, dedication and unceasing and precise comments and suggestions, this dissertation might not have produced the achieved goals. I must declare my appreciation.

I acknowledge the help of the faculty staff, especially Dr. Chiam Yin Kia, Professor Dr. Lee Sei Peck and Dr. Mumtaz Begum Mustapha for their guidance and assistance in the course of this research.

My brothers and sisters, thank you for your immense effort for bringing to this level. You have contributed a lot on this study and throughout my life by providing with all necessities despite I am parent-less. No! With you I understand I have them.

I appreciate the University of Maiduguri for sending me to study and upgrade my knowledge in both theory and practice.

Finally, I thank my friends for cheering me through their well-wishing, prayer and support at the course of this research.

University of Malaya

TABLE OF CONTENTS

ABSTRACT iii

ABSTRAK v

ACKNOWLEDGEMENTS vii

TABLE OF CONTENTS viii

LIST OF FIGURES xii

LIST OF TABLES xiv

LIST OF ABBREVIATIONS xv

LIST OF APPENDICES xvi

CHAPTER 1: INTRODUCTION 1

1.1 Introduction 1

1.2 Background of Study 3

1.3 Problem Statement 5

1.4 Research Questions 6

1.5 Research Objectives 6

1.6 Research Methodology 7

1.7 Significance of Study 7

1.8 Thesis Outline 8

CHAPTER 2: LITERATURE REVIEW 9

2.1 Introduction 9

2.2 Web Services 9

2.3 Service-oriented Architecture 10

2.4 Web Service Layers 12

2.4.1 SOAP (Simple Object Access Protocol) 12

University of Malaya

2.4.2 WSDL (Web Services Description Language) 12
2.4.3 UDDI (Universal Description, Discovery, and Integration) 13
2.4.4 HTTP (Hypertext Transfer Protocol) 13
2.5 Overview of SOAP 13
2.5.1 SOAP Message Binding 15
2.5.2 SOAP Message Binding Style 15
2.6 SOAP Performance Approaches 16
2.6.1 client-side Approach 17
2.6.2 Differential Serialization 19
2.6.3 Server-side Approach 20
2.6.4 Server-side Caching 20
2.6.5 Differential Deserialization 21
2.6.6 Message Encoding/Compression Approach 23
2.7 Message Streaming 26
2.8 Limitations of the Approaches in the Literature 28
2.9 Data Compression 30
2.9.1 Types of Data Compression 30
2.9.2 Text Compression 31
2.9.3 Statistical Text Compression 31
2.9.4 Dictionary Text Compression 32
2.10 Summary 36

CHAPTER 3: RESEARCH METHODOLOGY 37
3.1 Introduction 37
3.2 Research Conception 37
3.3 Review of Related Literature 37
3.4 Identification of Research Gap 38

University of Malaya

3.5 System Requirement Analysis and Design 39

3.6 Experiment Setup 39

3.6.1 Hardware Setup 40

3.6.2 Software Setup 40

3.6.2.1 Benchmark System Setup 41

3.6.2.2 Prototype System Setup 42

3.7 Data Compression Algorithm Modification 44

3.7.1 The LZ77 Compression Algorithm 44

3.7.2 Modified LZ77 Compression Algorithm 49

3.8 System Implementation and Execution 53

3.9 Summary 54

CHAPTER 4: SYSTEM REQUIREMENT ANALYSIS, DESIGN AND IMPLEMENTATION 56

4.1 System Analysis 56

4.1.1 Use Case 56

4.1.2 Activity Diagram 58

4.1.3 Sequence Diagram 58

4.2 System Design 60

4.2.1 Class Diagram 60

4.2.2 System Components 62

4.3 System Implementation 63

4.3.1 Web Services Implementation 63

4.3.2 System Execution and Evaluation 65

4.3.2.1 The Web Services calculation for the response time and the overhead 67

4.4 Summary 70

University of Malaya

CHAPTER 5: EXPERIMENTAL RESULTS AND DISCUSSION 71
5.1 Introduction.. 71
5.2 SOAP over HTTP Protocol 71
5.2.1 Normal Payload Response Time 71
5.2.2 Compressed Payload Response Time.. 74
5.3 SOAP over JMS Protocol 77
5.3.1 Normal Payload Response Time 77
5.3.2 Compressed Payload Response Time.. 81
5.4 Comparison between SOAP over HTTP and SOAP over JMS............................ 84
5.4.1 Normal Payload for SOAP over HTTP vs SOAP over JMS.................... 84
5.4.2 Compressed Payload for SOAP over HTTP vs SOAP over JMS 88
5.5 Messaging Communication Delivery Analysis 90
5.6 Summary.. 94

CHAPTER 6: CONCLUSION... 95
6.1 Introduction.. 95
6.2 Research Aims and Objectives 95
6.2.1 Research Objective 1 95
6.2.2 Research Objective 2 95
6.2.3 Research Objective 3 97
6.3 Contributions 98
6.4 Limitations... 99
6.5 Future Work.. 99
References 101
Appendices 109

University of Malaya

LIST OF FIGURES

Figure 1.1: SOAP request/response web services 2

Figure 1.2: XML Web services 3

Figure 1.3: Research questions and objectives mapping 7

Figure 2.1: Service-oriented Architecture 11

Figure 2.2: Web services layers 12

Figure 2.3: SOAP messaging structure 14

Figure 3.1: Literature review process for the SOAP performance enhancement for large volume messaging 39

Figure 3.2: Web services request/response. Services are provided by the server based on corresponding client request 41

Figure 3.4: The flow of the research methodology process for the SOAP performance enhancement for large volume messaging 54

Figure 4.1: Use case diagram for the SOAP performance Web services enhancement showing the detailed use cases and the actors of the Web services 57

Figure 4.2: Activity diagram for the SOAP performance Web services enhancement showing the dynamic flow of activities of the Web services. 58

Figure 4.3: Sequence diagram for the SOAP performance Web services enhancement showing the objects involved in the request-response implementation 59

Figure 4.4: Class diagram for the SOAP performance Web services enhancement showing the classes involved in the HTTP web services implementation...................... 61

Figure 4.5: Class diagram for the SOAP performance Web services enhancement showing the classes involved in the JMS web services implementation 61

Figure 4.6: Component diagram for the SOAP performance Web services enhancement showing the major components for achieving the functionalities of the implementation of HTTP web services 62

Figure 4.7: Component diagram for the SOAP performance Web services enhancement showing the major components for achieving the functionalities of the implementation of JMS web services 62

Figure 4.8: SOAP messaging architecture for large volume with modified LZ77 compression algorithm 64

University of Malaya

Figure 4.9: Flowchart for the implementation of SOAP/HTTP and SOAP/JMS 65

Figure 4.10: Workflow for the Web services showing how the endpoints interact and how the metrics for the services were captured for the analysis 66

Figure 5.1: SOAP over HTTP response time for normal payload transaction 73

Figure 5.2: SOAP over HTTP server, client, compression, decompression and payload overhead for compressed payload transaction response times....................................... 75

Figure 5.3: SOAP over JMS response time for normal payload transaction comprising payload generation overhead time, server, client and overall transaction response times 79

Figure 5.4: SOAP over JMS response time for compressed payload transaction comprising payload generation overhead time, server, client, compressed, decompress and overvall transaction response times... 82

Figure 5.5: SOAP over HTTP vs SOAP over JMS response time for normal payload transactions response times 86

Figure 5.6: Compressed payload transactions response times for 89

Figure 5.7: Normal payload successful delivery for SOAP over HTTP and SOAP over JMS 91

University of Malaya

LIST OF TABLES

Table 2.1: Some worthy works on SOAP performance improvement approaches.......299

Table 3.1: Hardware requirements..40

Table 3.2: Software system specifications used in the study..41

Table 3.3: Benchmark client-server SOAP web services with HTTP protocol.............42

Table 3.4: Prototype client-server SOAP Web services with JMS protocol..................44

Table 3.5: LZ77 Data compression technique showing the search buffer of size 8, look-ahead buffer of size 9 and window size of 17. 46

Table 3.6: LZ77 process of symbol encoding for the input 'aataaaattatatat'..................47

Table 3.7: LZ77 process of symbol decoding for the encoded input 'aataaaattatatat' ...48

Table 3.8: Modified LZ77 process of symbol encoding for the input 'aataaaattatatat' ..50

Table 3.9: Modified LZ77 process of symbol decoding for the encoded input 'aataaaattatatat' 51

Table 3.10: Pseudocode of modified LZ77 compression algorithm..............................53

Table 4.1: Performance metrics calculations 67

Table 5.1: Response time for normal payload (SOAP over HTTP) 72

Table 5.2: Response time for compressed payload (SOAP over HTTP)........................75

Table 5.3: Response time for normal payload (SOAP over JMS) 78

Table 5.4: Response time for compressed payload (SOAP over JMS)...........................81

Table 5.5: Normal payload transaction response times for SOAP over HTTP vs SOAP over JMS. 85

Table 5.6: Compressed payload transaction response times for SOAP over HTTP and SOAP over JMS 88

Table 5.7: Normal and compressed payloads success rate for SOAP/HTTP and SOAP/JMS 91

University of Malaya

LIST OF ABBREVIATIONS

API:	Application Programming Interface
BXSA:	Binary XML for Scientific Applications
CORBA	Common Object Request Broker Architecture
DCOM:	Distributed Component Object Model
HTTP:	Hypertext Transfer Protocol
JMS:	Java Messaging Services
JVM:	Java Virtual Machine
LAB:	Look-ahead Buffer
LZ77:	Lempel and Ziv 1977
LZ78:	Lempel and Ziv 1978
LZW:	Lempel–Ziv–Welch
MEP:	Message Exchange Protocol
RMI:	Remote Method Invocation
RPC:	Remote Procedure Call
SOA:	Software Oriented Architecture
SOAP:	Simple Object Access Protocol
SB:	Search Buffer
UDDI:	Universal Description, Discovery, and Integration
WSDL:	Web Service Description Language
WWW:	World Wide Web
XML:	Extensible Markup Language

University of Malaya

LIST OF APPENDICES

Appendix A: Java Implementation files, endpoints and addresses ………..………...109

Appendix B: Modified LZ77 compression algorithm codes 110

Appendix C: Sample raw data sent by the server 114

Appendix D: Sample raw data received by the client 115

Appendix E-1: Raw data for normal payload for SOAP over HTTP server execution 116

Appendix E-2: Raw data for normal payload for SOAP over HTTP received by the client 117

Appendix F1: Raw data for compressed payload for SOAP over HTTP server execution 118

Appendix F2: Raw data for compressed payload for SOAP over HTTP received by client 119

University of Malaya

CHAPTER 1: INTRODUCTION

1.1 Introduction

Today we depend on the Internet for varieties of information and services. The Internet has shaped human endeavor and simplified many challenges that seemed impossible. Individuals and organizations have leveraged the Internet to obtain services in order to maximize output, evade time waste or to reduce cost. Overwhelmingly, over the time, the Internet not only shares information but also shares computer resources, applications and provides services across communicating entities.

Services are provided by application services referred to as Web services. This is an open standard summation of protocols for integrating applications and extending data within the applications by leveraging the speed and reliability of the World Wide Web (WWW) (Nitin, Paul, Davies, & David, 2016). This provides an avenue for machines or applications on the web to effectively communicate with each other. The applications process and communicate complex routine messages easily over the Internet targeting to achieve reduced costs and increased delivery time within the constraint of fewer resources.

Yu and Chen (2003) defined Web services as a collection of protocols comprised of Simple Objects Access Protocol (SOAP), Web Service Description Language (WSDL), Universal Description, Discovery and Integration (UDDI) and the Extensive Markup Language (XML). Messages are formatted and tagged by the XML and use the SOAP as a protocol to transport the XML message over the Hypertext Transport Protocol (HTTP). The WSDL, then describes the SOAP conveyed XML as web services and how it will be contained and transported to be used by applications. The process of SOAP transport by wrapping and sending the XML messages over the HTTP has resulted in high processing time and consumes network bandwidth. This always incurs

University of Malaya

overhead at both application and network resources. The overhead cost has developed a need to enhance the SOAP to perform well by maximizing the delivery speed (Tekli, Damiani, Chbeir, & Gianini, 2012). Figure 1.1 shows the request/response cycle in Web services.

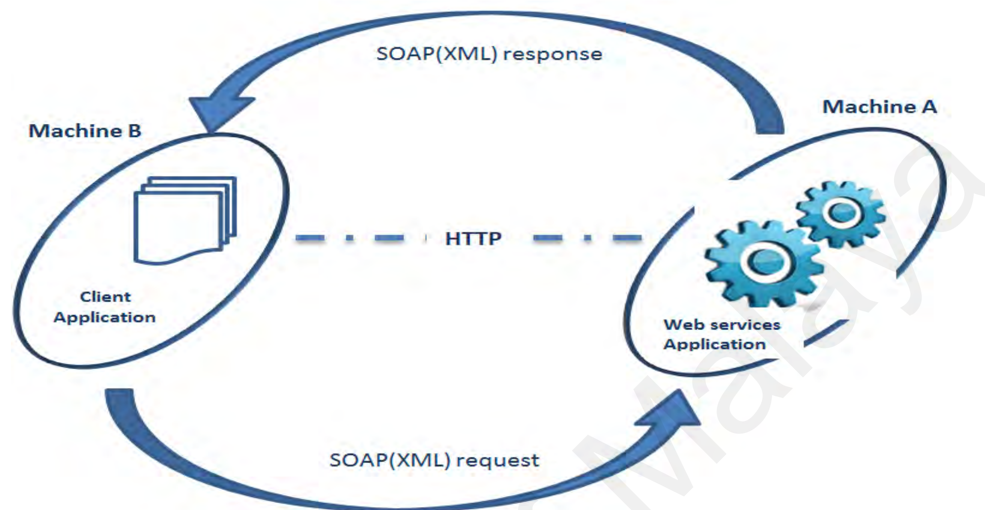


Figure 1.1: SOAP request/response Web services (W3Schools, 2014b)

Various SOAP performance improvement techniques have been developed aiming to reduce the XML size or the message size to be delivered (Mohamed and Zeki, 2017). In spite of its hindered performance, SOAP has provided a promising platform by leveraging the Internet to allow web services to communicate via its standardized protocol using the loosely coupled implementation. Thus, it has become a focus for software performance engineers to improve its performance for better and effective output (Chow, Meisner, Flinn, Peek, & Wenisch, 2014).

In order to further improve the SOAP Web services performance, this research intends to improve the performance of the SOAP by enhancing large-volume messages performance. This will help in high-volume message delivery with minimal overhead across web services.

University of Malaya

1.2 Background of the study

The Web is witnessing an explosion of information rapidly and information processing is taking a wider dimension. Information is processed on different machines and stored on different machines for instance, in cloud computing and data centers. Different middleware technologies have been used to send and receive information by the communicating applications (Iqbal, Shah, James, & Cichowicz, 2013).

Figure 1.2: XML Web services (Mohamed and Wijesekera, 2012)

Mutange, Okeyo, Cheruiyot, Sati, & Kalunda (2014) accorded that among the middleware technologies, Web Services are becoming the order of transactions in the field of distributed applications, with growing number of domains involving in composition of web services. This provides machine-to-machine interoperability communication across the web using XML-based standards to create and consume services by the provider and the consumer applications respectively (Juric, Rozman, Brumen, Colnaric, & Hericko, 2006). Figure 1.2 shows the XML web services provider as well the consumer and how the web services interact to produce the communication by sending XML/HTML based contents.

University of Malaya

SOAP as a major protocol in Web services, has been the vital building block in distributed application development where its functionality is deployed as a service via the Internet or intranet. SOAP XML-based implementation over HTTP is widely accepted to be used in data transfer by giving access to services on the web to communicate with each other with no constraint to any protocol, platform, operating system or programming language (Perez-Castillo, Guzman, Caballero, & Piattini, 2013; Zimmermann, Tomlinson, & Peuser, 2012).

Data are sent and received in formatted form as XML document in every transaction across the communicating Web services, which are embedded and transported as part of wordy XML (Pawar and Chiplunkar, 2017). As a result of its verbosity, XML has been a major bottleneck in the SOAP performance when sending or receiving data and is considered slower than its competing technologies like CORBA and DCOM (Isaac and Devi, 2014; Tekli et al., 2012).

Many studies have been conducted on Web services at both server and client sides to improve performance in order to minimize delivery time and enhance output quality (Kalyani, 2012). As reviewed by Abbas, Bakar and Ahmad (2014) and Pavan, Sanjay, Karthikeyan and Zornitza, (2013), SOAP performance improvement has been conducted on variety of data to justify its significance. Approaches such as data caching, serialization and deserialization as well as XML data compression (Massimiliano, Filippo, Xiang, & Ingolf, 2013) have been implemented to cater for XML's shortcoming. Quality of Service performance metrics; response time, throughput and payload on large scale enterprise are still not fully ascertained by researchers (Bosin, Dessì, & Pes, 2011).

University of Malaya

1.3 Problem Statement

Large number of data are increasingly deployed and exchanged over the Internet. Applications and services over the web interact and communicate to share and transfer messages (Mutange et al., 2014). Enterprises applications in data centers and the emergence of Software as a Service (SaaS) in cloud computing has led to the continuous transfer of high volume of data across applications on the web. Consequently, this exchange demands speed and guarantee of delivery to the requesting application/services (Val, Garcia-Valls, & Estevez-Ayres, 2009).

Web services provides an open standard protocol for interoperability among nodes (Kumari and Rath, 2015). SOAP as a major protocol in the Web services, provides inter-communication among applications using XML-based messaging over the HTTP (Abbas et al., 2014). Despites its popularity as a web protocol, the XML in SOAP has generated a lot of concern over its verbosity that affects the SOAP performance (Mutange et al., 2014; Pavan et al., 2013). As such, XML is marked as the bottleneck in effective SOAP Web services' delivery (Isaac & Devi, 2014; Pirnau, 2010). The XML verbosity generates communication overhead and high response time in communication.

Fu, Belqasmi and Glitho (2010) reported that HTTP is the basic SOAP Web services transportation protocol but is traditionally not a suitable candidate for transmission of high volume data in distributed applications. Nonetheless, SOAP over HTTP does not guarantee delivery of message as such it is very difficult to attest the transfer of critical and sensitive messages.

Hence, there is a need to address the issue due to the impending huge amount of continuous growing data expected on the web. This research will explore the performance of SOAP Web services on large payloads to improve the performance.

University of Malaya

1.4 Research Questions

Three research questions were designed in order to meet the objectives of this research accordingly.

RQ1: What is the performance effect of high payload on SOAP Web services' response time?

RQ2: What is the response time to successfully deliver a high payload and the overhead per transaction?

RQ3: Does the implemented approach improve the Web services response time and overhead for the high payload messages?

1.5 Research Objectives

This research deliberates on enhancement of SOAP performance especially for high volume messaging. To achieve the goal, the following objectives are set to accomplish:

RO1: To implement an approach for high payload exchange in SOAP Web services.

RO2: To determine the high payload response time and overheads in the implemented approach.

RO3: To evaluate the performance of the implemented approach and prototype in terms of response time and overhead.

Figure 1.3 shows the mapping of each research question to the research objective.

University of Malaya

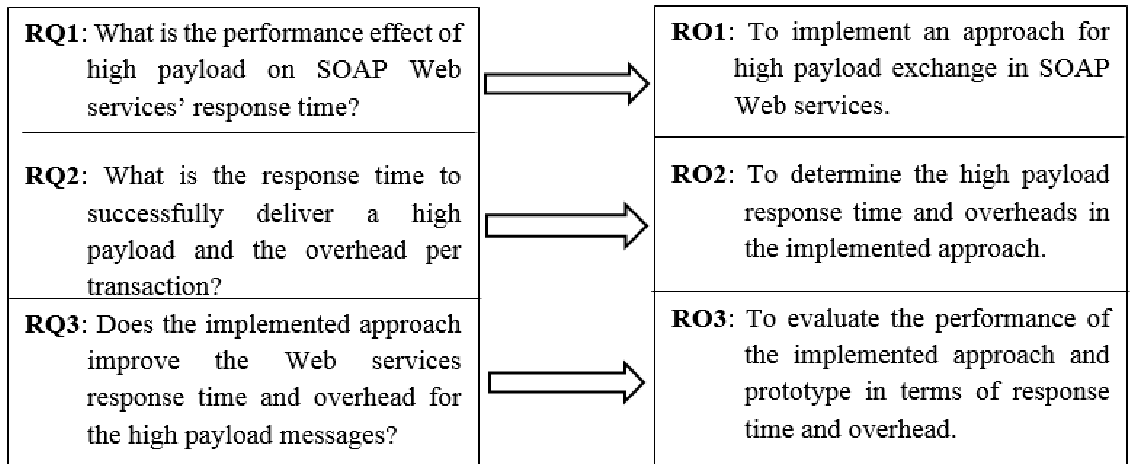


Figure 1.3: Research questions and objectives mapping

1.6 Research Methodology

The aim of this research is to implement a compression approach to send/receive high payload messages.

Web Services and Java based JMS is chosen as the framework. Two web services were created; SOAP over HTTP as a benchmark web service and SOAP over JMS as the experimental web service. LZ77 text compression algorithm is modified to accommodate more encoding symbols and then integrate into the two web services. The same message request is used for both SOAP over HTTP and SOAP over JMS web services to study the effect of the modified algorithm. The message is generated and increased and the response time, computing overhead and message size are monitored and measured for each transaction. The metrics were collected by taking the average of 50 trials in each transaction. Appendices E1 – F2 provide the samples of the metrics and how each metric was calculated.

1.7 Significance of Study

The approach proposed in this research will add to the approaches already discussed in the literature. Hence, add to the efforts made by researchers in improving the performance of SOAP. The proposed approach is expected to improve delivery time by

reducing the transaction response time and the computing overhead at both the service producer and the service consumer.

The outcome of this study will provide developers, researchers as well IT analyst with an insight of high-volume messaging across Web services standards.

1.8 Thesis Outline

The thesis is presented in six (6) chapters. Chapter 1 provides an overview of the SOAP performance enhancement for high-volume message transaction. This comprises of the introduction, study background, the statement of the study problem, significance of the study, research questions, research objective, overview of research methodology, significance of study and the thesis outline.

Chapter 2 of the thesis constitutes the literature review and basic theory of the concept of the Web services. Approaches for improving SOAP; message caching, differential serialization; differential deserialization and data compression are also presented in this chapter.

Chapter 3 presents the research conception, the experimental setup, the modified LZ77 algorithm, system implementation and execution. Chapter 4 provides the system requirement analysis, system design, and system implementation. Chapter 5 offers the results of the experiments and the discussion of the results. Chapter 6 concludes the research by highlighting the research contributions, research limitations and recommendations for future work.

University of Malaya

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

The aim of this chapter is to provide a thorough analysis of related references in the literatures, to provide a clear understanding of the different approaches and the methods used. Since SOAP is a major communication facilitator in web services architecture, improving its performance has been a focal point of most researchers in web service message deployment. SOAP performance improvement has been experimented on variety of data to justify its significance in web services. Server-side and client-side performance techniques are deployed with attempts to improve the performance (Massimiliano et al., 2013). Message compression/decompression is also applied in the sender/receiver exchange to improve the SOAP performance (Al-Shammary and Khalil, 2010).

The approaches, techniques and styles utilized by key contributing researchers in the area are studied. Performance metrics; response time, throughput and workload on large payload messaging are still not fully explored by researchers. Although, these literatures cover different approaches and techniques to improve the SOAP performance, this study is aimed to build on this body of knowledge by enhancing the SOAP performance on large volume messages. This chapter is categorized in the following order: web services, service-oriented architecture, SOAP implementation, SOAP performance improvement approaches, message streaming process, limitation of existing approaches, data compression, and types of data compression, data compression techniques and finally the summary.

2.2 Web Services

Web services are collection of protocols for enterprise applications that include programming, business logics and data that are communicated using HTTP protocol

University of Malaya

over the Internet (Maya and Ugrasen, 2012). As a collection it provides a set of principles for interacting between application components across different frameworks, operating systems and platforms (Ahmad, Sarkar, & Debnath, 2014).

Web Service transforms web applications to the advanced level of its functionality (W3Schools, 2014a). It provides a standard way to realize system integration effectively using network (Vandikas, Quinet, Levenshteyn, & Niemöller, 2011). It provides machine-to-machine interoperability communication across the web using an XML-based standard to create and consume the services by the provider and the consumer (Eugène and Fréjus, 2012; Vandikas et al., 2011).

Web Service is described as the boundary between the applications and the real world serving as an interface with defined operations to implement the business logic of an application delivered through a standard Internet protocols (Wagner, Roller, Kopp, Unger, & Leymann, 2013). It uses XML-based protocol: the SOAP, WSDL and UUDI to connect existing software applications (Hertis and Juric, 2014). Despite prevalent advocacy of web services, there are challenges that require proficient considerations. For instance, latency time during invocation may cause unpredictable result or even eventually lead to loss of message (Aihkisalo and Paaso, 2012).

2.3 Service-oriented Architecture

A standard for defining the web services is the service-oriented architecture (SOA) which is a technique that incorporates the interconnection between loosely coupled and flexible software components that are meant to operate independently (Katsikogiannis, Kallergis, Garofalaki, Mitropoulos, & Douligeris, 2018). The SOA supports the implementation of component reuse, scalability and flexibility. According to Gerić and Vrček (2009), generally, SOA is a set of standards for developing a unified and interoperable services and a component model for defining web services architecture. It

University of Malaya

has attracted popularity due to its architectural style in message-centric applications in distributed systems. SOA is agile and flexible and has competitive advantage in various context (Bu, 2011). It incorporates the relations among service provider, service requester and service registry with their respective actions of publish, bind and find Abhaya, Tari, & Bertok, 2012; Koulouzis, Cushing, Karasavvas, Belloum, & Bubak, 2012).

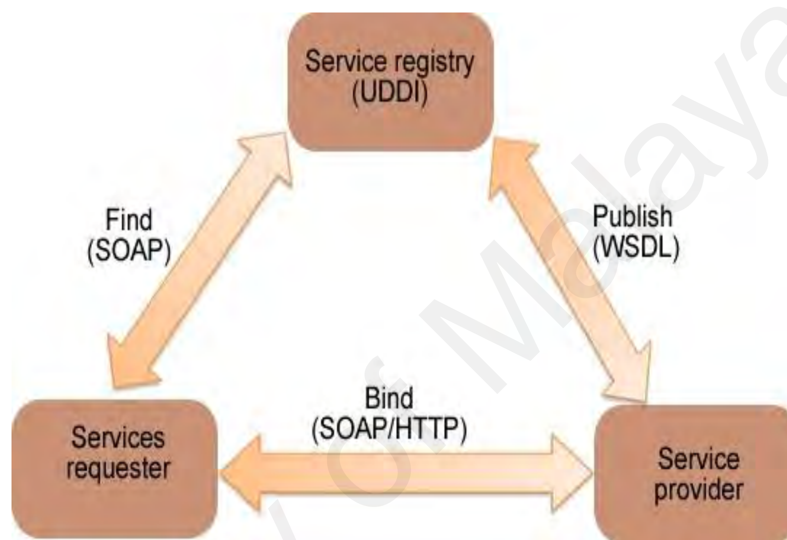


Figure 2.1: Service-oriented Architecture (Newcomer and Lomow, 2005)

Service provider creates the web services and to publish the services using the WSDL descriptive content at the registry, and to be found and consumed by the service requester. Figure 2.1 shows how the SOA is utilized by the service provider and service consumer in the web services transaction.

A service requester uses the UDDI registry to find a published service description by the registry and to bind or invoke the service provider.

Service registry is responsible for registering and arranging published service and allows both the service requester and provider to interact with each other and establish a transaction.

University of Malaya

2.4 Web service layers

Web services layers is a collection of layers on protocol standards that support XML-based communication across network. Banded together, these layers constitute a web service for publishing, describing, finding and transferring data effectively among services.

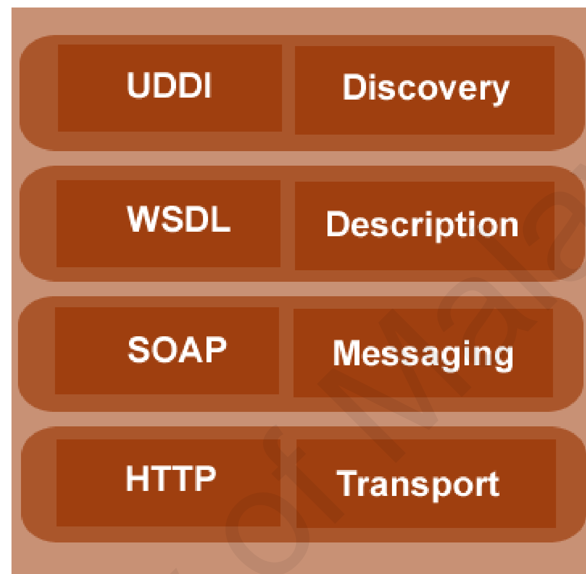


Figure 2.2: Web services layers (Newcomer and Lomow, 2005)

2.4.1 SOAP (Simple Object Access Protocol)

SOAP is an XML based protocol for accessing Web services (Maya and Ugrasen, 2012). It uses a set of XML information and HTTP over the network to exchange information across (Katsikogiannis, et al., 2018). It serves as the standard format for exchanging data in form of messaging over the HTTP. SOAP is the third layer of web services as shown in Figure 2.2.

2.4.2 WSDL (Web Services Description Language)

The WSDL describes the overall information that is effective for transmission in a request/response for a particular Web services (Hertis and Juric, 2014). As shown in Figure 2.2, WSDL is the second layer of web services serving basically as a standard

University of Malaya

XML file that contains well-defined information on all associated request/response transactions in SOAP.

2.4.3 UDDI (Universal Description, Discovery, and Integration)

The UDDI serves as a register where firms worldwide can create their lists on the Internet to publish and discover each other (Hertis and Juric, 2014). It provides a central control of Web services components and streamlines services with each other to publish and consume data effectively. UDDI is the first layer of web services as shown in Figure 2.2.

2.4.4 HTTP (Hypertext Transfer Protocol)

HTTP is a strategy for encoding and transmitting data between a service provider and service requester over the Internet (Iqbal et al., 2013; Mohamed and Wijesekera, 2012). HTTP is fourth layer in the web services layer as seen in Figure 2.2. This is an application level protocol following the request-response archetype where the requester gets information from the provider based on the requester's request.

2.5 Overview of SOAP

SOAP is a lightweight XML-oriented messaging protocol for sending encoded information across the network (Asadollah and Chiew, 2011). Prior to their dispatch, web services request and response messages are converted into a more portable format - XML that can be sent across the network. SOAP is an effective way to access web services and send information via the distributed systems. Although, DCOM and CORBA are conventional object middleware of communicating messages in distributed system but are not designed to work with HTTP. SOAP communicates XML messages because of its platform, protocol and language independence.

University of Malaya

SOAP as a messaging protocol coordinates programs operating on distinct platforms and different environments to communicate and operate together across multiple protocols such FTP, SMP and HTTP (Kalyani, 2012). Figure 2.3 shows the SOAP messaging structure.

Figure 2.3: SOAP messaging structure (Newcomer and Lomow, 2005)

i. **SOAP Envelope:** Always use as <envelope> in the SOAP messaging, is the core element in all SOAP messages. SOAP <envelope> is a highly flexible basic unit of message exchange from one SOAP processor to another. It is comprised of two parts: the non-compulsory <header> and a compulsory <body>.

ii. **SOAP Header:** The <header> of the SOAP <envelope>, although optional but is utilized in pushing application associated information along the message path and to be processed by the SOAP. The associated information is not payload related and are organized in Header blocks with individual defined schema.

iii. **SOAP Body:** This carries the actual XML-encoded application's vital information. The <body> is a compulsory part of the SOAP envelope as it carries the payload. The information encoded in <body> is the actual information intended to reach the receiver. The WSDL document is used to define the schema of the SOAP body.

iv. **SOAP Fault:** The SOAP <fault> as it implies, is a sub-element part of the SOAP <body>. Its role is to track and report errors occurred during the processing of any SOAP message. The <fault> is generally used for error trapping and reporting. Major work of <fault> is to identify wrong formatting or non-existing method call.

2.5.1 SOAP Message Binding

Binding is the real protocol and message formats for the operations and messaging described for a specific port type. The SOAP binding composed of system components that permit SOAP messages to be efficiently exchanged by means of the transport protocol. These mechanisms define the format of the message and the underlying protocol specifics to a web service. The SOAP binding element comprised of two attributes; style attribute and transport attribute (Simon, Goldschmidt, & Kondorosi, 2014).

2.5.2 SOAP Message Binding Style

SOAP predominantly uses XML for web service messaging transmission. The SOAP WSDL document is generally used to describe the web services. All web services created by SOAP protocol use either document or RPC message style for data serialization. Nonetheless two more modes: literal and encoded are also added to aid in marshalling the application objects (Simon et al., 2014).

University of Malaya

i. **RPC** – style: This style is well standardized and a less complex style in message serialization. It uses method call to a remote object based on parameter passing that makes calling a web service easier since the method is part of the application code. It is a tightly coupled technique and changing parameter generally influence the whole definition of the web service.

ii. **Document** – style: There is no standard rule on how SOAP message is formatted which allow no restriction of how SOAP body will be constructed. Its flexibility tolerates external XML as well as its schema to be simply included to the body. Parameters and other part of application structure can be altered without affecting the web service definition, for the fact that document-style is loosely coupled.

iii. **Encoded**: It follows the SOAP rules of message encoding by wrapping the message in the body element prior to dispatch to the host. Although, just like document style, it does not have any definite standard of defining its schema, it insures that body serialization is properly accomplished. This style uses the XML to marshal its data.

iv. **Literal**: In this style the message is marshaled according to the schema that is already defined by the WSDL document of the web service. This gives the client knowledge of how the individual message is formatted. The schema here is well defined and the abstraction defines how the input and output has to contain the formatted message.

2.6 SOAP Performance Improvement

The SOAP performance improvement has been experimented on a variety of data to justify its significance in web services. Server-side and Client-side performance techniques are deployed in attempts to improve the performance (Massimiliano et al.,

University of Malaya

2013). Also, performance is also targeted to be improved by compressing the message exchanged across the network.

2.6.1 Client-side Approach

With the aim to increase the performance of Web services, some studies like Arteaga and Zhao (2014), Banditwattanawong and Uthayopas (2013), Bonetta, Peternier, Pautasso, and Binder (2012), Bzoch and Safarink (2013) and Sriwiroj and Banditwattanawong (2015) tilted their studies to client side in their quest to decrease the transaction response time. In client-side approach, the web services rely on the client device to perform most of the computing operation. This approach is efficient when similar SOAP messages are revisited by the client's web services. The parsing and calling of the SOAP messages always consume a lot of time and bandwidth (Du, Zhao, Han, & Li, 2013). The client checks and acknowledge if the requesting message is in the cache, else obtain it from the server. This process hugely reduces the response time and network traffic. Here, client-side caching and differential serialization are the approaches used in improving the SOAP performance.

To show the effectiveness of using caching in the client side, Kiran and Andresen (2010) used HTTP as the transport layer, implemented an RPC-style rather than the message based for the caching. This reduces the network traffic by reducing the number of accesses by the client fetching same data from the server. The implemented result was remarkably good as it improves the round trip (per second) by 800% less compared to CORBA or Java RMI. In spites of its yielded overwhelming performance, this method posed a challenge on how the data at the client side will be updated and how frequent the updates should be performed. These challenges were not solved by the authors and therefore the response time performance improvement cannot be completely established.

University of Malaya

Bzoch and Safarink (2013) conducted a research on client-side caching to compare caching policies: LRD, LRU, LRU-K, MRU, LRUF-SS and LFU-SS in system application and to choose the optimal one. The researchers developed algorithms referred to as caching policies and used simulator to test and proof each policy using a simulated environment with equally sized data blocks. The authors experimented set of caching policies in order to identify and choose the best caching policy for adoption. Experimenting with small (1MB – 5MB) and large (16MB – 512MB) files with different number of requests, users access the files from the server and cache the files in the data blocks and the hit count in each policy were documented. The result revealed that the best caching policy is the LFU-SS. The implementation of this research is primarily targeted for mobile systems. The study does not provide a method for garbage collection or deletion of old unwanted cached data. This will lead to inconsistency of data and computational error due to impending unwanted data.

A study by Juric et al. (2006) compared and evaluated the performance factors of cloud and traditional services on web servers. In their studies, they subjected virtual servers. More virtual machine in the cloud realized lower response time and better throughput, but one virtual machine showed no difference with the physical server. Nonetheless, their study found that increasing the number of physical servers does not increase the throughput. The performance effect on the transaction of the SOAP messages was not realized. More precisely, this study identified only the influence of having more virtual machines as a factor to improve response time and throughput. The author did not describe the implementation of the web services clearly.

University of Malaya

2.6.2 Differential Serialization

Serialization is the method of translating an application object into series of formatted SOAP/XML messages that can be transported through channel from the client side to server side (Mallad, Murphy, and Deng, 2017).

In differential serialization, when services are exchanged from the server, the web service in the client side tracks any changes in the data structure of the previous exchange and acts only on the objects with new references. This process improves the response time, round trip and computation load at the client side. Attempting to improve the SOAP performance, Abu-Ghazaleh and Lewis (2005), Abu-Ghazaleh, Lewis and Govindaraju (2004), and Suzumura, Takase and Tatsubori (2005) used the differential serialization approach.

Abu-Ghazaleh et al. (2004) conducted a differential serialization study by developing an algorithm that facilitates the reuse of SOAP message structure at the client side. The algorithm creates a procedure for message templating. Message structure is determined and saved as templates for reuse by remote web service with similar or closely similar structure with the saved template. Contents, size and associated ID are matched to get the similarity. The outgoing message uses the structure or part of the structure of the saved template rather than generating its own structure. This process encourages reuse of computational procedures and reduces the computation overhead due to regeneration. The issue here is, at long run each remote web service will have its own templates and continue to grow others as the structures gradually varies. This delimits the applicability of using this style.

An improvement over the work by Abu-Ghazaleh et al. (2004), was carried out a year after by Abu-Ghazaleh et al. (2005), adding to the previous work with an algorithm for resizing message fields. In the situation when the client-side new message to be

University of Malaya

differentially serialized is larger than the previously serialized/differentially serialized message, the algorithm allows the borrowing of space from neighboring fields (stealing) to accommodate the larger new data. The research was well implemented even though failed to address a situation of sending several large messages simultaneously. The authors acknowledged that stealing from neighbors can tend to cause unpredictable error according to the authors. Also, the implementation lacks the economic ability to downsize previously upsize field when accommodating smaller message, thus some spaces will be redundant. From the implementation, it remains impractical to determine the behavior of the sent message at the server side for the researchers excluded the full utilization of the server side completely by creating only a dummy SOAP at the server.

2.6.3 Server-Side Approach

Research by Aali and Farkhady (2011) used the server to enhance the communication performance of web services. Large portion of web services computation is controlled by the server. The server web service bonds the communication by sending messages to the client web service based on requests. In this regard, research effort is vested at the server side to improve the SOAP performance. Server-side caching and differential serialization are techniques used at the server side in achieving the performance objective.

2.6.4 Server-Side Caching

The server-side caching is a process of temporarily holding the active computational data at the server which can easily and repeatedly be accessed by the clients. This is efficient in improving performance by avoiding computation at the remote web services. In this approach, most computational costs are engaged at the server. The response time is minimized with only processed and computed messages are transported to the client

University of Malaya

web services. The response time perceived by the requester is faster and hence minimized.

Aali and Farkhady (2011) demonstrated the application of server-side caching by caching a compressed SOAP message. Messages were cached at the server and sent to the requesting remote web service to be consumed. In this implementation, the clients are physical computers connected by 10 MBPS hub. File size with different sizes (0.2MB – 7.2MB) were sent to the client and corresponding time taken was recorded. Taking into cognizance the result, when SOAP messages were sent with caching and without caching, the former indicates faster response time. The overall result yielded that the response time and the throughput improved as the cached and compressed message size is increased. The authors did not provide details of their implementation, hence is difficult to understand their assessment of the overall research. The performance of this implementation was manually captured despite that the authors used host and client machines for the implementation.

2.6.5 Differential Deserialization

Deserialization is the process of converting back the application object from unformatted SOAP/XML messages that has been transported through channel from the client side (Mallad et al., 2017). In differential deserialization, the web service in the server side track any changes made to its data structure that made it differ from the previous data structure and process the different region that is not previously available. This process reduces the response time, round trip and computation to be performed at the server side. An attempt on this method was made by Abu-Ghazaleh and Lewis (2006). In the method, incoming message is deserialized and linked to the internal automata. The message is also matched with existing to check for element similarity. The SOAP engine will only process the dissimilar region of the linked application

University of Malaya

object through partially deserializing and concatenating them and reset the fields. In the experiment, the throughput was measured by running request threads 30,000 times and results were obtained and recorded. Comparing the result for deserialization request with and without differential technique, the former recorded 288% throughput than the later.

From the experimental procedures, the web services application objects tend to have repetition of some elements hence it will not give an optimal solution since the repetition number can be different for each request. Nonetheless, problem may tend to impede the web services as the size of the automata may grow as number of requests increase and there was no procedure for garbage collection in the implementation.

In an effort to avoid absolute deserialization, Abu-Ghazaleh and Lewis (2005) developed a strategy of using a checkpoint algorithm that checks the state of the message deserializer at some points. The light-check-point method (LCP) applying to the normal differential deserialization, uses checksum comparison to determine whether an incoming message is the same with the previous or has some similarities with the previous to facilitate the deserializer to avoid the portion with the similarity. Each LCP has reference to the previous checkpoint that contains states it shares with others. The one with the same previous checkpoints are then referred to as same group. With these checkpoints, difference is calculated by identifying the changes that have been made to the incoming message by mismatched.

The authors subjected message sizes (0.35MB – 3.5MB) and applied to the regular differential deserialization and LCP and checkpoints were created at strategic point of n-bytes message sizes. The memory usage by LCP indicates low usage which conversely implies high performance in deserialization time over regular differential deserialization. LCP needs only 3% of the memory needed by the regular differential

deserialization. This study experimented with low byte payloads. Besides, calculating checksums is prone to error generation, especially when dealing with scientific data from computations.

In summary, these techniques have their individual shortcomings, though, depending on the performance trade-off they can be applied to different scenarios.

Serialization/deserialization requires encoding and decoding and this causes a lot of computing overhead at both ends. XML parsing tends to be slow owing to high processing overhead. Parsers in XML act heavily on the messages to process and this demands a lot of resources.

Caching is not suitable if the XML message demands constant update or is not large enough. It also tends to be slow if the resource is not found or initiated in the cache that therefore need to be processed.

The trade-off in these techniques is that delivery time is more concerned than the processing overhead.

2.6.6 Message Encoding/Compression Approach

Message compression is a technique for reducing the size of message required to be stored or transmitted across by using an encoded method (Liu, Mei, Wang, O'Neill, & Swartzlander, 2018). According to Nguyen, Nguyen, Duong and Snasel (2016), in text compression, lossless methods are adopted to decrease the original XML message size. Compressed message is decompressed at the other end of the network and this achieves less transmission time and less hosting space.

Introducing a new approach in experimenting a SOAP XML compression, Al-Shammary and Khalil (2010) adopted a combination of fixed length and Huffman

University of Malaya

encoding in their approach. The encoding is supported by XML tree and binary tree to remove the message closing tags. The messages were gathered in view of comparable messages into element bunch structure. 160 XML records were utilized for the test. The messages were isolated into 4 equivalent groups, containing 40 messages each and sorted into small, medium, large and very large bytes individually.

The fixed-length and Huffman encoding algorithms were applied to these messages groups and the results were observed. The compression ratios for small was 2.175, medium was 3.15, large was 7.15 and very large was 11.125. The method proved to be efficient for large and very large messages, but subjecting the method to very small and small bytes tends to have no effect and sometimes incur computation overhead. The problem with their approach is that it is always hard to predict absolute performance because they used lightweight data for the experiments. This will reduce the ability of the technique in identifying the compression ratio.

Complementing the work of Al-Shammary and Khalil (2010), a dynamic-clustering based aggregation was implemented by Abbas et al. (2014). They used the common Term Frequency-Inverse Document Frequency (TF-IDF) as the weighing factor with Euclidean distance method for estimating the degree of similarity among SOAP messages. The messages were grouped based on similar messages into dynamic cluster form. After clustering the authors applied Huffman compression algorithm to compress the XML messages clusters as one compact message. 160 XML documents were used for the experiment. The messages were divided into 4 equal groups each containing 40 messages and categorized into small, medium, large and very large bytes respectively.

In order to obtain the compression ratio, the total size of the XML messages is divided by compression size of the XML messages. The result revealed that the compression ratio of the small bytes was 2.91, medium was 8.35 while large and very

University of Malaya

large groups were 17.36 and 19.89 respectively. The conclusion is that the bigger the message size, the better the compression. Therefore, having large size of XML message yields a better compressed result. It is hard to know the capability, especially to determine the compression ratio because they used lightweight data for experiments. This will reduce the ability of the technique in identifying the compression ratio. Besides, Huffman encoding is not suitable for long characters (Al-Shammary et al., 2010).

By and large, compression always yields faster transmission time but the compression application mostly adds overhead and take on part of the system memory. Despite that, compression approach has more benefit than the other approaches. Although the transmission was also successful, the researchers did not provide solutions to avoid loss of data during transmission.

Lam and Rossiter (2013) proposed a framework for streaming compressed multimedia contents. The authors implemented and simulated SOAP over HTTP web services with compression to stream the multimedia data. The experiment was simulated using nS-2 network simulator. The outcome of their study revealed that SOAP over HTTP binding with binary XML scheme compression has produced best performance compared to a basic multimedia scheme.

Closely related to this work is the study by Kadouh and Albashiri (2014). The researchers developed an algorithm and deployed it to compare the effect of their algorithm on request/response operations on SOAP over HTTP web services. The authors applied calculator application with different operations and of sizes between 314 and 400 bytes against response times from 10503 to 19088 milliseconds. The services also used image files of sizes between 1030 and 1993 bytes which yielded response time between 121 and 285 seconds. The overall result revealed that the

University of Malaya

transmission with the algorithm yielded a better response time for both XML messages and images.

Another research by Chiu, Devadithya, Lu and Slominski (2005) created a generic schema for binary XML exchange called Binary XML for Scientific Applications (BXSA). Unlike SOAP/HTTP which is not a suitable for transmitting binary data, BXSA was bounded to TCP as the transport protocol. The authors compared the performance of the BXSA against the normal HTTP, XML-HTTP and FTP binding. The protocols where subjected to data size 1000 bytes shows that BXSA maintained a constant linear response time of averagely 11,000 milliseconds throughout while Grid-FTP is incredibly high with response time around 230,000 milliseconds through irregular transition levels. But the same schemes subjected to large bytes revealed that BXSA has response time of 205,128 milliseconds when the size is 1.4 MB. When the data is as large as 56 MB, the response time is 175 milliseconds. The overall result suggested the BXSA has better performance than the other schemes. Although the BXSA has demonstrated the ability to exchange large data size, the authors did not consider how data loss can be avoided. Nevertheless, the response time seems to be low especially for the small data size.

2.7 Message Streaming

Message streaming is a technique for steady and continuous sending of message from one point as provider to another point that receives (Isaac and Devi, 2014). This involves the uninterrupted transfer of message from a provider to the receiver. Here, the receiver application processes the message sent in a stream style where the excess messages are buffered before processing. Transmission and continuous delivery of high-volume message is always challenging and time demanding, hence it requires an exchange protocol in the SOAP binding (Isaac and Devi, 2014).

University of Malaya

Studies on streaming has been conducted by Bou, Amagasa and Kitagawa (2014), Shams and Sarkar (2014), Isaac and Devi (2014), Kanoun and Schaar (2015), Nakamoto and Akiyama (2015), Sarkas et al. (2008) and Zhang et al. (2008). The studies traverse in the same vein to improve the quality of output message by leveraging the application or the bandwidth. With the exception of Isaac and Devi (2014) that worked on multimedia message streaming, all other authors primarily worked on message-oriented streaming process. Limitation to their implementation holds that their works rely in the ability that the site must have an installed application to connect to the server or to handle the streamed message for processing. Appel, Frischbier, Freudenreich and Buchmann (2012) and Isaac and Devi (2014) engaged the Web Services to implement the message streaming.

In their work, Isaac and Devi (2014) argued that current standards were not sufficient in handling large files streaming, especially multimedia message. Therefore, they opined the use of SOA, which they implemented through the introduction of Message Exchange Protocol (MEP) in SOAP over HTTP.

Another application of Web services in streaming was implemented by Appel et al., (2012). Series of occurring events are collected as message is relayed over the Internet. Embedded message tends to be unpredictable in volume therefore needs a constant and continuous procedure of delivering. To obtain solution for the cumbersome events message, the authors used application and transport layer of SOA to capture process and transport the message in form of streams. Though the final result performance was not yielded but they guaranteed a secured streaming from denial of service.

Large message streaming was studied by Girtelschmid, Steinbauer, Kumar, Fensel and Kotsis (2014). Their work forecast on the future delivery of mined message of the emerging Big-message and cloud computing which are geometrically increasing. They

University of Malaya

implemented a big message streaming platform in anticipation that the rule engine will be able to lessen the workload of a high-volume message and hence improve the performance and guarantee the delivery.

Message streaming simplifies the transmission of continuous message over the Internet. Studies have been conducted to improve the performance in terms of throughput and quality of delivered message. Different techniques and protocols were used by different authors to improve the performance. In order to have a guaranty delivery with improved performance, Web services was suggested by some authors like Appel et al., (2012) and Isaac and Devi (2014) because of its reusability, growth and security in request-response operation.

2.8 Limitations of the Approaches in the Literature

The study referenced in the literature have improvement on the performance of the SOAP. But limitations associated with the approaches have been identified and need to be solved to further improve the performance of the SOAP. Table 2.1 shows some important research on the SOAP performance improvement approaches.

Generally, the approaches used in these studies experimented their implementations on small SOAP payloads. Using lightweight message as the running data will decrease the ability to identify effectively if these implementations can work well with large payloads. The implementations were experimented using RPC over HTTP which is absolutely synchronous in nature, hence, cannot work in general situation, especially with the advent of big data and cloud computing.

University of Malaya

Table 2.1: Some worthy works on SOAP performance improvement approaches

Title of research and Author(s)	Approach	Method and findings	Limitations
Abu-Ghazaleh, N., Lewis, M. J., & Govindaraju, M. (2004). Differential serialization for optimized SOAP performance.	Client side serialization	Developed an algorithm that facilitates the reuse of SOAP message structure at the client side by creating templates. Similar messages use the same template.	The outgoing message uses the structure or part of the structure of the saved template rather than generating its own structure.
Bzoch, P., & Safarink, J. (2013). Simulation of client-side caching policies for distributed file systems	Client side caching	Developed an algorithm and simulate and compare caching policies; LRD, LRU, LRU-K, MRU, LRUF-SS and LFU-SS in system application and to choose the optimal time. The result revealed that the best caching policy was the LFU-SS with 4% and 2% respectively in lower and higher caches.	The study does not provide a method for garbage collection or deletion of old unwanted cached data. This will lead to inconsistency of data and computational error due to impending unwanted data.
Abu-Ghazaleh, N., & Lewis, M. J. (2005, 13-14 Nov. 2005). Differential checkpointing for reducing memory requirements in optimized SOAP deserialization	Server side deserialization	Developed a strategy of using a checkpoint algorithm that checks the state of the message deserializer at some points. Uses checksum to determine whether an incoming message is the same with the previous one, else the deserializer will process the dissimilar portion.	This study was experimented with low byte payloads. Also calculating the checksums is prone to error generation, especially when dealing with scientific data from computations. Overhead is also a factor to consider.
Aali, S. H., &Farkhady, R. Z. (2011). A Combination Approach for Improvement Web Service Performance	Server side caching	These authors cached a compressed message (7.2MB) and sent to the client. The overall result yielded that the response time and the throughput improved as the cached and compressed message size increases.	The authors did not provide details of their implementation, hence is difficult to understand their assessment of the overall research. The performance of this implemention was manually captured despite that the authors used host and client machines for this implementation. This might be error prone.

University of Malaya

Abbas, A. M., Bakar, A. A., & Ahmad, M. Z. (2014). Fast dynamic clustering SOAP messages-based compression and aggregation model for enhanced performance of Web services	Compression	The messages were grouped based on similar messages into dynamic cluster of 4 groups and applied Huffman compression algorithm. In order to obtain the compression ratio, the total size of the XML messages is divided by compressed size of the XML messages. The result revealed the compression ratio of the small bytes as 2.91, medium as 8.35 while large and very large groups has 17.36 and 19.89 respectively.	It is always hard to predict absolute performance when lightweight data are used for the experiments. This will reduce the ability of the technique in identifying the compression ratio.

2.9 Data Compression

Data compression is the compacting of information into smaller representative without missing its quality (Sayood, 2002). The compressed data is represented in digital form to save space or transported over a network (Hong, Zhang, Wang, Li, & Liu 2016; Kruse and Mukherjee, 1997). The data is converted back to its original form when demanded. The process of converting the data back to the original form is termed as decompression (Hong et al., 2016).

2.9.1 Types of Data Compression

Data compression is divided into two techniques; the lossy compression technique and the lossless compression technique. Each technique has its advantages and disadvantages, and suitability.

The lossy data compression techniques convert data to a set of digital bits while ignoring the less important parts of the data. In these techniques, the exact replica of the original data cannot be recovered. This type of compression is generally applied in audio, video and image data. Widely used types are Huffman coding, LZW, JPEG and MP3.

University of Malaya

The lossless compression techniques convert data to a set of digital bits without compromise to lose any part of the data. Hence, the reconversion of the data recovered the exact copy of the original data. Example of lossless data compression are text files and programs that hold every single bit as vital. Popular types of lossless compression are Shannon-Fano, Huffman, Arithmetic Code, LRE, LZ77 and LZ78.

Lossless compression methods are used for text-based data compression (Sayood, 2002). Texts files such as documents and programs are very essential, and ignoring a little part can distort the meaning and the content of such files (Kumawat and Chaudhury, 2013). Hence, retaining the original content of the compressed file is essential.

2.9.2 Text compression

Text, like other forms of data is compressed with intend to save space or to decrease bandwidth consumption over a network. Text are compressed using a lossless method of compression to avoid data loss (Oswald and Sivaselvan, 2017). Any part of the text data is important in the compression/decompression, and therefore the decompression should lose no piece of the original text data (Hansen and Lewis, 2018; Memon and Sayood, 1995). Widely used techniques in text compression are the statistical text compression and the dictionary text compression (Bulus, Carus, & Mesut, 2017).

2.9.3 Statistical Text Compression

This type of compression is comprised of two fragments: the model and the coder. The model part creates the statistical properties of the input sequence while the coder part compresses the input sequence obtained by the model (Cao Dix, Allison, & Mears, 2007).). The method uses variable-size codes in which the shorter codes are allocated to the symbols frequently appear, while the longer codes are assigned to the infrequent symbols (Sayood, 2002).

University of Malaya

The model is built in the form of a binary tree as probability distribution and the leaf nodes are the symbols which probability values are sorted in ascending order. In the process, two symbols that appear to have the two lowermost probability values are joint to produce a new parent node as a composite symbol with the probability sum of the two symbols as its new probability. The process is reiterated with the new list until the composite node can no longer be reproduced. In the decompression, the same binary tree is decoded to produce the symbols again.

In statistical method, the knowledge of the frequency of certain part of the data is known in advance (Liu et al., 2018). The disadvantage of this method is that the tree acts similar to a dictionary that at one time in the beginning is encoded and this creates an initial overhead of the process.

Widely used statistical compression algorithms include the Arithmetic Coding, Run-Length Encoding, Shannon-Fano Coding and Huffman Coding (Shanmugasundaram and Lourdusamy, 2011).

2.9.4 Dictionary Text Compression

In dictionary text-based compression, the input is a set of symbols. The symbols are compressed by using index to search and replace symbols with pointers referring to previously encountered symbol contained in the dictionary. The variable length input is substituted by a reference to the symbols existing in the dictionary. Using the indices to substitute lengthy variable set of symbols provides an efficient and manageable approach to compress large data (Ghosh, and Ganguly, 2015; Larsson and Moffat, 2000). There are two algorithms used in the dictionary compression, namely the static and adaptive algorithms.

University of Malaya

The static dictionary compression is applicable, in most cases to large and fixed set of symbols. According to Bell et al. (1990), this type of compression is permanent and the dictionary has a prior knowledge of text to be compressed which makes it to be language specific.

On the other hand, in the adaptive compression, the dictionary is formed from the previously encoded sequence of symbols. Then, further incoming symbols set are compressed by the dictionary statically. Repeated symbols pattern is recognized and represented in the dictionary efficiently by the encoder. The encoder indexes and replaces the symbols by the index of similar symbols previously encoded. The adaptive dictionary is built by addition of new incoming symbols and adjusts its size by removing the leftmost encoded symbols from the dictionary. Most common adaptive algorithms are the LZ77, LZ78 and LZW. Other variants like LZ4, LZSS, LZH and the LZR are also applied in the text-based compression.

Considering the two lossless compression techniques, the disadvantage of statistical compression is that the method incurs initial overhead due to the dictionary-like advance input (Hansen and Lewis, 2018).

In the dictionary method, the decompression is faster and easy owing to the fact that the encoded symbols are randomly accessed and retrieved from the dictionary based on the relative positions. Using indices to represent variable length symbols provides an efficient and manageable way of handling large number of symbols at the same time (Abbas et al., 2014).

The adaptive dictionary compression method is more appropriate than the static compression method. Adaptive compression compresses multiple symbols and encodes

University of Malaya

the symbols on-the-fly. Contrarily, in the static dictionary method the symbols are predefined and constant (Liv, Wang, & Zhong, 2015).

Considering the text-based input to be used in experimenting the SOAP Web services, this research adopted an adaptive dictionary technique - the LZ77 (Ziv and Lempel, 1977) algorithm for text compression. The LZ77 is the most popular adaptive dictionary method and most widely used and effective text compression algorithm (Policriti and Prezza, 2016). Study by Kumawat and Chaudhury (2013) and Policriti and Prezza (2016) suggested that dictionary compression is effective in text compression.

Williams (1991) modified an LZ77 algorithm based on Fiala and Greene (1989), with fewer line of code but faster than the LZ77. The algorithm uses the Lempel technique and a hash table for the data compression. It checks for a match of 3-byte length by checking the dictionary throughout, and only copy a byte when it cannot get a copy of 3 bytes. The hash table of 4096 pointers was used to map the 3-byte key to a single pointer. The pointer can then point to any position in the Lempel for 3-byte matched. The decompression processes item one by one which may be a byte or bytes and adds to the output as a single byte by locating the offset and length pair for bytes.

To test the performance of the modified algorithm, the researchers tested it on high level language using C by comparing it with the A1 algorithm. The result showed that the modified algorithm is 10 times faster in execution time than the A1 algorithm. When used on low level using 68000 assembly language by dividing each file in of block 16k, the result showed that the modified algorithm running on assembly language is best for data of small blocks

University of Malaya

Unlike Williams (1991), a study on another variant of LZ77 was carried out by Weimin, Huijiang, Yi, Jingbao and Huan (2008) to improve the decompression of data. The authors proposed an algorithm that improve on the decompression of data. The algorithm is a hybrid between LZ77 and Huffman codes with Adaptive Markov Chain (AMC). The LZ77 search for a match of 3 bytes only and if found, a token is assigned to it with an added bit of 1 unit. If no 3-bytes match found, the algorithm encodes the byte without modification and a bit of 0 unit is added to the encoded byte. The bit differentiates the token and the non-modified byte during decompression.

When decompressing the data, the bytes from the Huffman codes are converted to bits by the AMC and its probability is checked. This reduces data redundancy and lower memory overflow. The result revealed that the algorithm performed better for image and video files.

Work on optimization of LZ77 performance by Kumawat and Chaudhury (2013) uses double [l, c] instead triple [o, l, c] whenever the length of the match is equal to the offset. This algorithm work exactly similar to the LZ77 except encoder must always check and compares the length and the offset for similarity. If found, a delimiter is added and the offset will be discarded. The length and the codeword will instead form the entry [l, c]. This reduces the number of bits to be entered into the dictionary. The decompression is slower compared to the LZ77 because the pointer has additional of checking the triplets and the doublets entries during decoding. The overall performance is slow but the algorithm produced a better compression ration better than the conventional LZ77.

The asymmetrical deficiency of the LZ77 was countered by Mahmood, Islam, Nigatu and Henkel (2014). The study proposed modification on the LZ77 algorithm by introducing a bi-directional method of reading the algorithm. The encoding process is

University of Malaya

very similar to the LZ77 except that when a match is found, the offset and the length of the match in both the search buffer and the look-ahead buffer and the codeword in the look-ahead buffer with a flag bit are output. The experiment used various file formats of different sizes to compare the two algorithms. The result shows that the modified algorithm has a good compression ratio, especially for symmetrical data.

2.10 Summary

The review of literary works in this chapter is predominantly concentrated on improving SOAP performance. Different approaches such as caching at server and client side, and differential serialization and differential deserialization as well as message compression have been critically assessed and optimal results were analyzed.

Researchers have made attempts to improve the web services performance through reducing response time by decreasing the number of time message will be visited or through the reuse of computational components. Some techniques were implemented to compress the encoded message before sending to the receiver. The compression technique makes the transmission to be lighter and utilizes low bandwidth, and eventually increases the throughput of the web services.

The researchers while tracing their implementations and results did not address the issue of improving the performance of SOAP when subjected to large payloads. To qualify this assertion, it is significant to model an improved SOAP technique for high volume message exchange capable of guaranteeing the delivery of the message with low response time and less overhead.

University of Malaya

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

The primary goal of this research is to answer the research questions that relate to SOAP performance enhancement as stated in Chapter 1. This chapter explains the methodology used in accomplishing the goal of the research. It discusses the hardware and software arrangements for the experiment and how they are configured to answer the research questions presented in Chapter 1. Web services were implemented and transacted over two transport protocols: HTTP and JMS. The rest of the chapter explains the process. This chapter is organized into seven (7) sections: research conception, review of related literature, identification of research gap, system requirement analysis and design, experimental setup, data compression algorithm modification, and system implementation and execution.

3.2 Research Conception

The idea of this research "SOAP performance enhancement for high volume messaging", was conceptualized from an interest in reading on various data exchange techniques. The initial concept encompasses the knowledge acquired on data exchange at the endpoints or over the network. After further reading, it came to light that very few studies have been conducted on how data is processed and utilized at the transmission endpoints. It is understood that a good knowledge of this area will support study in the area of cloud computing, big data, and computing servers.

3.3 Review of Related Literature

The literature review formed the fundamental start-up of this research. Topics on Web services were chosen and relevant keywords formulated for the search on reputable journal databases. Several searches were performed in Web of Science, Association of Computing Machineries and other high-quality databases. The articles relevant to the

University of Malaya

topic were downloaded and read through. Figure 3.1 shows how the flow of the literature review process for this research was conducted.

The topics were further narrowed and the relevant articles were filtered into survey/review and technical/empirical papers for study. The survey/review articles were studied to get a focal point for the research. Topics on SOAP Web services was chosen, more precisely on the performance of the services. After getting the research direction, quality related empirical articles were thoroughly studied. The procedures, methods, approaches, and techniques used in the methodologies of the previous researches were considered and comprehended. The methodologies were compared to the findings of each research to understand the empirical evidences. The key findings were presented in Table 2.1 in Chapter 2.

3.4 Identification of Research Gap

The research gap for this study was established based on the empirical findings of the synthesized literature. The findings identified a key research area in web services that seems to be useful but not much explored. The area is the performance of SOAP Web services enhancement. Final stage of the literature review process is the identification of research gap as shown in the last stage in Figure 3.1.

University of Malaya

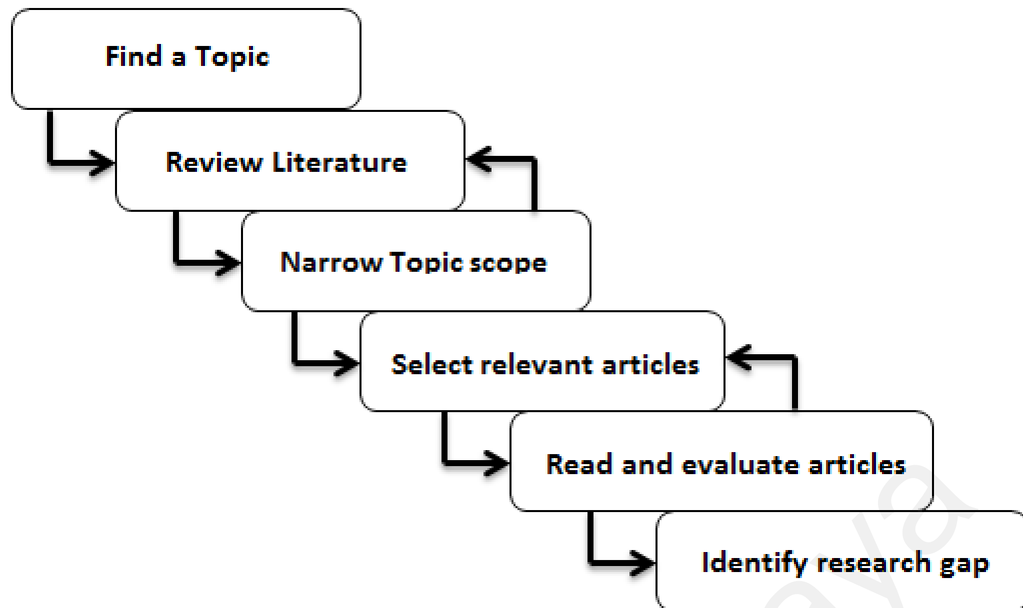


Figure 3.1: Literature review process for the SOAP performance enhancement for large volume messaging

3.5 System Requirement Analysis and Design

After identifying the research gap from the literature review process, this stage proceeded to identify the way for achieving the improvement of the SOAP Web services performance. To effectively obtain useful experimental results, system requirements were carefully identified. The solution was obtained by analysing and designing the requirements based on the research objectives. Unified Modelling Language (UML) 2.0 was used to model the analysis and the design for the system requirement as presented in Chapter 4.

3.6 Experiment Setup

This section presents and discusses the setting up of the experiment for the Web services transaction. It provides detail procedures of hardware and software selection and usage for the research.

3.6.1 Hardware Setup

To achieve the required goal, the experiment used a high specification personal computer (core i7) as the machine for this research. The machine was used as a virtual provider and consumer for both services in the experiment. Hence no external hardware use in this research. Table 3.1 shows the hardware system requisites used in the implementation of this research. The section, nonetheless gives the details of how the system is set up and unified to work and address the problem statement of the research.

Table 3.1: Hardware requirements

Sno.	Component	Specification
1	Processor	Core i7; Duo core @ 2.30ghz
2	Memory	8GB
3	Hard disk	1 terabyte

3.6.2 Software Setup

The software setup uses windows 10 as the operating system and J2EE with Weblogic server as the programming environment and virtual server respectively. Table 3.2 shows the detailed components used.

Two SOAP Web Services were designed to run on two different transport protocols namely HTTP and JMS. Hence the two systems - SOAP over HTTP as the benchmark system and SOAP over JMS as the prototype system. Each system comprises two communicating parties which are the web service server/provider and the web service client/consumer. The client requests for service from the server, in turn the server responds by providing the requested service. For each transport protocol, two data formats: normal and compressed payloads were exchanged between the server and the client. Figure 3.2 depicts the request/response communication scenario.

University of Malaya

Table 3.2: Software system specifications used in the study

Sno.	Component	Specification
1	Operating System	Windows 10 pro edition
2	Programming environment	Jdeveloper (Java EE)
3	Application server	Oracle WebLogic 12C, version 12.1.3

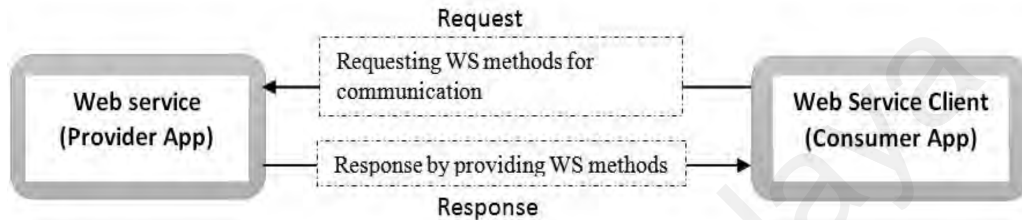


Figure 3.2: Web services request/response. Services are provided by the server based on client request

3.6.2.1 Benchmark System Setup

In the benchmark setup, two SOAP web services were created: the server web service and the client web service. Table 3.3 shows the SOAP over HTTP web services comprising the classes and the methods used in the implementation

The server contains the web service provider class, the compressor class and the controller class for handling function call for normal and compressed methods. The server establishes the proxy contract to bind the connection to the HTTP while the compressor class encodes and compresses the input message.

On the other side, the client web service contains web service consumer class and decompressor class. The methods in the client use the service and port name of the server to establish the WSDL contract and invoke the web service for communication. The decompressor class decodes and decompresses the input message received from the server.

University of Malaya

Table 3.3: Benchmark Client-Server SOAP web services with HTTP protocol

Web service server *Application name: WSProvider* *Web service name WSProviderApp*	**Web service client** *Application name: WSConsumer* *Web service name: WSConsumerApp*
Class: WSProvider **Methods**: a) WSProviderPort b) messageGenerate c) timer **Class**: compressor **Methods**: a) searchBuffer b) appendBuffer c) readBuffer d) increaseBuffer **Class**: controller **Methods**: a) compressedMessage b) normalMessage	**Class**: WSConsumer **Methods**: a) WSReceiverProxy b) WSPProviderService c) onMessage d) timer **Class**: decompressor **Methods**: a) compareBuffer b) appendBuffer c) increaseBufferSize
The server provides the WSDL for the web services' contract.	The client lookup the service name and port name to establish the contract of the WSDL and invoke the Web service. WSProviderService service = new WSPProviderService(); WSProvider port = service.getWSProviderPort()

3.6.2.2 Prototype System Setup

The prototype consists of two SOAP web services: the server web service and the client web service bounded over JMS. The SOAP over JMS is shown in Table 3.4 indicating the web services with the classes and the methods used for the implementation.

Unlike HTTP, JMS is a message centric API hence the transport holds at the API level (Al-Rassan and Alyahya, 2015). The JMS protocol allows connection, queue, sender and receiver to be defined at the implementation level.

University of Malaya

The server contains three classes: controller class, provider class and compressor class. The controller class holds function calls for normal and compression message. The server establishes the binding contract to bind connection to JMS using a connectionFactory and queueSender inbuilt methods of the JMS. The compressor class encode and compress the input message to be sent.

The client web service contains two classes; normal class and decompressed class. The client uses connection factory created at the JMS protocol to invoke the server. The client then uses the connectionFactory and the queueReceiverer to receive the requested WSDL file to be consumed. The decopmressor class decodes and decompresses the input message received from the server.

University of Malaya

Table 3.4: Prototype Client-Server SOAP Web Services with JMS Protocol

<table>
<tr><th>web service server
Application name: WSProvider
Web service name WSProviderApp</th><th>Web service client
Application name: WSConsumer
Web service name: WSConsumerApp</th></tr>
<tr><td>Class: WSProvider
Methods:
a) connectionFactory
b) queueSender
c) messageGenerate
d) timer

Class: compressor
Methods:
a) searchBuffer
b) appendBuffer
c) readBuffer
d) increaseBuffer

Class: controller
Methods:
a) compressedMessageHandler
b) normalMessageHandler</td><td>Class: WSConsumer
Methods:
a) connectionFactory
b) queueReceiver
c) onMessage
d) timer

Class: decompressor
Methods:
a) compareBuffer
b) appendBuffer
c) increaseBufferSize</td></tr>
<tr><td>The server provides the WSDL for the web services' contract and remains connected via;
msgQueue = (Queue) ctx.lookup("jms/MainQueue")
connFactory = (QueueConnectionFactory) ctx.lookup("jms/MainConFac")
msgSender = queueSession.createSender(queue)</td><td>The client looks up the service name and port name to establish the contract of the WSDL and invokes the WS.
msgqQueue = (Queue) ctx.lookup("jms/MainQueue")
connFactory = (QueueConnectionFactory) ctx.lookup("jms/MainConFac")
msgRecevier= queueSession.createReceiver(queue, messageSelector)</td></tr>
</table>

3.7 Data Compression Algorithm Modification

This research modified the LZ77 compression algorithm to encode more symbols. A frequency (2) is used in the look-ahead buffer (LAB) to tag any first two adjacent symbols encountered in the LAB during encoding. The tagging reduces the number of pointers in the dictionary while yielding the same output as the conventional LZ77.

3.7.1 The LZ77 Compression Algorithm

The LZ77 (Ziv and Lempel, 1977) is a lossless compression algorithm using an adaptive dictionary method. The name LZ77 was formed from the research by

University of Malaya

Abraham Lempel and Jacob Ziv in 1977. The algorithm compresses data on-the-fly by constructing a data dictionary as the inputs are read (Oswald and Sivaselvan, 2018).

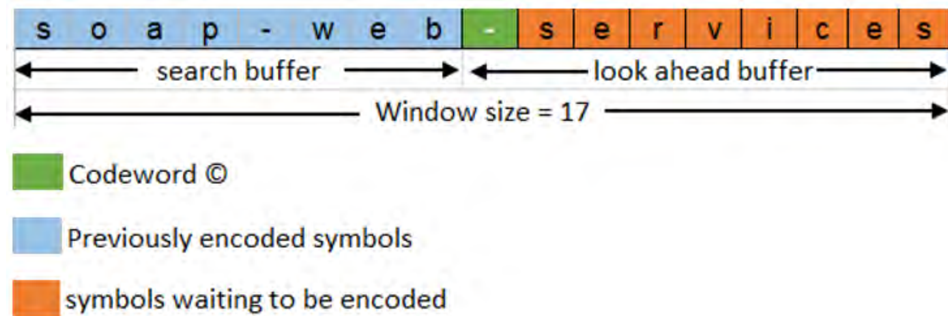
The LZ77 is a universal dictionary-based algorithm for sequential data compression. The algorithm has a good compression ratio with good execution speed than statistical compression method (Oswald and Sivaselvan, 2018; Policriti and Prezza, 2016). The decompression is faster than the compression and starts immediately as the encoded inputs are available at the decoding part of the algorithm (Priyatna, and Mantoro, 2018; Hong et al., 2016). As such, most studies on LZ77 focus on the encoding to improve the execution time or the compression ratio. Each requirement comes with a trade-off, a better compression comes at the expense of execution time and vice versa.

The dictionary is a container in the form of a circular buffer for holding encoded symbols (Salomon, and Motta, 2010; Barrington, and Dechev, 2015). In the LZ77, a technique referred to as sliding window is used in a form of a window having two slides with the left slide for the Search Buffer (SB) holding the encoded symbols and the Look-ahead Buffer on the right slide containing symbols yet to be encoded (Mahmood et al., 2014).

The SB is the dictionary containing the encoded symbols and the algorithm searches the dictionary for a longest symbol match with the beginning of the LAB. When match is found, instead of the symbol to be entered into the dictionary, a pointer to a location for a similar symbol previously encoded in the dictionary is entered.

University of Malaya

Table 3.5: LZ77 Data compression technique showing the search buffer of size 8, look-ahead buffer of size 9 and window size of 17.



The encoding consists of tuple <o, l, c> for any input. The length (l) is the maximum length of the symbol found, the offset (o) is the distance of the found symbol from the previously encoded symbol in the dictionary, and the codeword (c) is the next symbol in the LAB waiting to be encoded (Ziv and Lempel, 1977). Table 3.5 illustrates the LZ77 algorithm with window size of seventeen (17), the SB size of eight (8) and the LAB size of nine (9).

The LZ77 process begins by first entering the symbols into the LAB. The first symbol from the left is considered for the search. The LAB searches left-wise for the longest matching in the SB. If no match found, zero is entered against the offset (o) and the length (l) while the search symbol is entered as the codeword (c). Otherwise, if match is found the offset, the length and the next symbol as codeword are entered in the form <o,l,c> as output. During encoding, SB always holds and stores the encoded symbols, and discard the left most symbol anytime the window length is overflowed.

To demonstrate the algorithm modified in this research, consider 'aataaaattatatat' as the input for the encoding and decoding using both LZ77 and the modified LZ77 algorithms. Here, the window size is fourteen (14) with eight (8) and six (6) for the size of the SB and the LAB respectively.

Table 3.6: LZ77 process of symbol encoding for the input ‘aataaaattatatat’

	←search buffer→								←Look ahead buffer→						
index	8	7	6	5	4	3	2	1	1	2	3	4	5	6	Output
1									a	a	t	a	a	a	<0,0,a>
2								a	a	t	a	a	a	a	<1,1,t>
3						a	a	t	a	a	a	a	t	t	<3,2,a>
4			a	a	t	a	a	a	a	t	t	a	t	a	<5,2,t>
5	a	t	a	a	a	a	t	t	a	t	a	t	a	t	<8,3,t>
6	a	a	t	t	a	t	a	t	a	t					<2,2,*>

Table 3.6 shows the encoding process of LZ77 and the corresponding output of each encoding process. In the first encoding the dictionary of the SB is empty, as such zero (0) is recorded for both the offset and the length and ‘a’ is entered as the codeword. Thus, the output is <0,0,a>. The window slides and ‘a’ now formed the first entry of the SB. For the second encoding process, the encoder will use ‘a’ in the LAB as the search symbol to look for match in the SB. Match is found in offset 1 (1st position in the SB) with length 1 (length of the symbol) and the next symbol ‘t’ as the codeword, thus the output is <1,1,t>. The search symbol ‘a’ and the codeword ‘t’ now slides from the LAB to the SB. Thus, the SB contains ‘aat’ as the new dictionary entries. For the third encoding, the search symbols are ‘aa’, and match is found in the SB in offset 3 and the length of 2 for the 2 symbols ‘aa’ with ‘a’ as the codeword. Hence the output is <3,2,a>. The search symbol ‘aa’ and the codeword ‘a’ now slides from the LAB to the SB. Therefore, the SB contains the previous symbols ‘aat’ and the new symbols ‘aaa’ to form ‘aataaa’ as the new dictionary entries.

The fourth encoding using the search symbol ‘at’ of the LAB, found a match in offset 5, length of 2 and codeword of ‘t’ forming the output <5,2,t> . The fifth encoding found a match in offset 8, length 3 and codeword ‘t’, thus the output is <8,3,t>. The last encoding has both the offset and length of 2 with no codeword. The absence of the codeword is because there is no next symbol in the LAB to be encoded.

Decompression is the conversion of the encoded symbols to the original form. The compress data is converted by decoding the encoded symbols. The process involves no searching of any symbol.

To decode back the encoded outputs, the decoder only translates the symbols back to the original input based on the dictionary positions. The symbols are decoded by copying the positions already constructed during the encoding process. For matched symbols, the dictionary copies the whole specified number of symbols based on the offset, the length and the codeword. Table 3.7 demonstrates the decompression process of the encoded output of Table 3.6 based on LZ77.

Table 3.7: LZ77 process of symbol decoding for the encoded input ‘aataaaattatatat’

Sno	Encoded output	Decoded symbol(s)	Decoded output
1	<0,0,a>	a	a^1
2	<1,1,t>	at	$a^3\underline{a^2t^1}$
3	<3,2,a>	aaa	$a^6a^5t^4\underline{a^3a^2a^1}$
4	<5,2,t>	att	$a^9a^8t^7a^6a^5a^4\underline{a^3t^2t^1}$
5	<8,3,t>	atat	$a^{13}a^{12}t^{11}a^{10}a^9a^8a^7t^6t^5\underline{a^4t^3a^2t^1}$
6	<2,2,*>	at	a a t a a a a t t a t a t $\underline{\text{a t}}$

To decode the 1st encoded output <0,0,a>, the decoder entered only the codeword ‘a’ in positon 1 as ‘a^1’ since both the offset (position) and the length are zero. The second encoded output <1,1,t> is decoded based on the position of the entry in the first decoded output. Therefore, for position 1, implies ‘a’ in position 1 with length of 1 and the‘t’ as codeword. Hence, the second entry is ‘a’ and codeword ‘t’. Having the previous output ‘a’ with the new entry of ‘at’, this formed the decoded output ($a^3a^2t^1$).

To decode the output <3,2,t>, the decoder takes from the previous decoded output ($a^3a^2t^1$) and count the position and the length of the new entry. Here, in position 3 and

University of Malaya

length 2 is ‘a^3a^2’, with codeword ‘t’ thus forming the entry ‘aaa’. Having the previous decoded output ‘aat’ and the new entry ‘aaa’, this formed a decoded output of ‘aataaa’.

Same process is applied to decode the 4th <5,2,t> and the 5th <8,3,t> encoded outputs. The last encoded output <2,2,*>, has is no codeword, therefore only the position 2 and the length 2 are recorded as the entry to produce (a^2t^1). As seen in the last decoding process in Table 3.7, the final decoding returned back the original input ‘aataaaattatatat’ exactly as `a a t a a a a t t a t a t a t`.

3.7.2 Modified LZ77 Compression Algorithm

The compression process in the modified LZ77 is similar to the conventional LZ77. But the difference is the use of frequency (2) in the modified algorithm. In the modified algorithm, when match symbols are found in the SB during encoding, instead of considering the search symbol(s) in the LAB, the algorithm moves one time and checks the adjacent symbol(s) in the LAB for similarity with the current search symbol(s). If the adjacent symbol(s) is/are similar with the search symbol(s), then the offset and the length of the search symbol are entered, and frequency of 2 is added to the pointers' variable. The added frequency implies that the match in that position is repeated twice. Then, next symbol after the adjacent similar symbol(s) is considered as the codeword while the adjacent symbol(s) is/are ignored hence not included in the part of the SB dictionary.

The same input ‘aataaaattatatat’ was used to demonstrate the compression and decompression of the modified algorithm. The Table 3.8 shows the compression process indicating the frequency of the occurring similar symbols in the LAB,

University of Malaya

Table 3.8: Modified LZ77 process of symbol encoding for the input 'aataaaattatatat'

	← search buffer →								← look ahead buffer →							
index	8	7	6	5	4	3	2	1	1	2	3	4	5	6	**Output**	**Freq.**
1									a	a	t	a	a	a	<0,0,a>	1
2								a	a	t	a	a	a	a	<1,1,t>	1
3						a	a	t	a	a	a	a	t	t	<3,2,t>	2
4			a	a	t	a	a	t	t	a	t	a	t	a	<4,2,t>	2
5	a	t	a	a	t	t	a	t	a	t					<2,2,*>	1

In Table 3.8, all outputs show that symbols have been matched except in the 1st output; <0,0,a>, where zero (0) is recorded for both offset and length and the search symbol 'a' as the codeword. The second encoding <1,1,t>, has one (1) as the entry for respectively the offset and length with 't' as codeword.

During the third encoding process, a matched symbol 'aa' is found at offset 3 in the dictionary. Following the rule of this modified algorithm to tag frequency of 2 for similarity of search symbol and the adjacent symbol, the encoder will move one more step right wise to check for another similarity in the LAB. Here, the encoder found another 'aa' directly adjacent to the previous one. Therefore, a frequency of two (2) is entered into the dictionary as part of the pointers' parameter and ignore the second found symbols 'aa' without assigning any pointer to them. The sliding window will move over two symbols 'aa' only instead of four symbols 'aaaa', because the location of the second two symbols 'aa' is not recorded in the dictionary. The output will therefore be recorded as the symbols 'aa' appeared twice with 't' as codeword, thus represented as <2[3,2],t>.

The same applied to the fourth encoding. A matched symbol 'ta' is found at offset 4 in the dictionary with the symbols 'ta' appeared twice in the LAB. Frequency of (2) is attached to the pointer's parameter and the sliding window moves over two symbols 'at' instead of four symbols 'tata because the location of the second two symbols 'ta' is not

University of Malaya

recorded in the dictionary. The output will thereby be recorded as the symbols ‘aa’ appeared twice with ‘t’ as codeword, thus represented as <2[4,2],t>. The last encoding output <2,2,*> has both the offset and length of 2 with no codeword. This is because there is no other symbol in the LAB to be encoded.

To decode the encoded output of Table 3.8, the frequency rule is applied. Table 3.9 shows the decoding process of encoded entries in Table 3.8. In the decoding process, once frequency of 2 is found, the decoder repeats the decoding symbol(s) found at that position.

Table 3.9: Modified LZ77 process of symbol decoding for the encoded input ‘aataaaattatatat’

Sno	Encoded Output	Decoded symbol(s)	Search frequency	Decoded output
1	<0,0,a>	a	1	a^1
2	<1,1,t>	a	1	$a^3a^2t^1$
3	<3,2,t>	aa	2	$a^6a^5t^4a^3a^2$(aa)t^1
4	<4,2,t>	ta	2	$a^9a^8t^7a^6a^5$(aa)$t^4t^3a^2$(ta)t^1
5	<2,2,->	at	1	a a t a a a a t t a t a t a t

The first encoded output is <0,0,a>. Therefore, the search symbol ‘a’ is decoded with the offset and length are zero. The second encoded output <1,1,t> having both offset and length of one (1) with ‘t’ as the codeword, counting from the offset of the previous decoded output a^1, symbol ‘a’ is entered with codeword ‘t’, thus symbols ‘at’ are decoded. To decode the third encoded output <3,2,t>, the frequency (2) is utilized. The decoder reads the encoded output as <2[3,2],t> signifying that the content of offset 3 and length 2 will be outputted twice with ‘t’ as codeword. Counting the offset from the subsequent decoded output $a^3a^2t^1$, two symbols ‘aa’ with ‘t’ as the codeword. Considering the frequency (2), thus ‘aa’ is repeated and producing the decoded the output as ‘aaaat’.

University of Malaya

The same holds for the fourth output. The decoder reads the encoded output as <2[4,2],t> indicating that the content of offset 4 and length 2 will be output with ‘t’ as codeword. Counting the offset from the subsequent decoded output $a^6a^5t^4a^3a^2$(aa)t^1, the decoder will not include the symbols (aa) since is not available in the encoding history. As such only the offset of the existing symbols a^3a^2 will be referred to by the pointer, thus the output will be symbols ‘ta’ with frequency (2) and codeword ‘t’. Based on the rule, the symbols will be repeated and include the codeword at the end. The resulting output will then be ‘ta(ta)t’.

The last stage is the decoding of the encoded output <2,2,*>. The output has offset and length of 2 with no codeword. The decoder simply counts the offsets of the previous decoded output, without considering the repeated symbols (aa) and (ta). These repeated symbols have no history in the encoder because they are not being referred during encoding. Counting the positions from the previous decoding output $a^9a^8t^7a^6a^5$(aa)$t^4t^3a^2$(ta)t^1, the offset 2 and length 2 is a^2t^1 or precisely ‘at’. As it can be seen in the last decoding in Table 3.9, the final decoded output is `a a t a a a a t t a t a t a t`, which is exactly same with the original input ‘aataaaattatatat’.

The conclusion is that both the output from the LZ77 and the modified LZ77 are exactly the same. But the modified LZ77 produced the result in five (5) stage while the LZ77 produced the result in six (6) stages. This implies that the modified LZ77 use fewer pointers to execute the same input. Fewer pointers in the encoding process, implies more symbols can be accommodated in the SB’s dictionary. Table 3.10 shows the pseudocode of the modified LZ77 compression algorithm.

University of Malaya

Table 3.10: Pseudocode of the Modified LZ77 Compression Algorithm

```
1:  begin
2:    get input from server
3:    while (input is not empty) do
4:     begin
5:       locate the longest prefix r of entry starting in coded part
6:       locate the search symbol in non-coded part
7:       check the adjacent symbol after search symbol for similarity
8:       if no similarity then goto 11
9:           if similarity found
10:            then record the frequency (2) of occurring symbols
11:            l:= position of r in window
12:            m:= length of r
13:            n:= first symbol after r in the input entry
14:            output 2[(l,m,n)] as encoded   goto  20
15:          endif
16:       l:= position of r in window
17:       m:= length of r
18:       n:= first symbol after r in the input entry
19:       output 2[(l,m,n)] as encoded
20:       move next symbol
21:      endif
22:   endwhile
23:  end
```

3.8 System Implementation and Execution

The Web services were implemented using Oracle WebLogic Server in J2EE environment. The implementation used a virtual client-server scenario to run and execute the request-response service. The payloads are in normal and compressed formats.

Each message format in both web services was executed 50 times and metrics were monitored, measured and recorded. In order to effectively capture the desired metrics: response time, overhead and payloads. The average of the executions is captured and samples are shown in Appendices E1-F2. These metrics at the server and client endpoints include payload generation overhead, payload size, server response

time, client response time, compression overhead, decompression overhead and transaction response time.

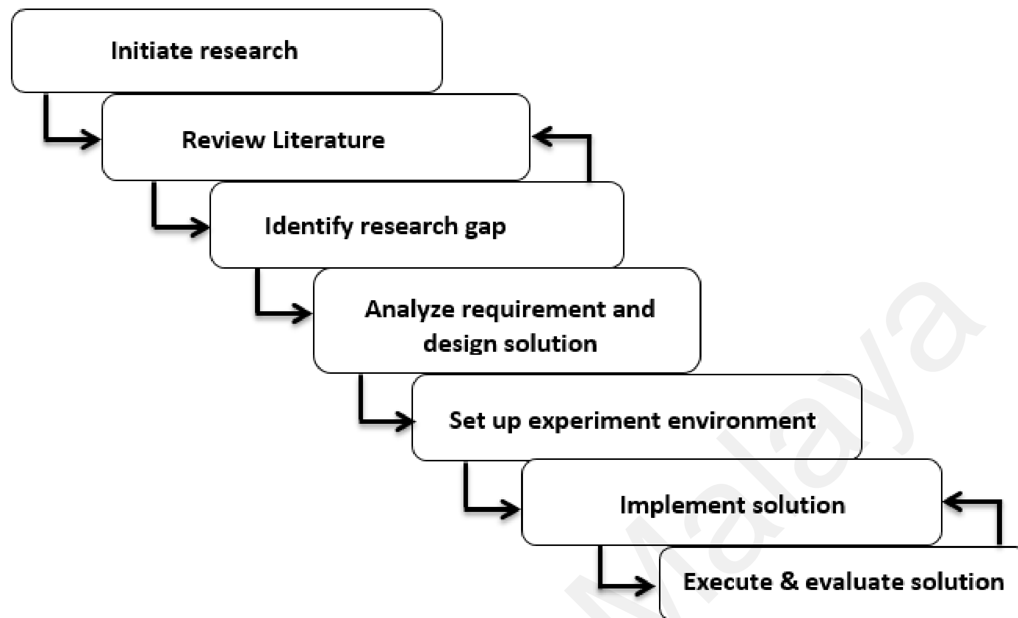


Figure 3.3: The flow of the research methodology process for the SOAP performance enhancement for large volume messaging

3.9 Summary

This Chapter explains how the research was conducted. It gives details of how the research was initiated by identifying the research area and the research gap. The process of identifying the system requirement and the design of the solution is explained in this Chapter. The setting of the experiments and the implementation and execution of the implementation were also explained.

The system requirements to perform the experiment of the research were carefully identified. SOAP over HTTP and SOAP over JMS Web services were appropriately set up for the experiment. Each system comprises of the server and the client web services for the exchange of messages in two different formats: normal message and compressed message. While normal message was executed without constraint, the compressed version uses a modified LZ77 compression algorithm. Both web services were

University of Malaya

implemented and executed 50 times and at every phase, metrics were monitored and recorded. The metrics formed the results of this study. The results of the experimental metrics calculations and the plot of the graphs are presented in Chapter 5.

University of Malaya

CHAPTER 4: SYSTEM REQUIREMENT ANALYSIS, DESIGN AND IMPLEMENTATION

This chapter presents how the research problem under study was analysed and designed. To effectively obtain useful experimental results, the chapter carefully provided the solution through identification of the system requirements. The solution was obtained by analysing and designing the requirements based on the research objectives. Unified Modelling Language (UML) 2.0 was used to model the requirement analysis and the design for the system. The following representations of the UML; use case, class diagram, activity diagram, sequence diagram and component diagram were utilized in the design.

4.1 System Analysis

The purpose of meeting the research objectives was analyzed here in order to identify the requirements and to model a system that will answer the research questions. The problem is studied and analyzed to understand and identify the component to be used in the implementation. UML Use case, Class diagram and Sequence diagram were used for the analysis.

4.1.1 Use Case Diagram

The use case depicted the functional reality of how the solution will work between two or more applications or machines in the web services transaction. It represents the functional requirement of the Web services showing how graphically the services can be requested and be consumed. Figure 4.1 shows the transaction between service provider and the service requester in the Web services. The figure is a graphical generalization for modelling Web service with any transport protocol.

University of Malaya

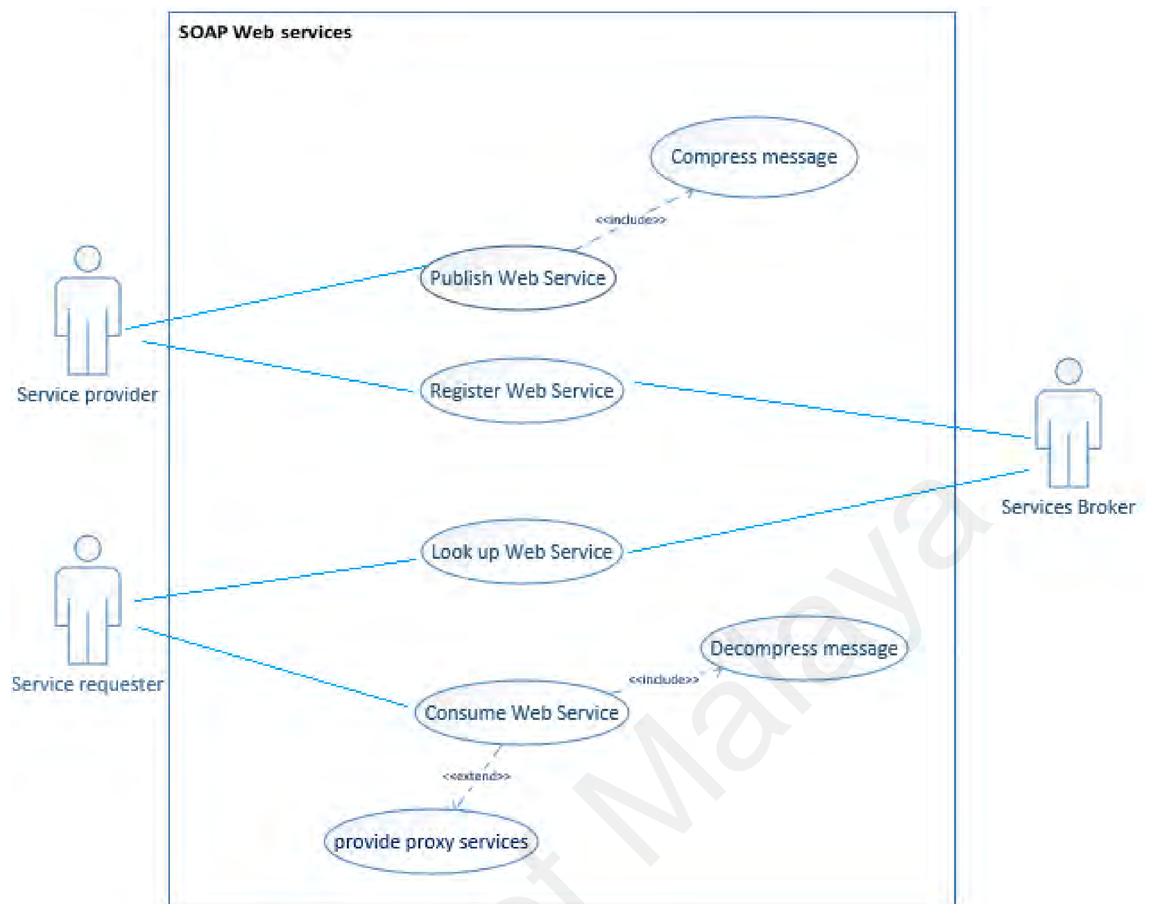


Figure 4.1: Use case diagram for the SOAP performance Web services enhancement showing the detailed use cases and the actors of the Web services

The use case diagram consists of three (3) actors and seven (7) use cases. The model uses publish-subscribe pattern to provide and consume the services. The *service provider* actor publishes the service by including the compress message use cases. The provider web service includes the *compress message* use case to compress the message if the message needs to be compressed. The *service requester* then subscribes via the *consumer web service* use case to consume the provided service by the *service provider* by including the *decompress message* use case to decompress the message provided by the *service provider*.

University of Malaya

4.1.2 Activity Diagram

The activity diagram describes the dynamic workflow of the Web service. It shows the interaction between the provider and the consumer web services. This captured the initiation of the request to the stage when the service will be provided and consumed. Figure 4.2 illustrates the activities in the Web services indicating the activity flow in the system. The figure is a generalization diagram for both HTTP and JMS web services.

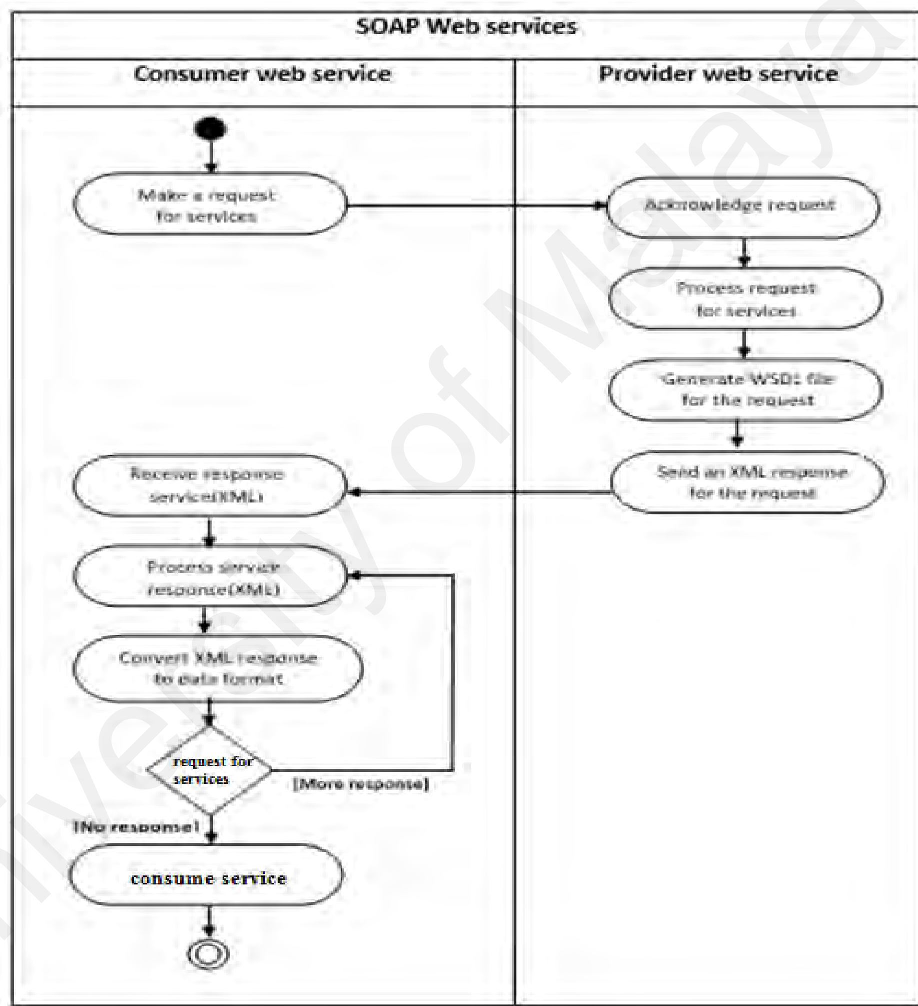


Figure 4.2: Activity diagram for the SOAP performance Web services enhancement showing the dynamic flow of activities of the Web services.

University of Malaya

4.1.3 Sequence Diagram

A sequence diagram is utilized to model the dynamic behaviour of the objects in the solution of the Web services. Here, the sequence diagram models the sequential flow of

objects participating in the Web services. It depicts how messages are exchanged in the Web services communication over time.

Figure 4.3 shows the chronological procession of the objects from the requesting of the service to its consumption.

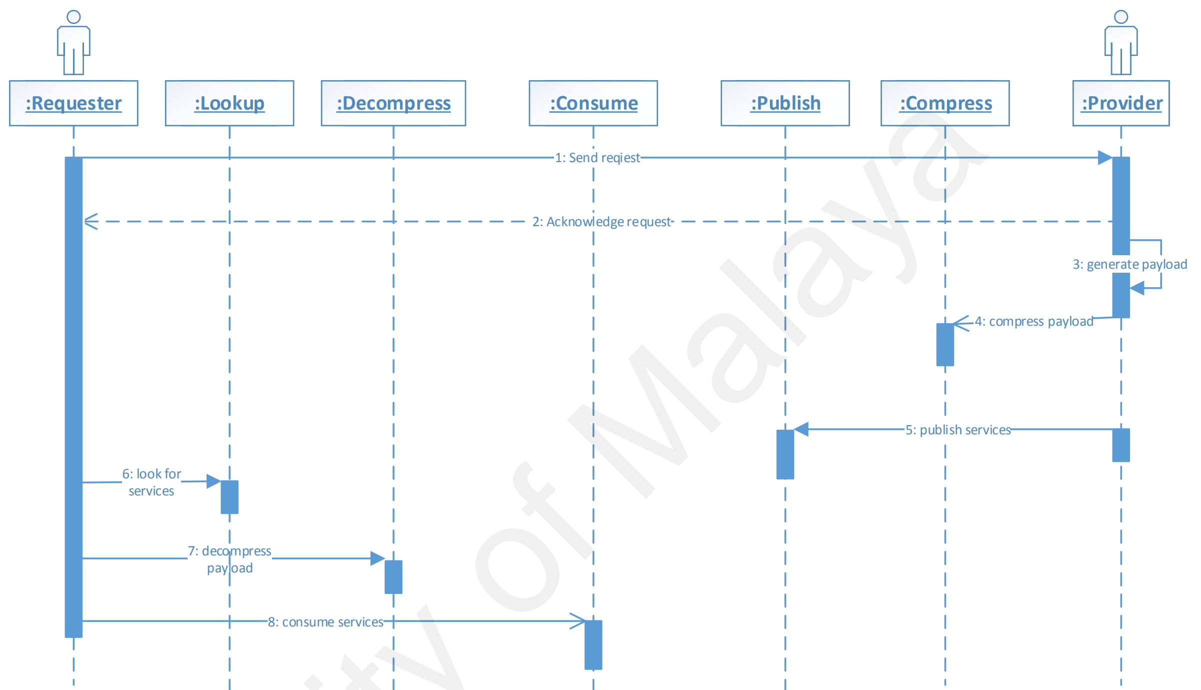


Figure 4.3: Sequence diagram for the SOAP performance Web services enhancement showing the objects involved in the request-response implementation

The requester makes a request for service and the provider responds by acknowledging the request. The provider generates a message (payload) and compresses the message (if there is a need to be compress). The provider then publishes the compressed payload for consumption. The requester looks up for the services, if available the requester decompresses the payload and finally consumes the service.

4.2 System Design

The system design deals with the creation of a solution from the requirements analysis. It provides a static solution to the stated problem under study. It defines the modules and the components needed for meeting the research objectives. Based on the system analysis, the system is designed by defining all the elements involved in the analysis and converting them to define the solution to the research problem. UML class diagram and component diagram were used in this research for the solution design.

4.2.1 Class Diagram

The class diagram shows the stationary model view of the solution. It describes how the classes and their elements are arranged to work in the entire Web services experiment. As shown in Figure 4.4 and Figure 4.5, the classes, methods, operations and constraints were related and modelled to capture and describe the responsibility of the system. However, this diagram did not model the solution of the transport aspect. Figure 4.4 is the bench mark design consisting of controller class, web service provider class, web service client class and their corresponding compressor and decompressor classes. This model is designed to be transported via HTTP. Figure 4.5 is the prototype comprising of controller class, web service provider class, web service client class and their corresponding compressor and decompressor classes and additional class for calling other functions of the prototype. This prototype model is designed to be transported via JMS.

University of Malaya

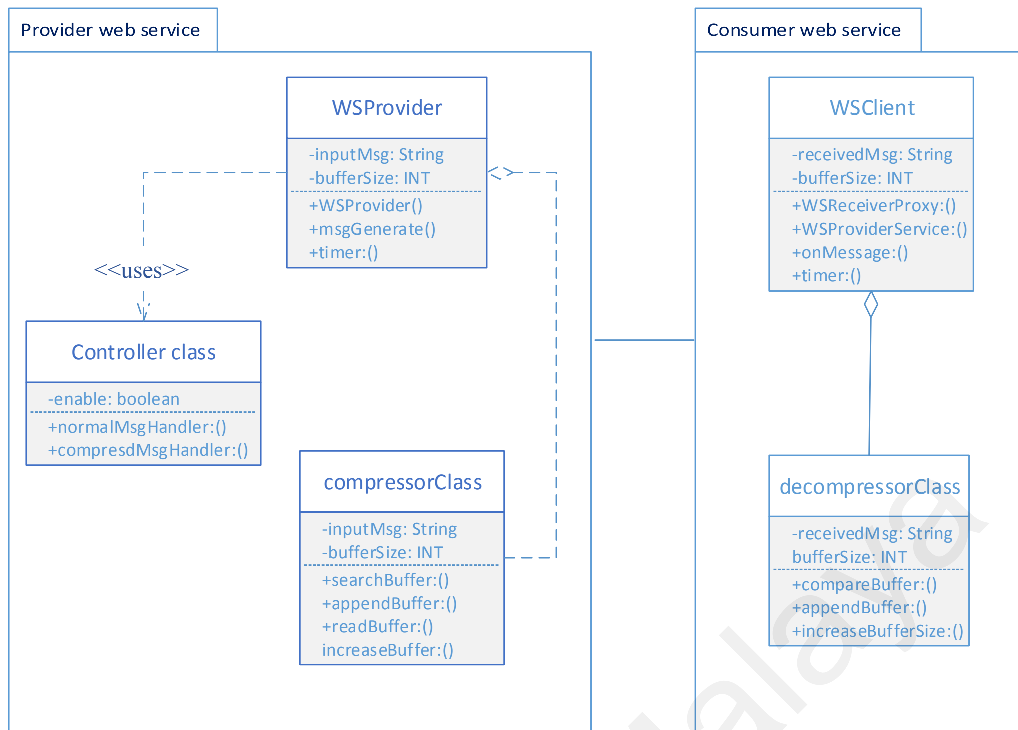


Figure 4.4: Class diagram for the SOAP performance Web services enhancement showing the classes involved in the HTTP web services implementation

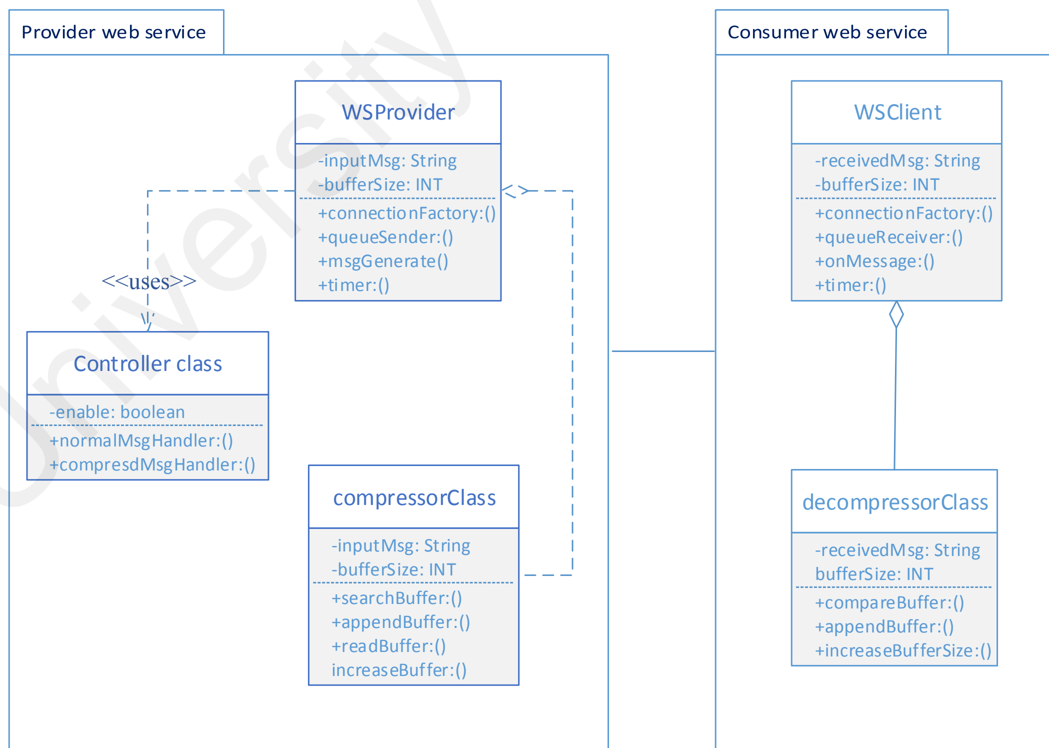


Figure 4.5: Class diagram for the SOAP performance Web services enhancement showing the classes involved in the JMS web services implementation

University of Malaya

4.2.2 System Components

Figure 4.6 and Figure 4.7 show the major component diagrams for the Web services. The diagrams show how the entire provider/consumer services is modularized for effective control and ease of reuse. Figure 4.6 depicts the component diagram for SOAP web service over HTTP transport while Figure 4.7 depicts the component diagram for SOAP web service over JMS transport.

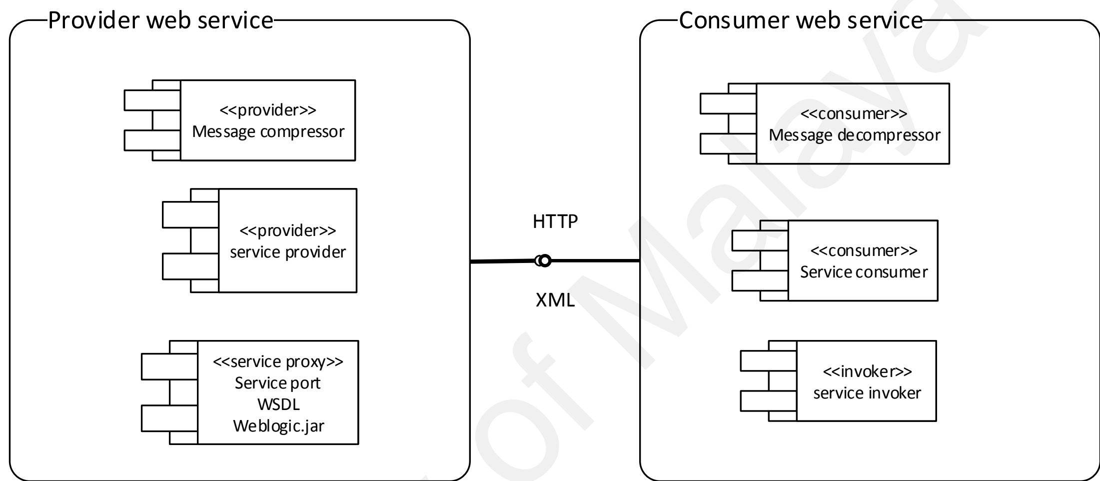


Figure 4.6: Component diagram for the SOAP performance Web services enhancement showing the major components for achieving the functionalities of the implementation of HTTP web services

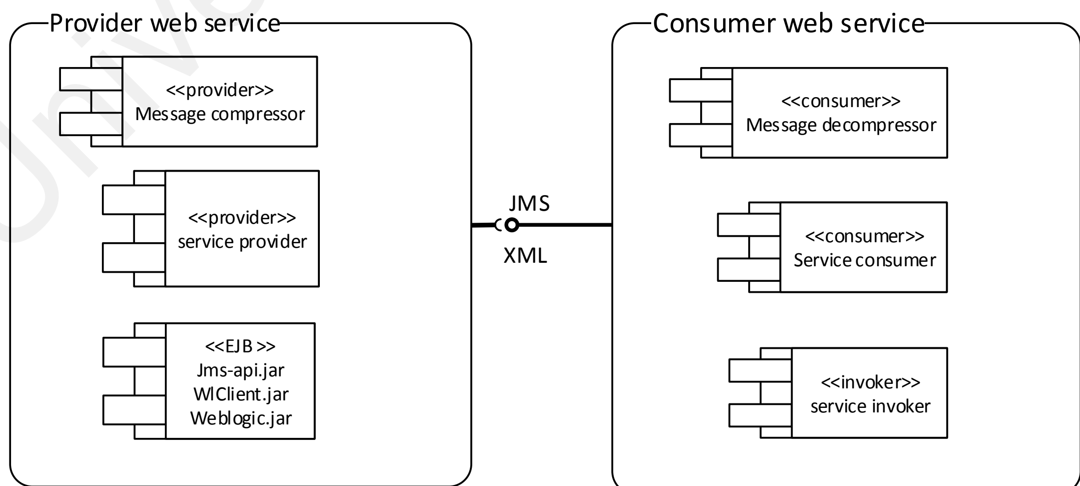


Figure 4.7: Component diagram for the SOAP performance Web services enhancement showing the major components for achieving the functionalities of the implementation of JMS web services

University of Malaya

4.3 System Implementation

The implementation is the construction of the system for operational purpose. This stage is a by-product of the system analysis stage. The system is built and put to use for evaluation. J2EE (Kalin, 2013) was used to code the solution and the client-server used a WebLogic server (Saab, Coulibaly, Haddad, Melliti, Moreaux, & Rampacek, 2012) to handle the message exchange.

4.3.1 Web Services Implementation

Both experiments were conducted using top-down approach. The implementation hardware and software environment are mentioned in Sections 3.6.1 and 3.6.2 respectively. In each exchange protocol, server and client web services were created and their classes and methods were annotated to be bounded at run time during the transaction.

In the SOAP over HTTP web services, the experiment was executed in order to obtain the WSDL and extract the endpoint (the client-side linker). The WSDL is then used to expose the web services to be available at the client side.

Unlike the HTTP version, the JMS version uses its connection properties to define the sender, receiver, connections and sessions once. The wsimport annotation of the SOAP over JMS web services creates the endpoint, bindings and the WSDL to be used in the service consummation.

The development phase of the implementation in both web services was ran and tested as the implementation progressed. Overall testing was achieved through identifying logical and run time errors and then debugged. The final tested experiment was then put to use for the analysis. Figure 4.8 shows the architecture of the web

University of Malaya

services from adopted Oracle web services (Brydon and Singh, 2010) and modified to suit the purpose of this research.

The experiment was performed using the Oracle WebLogic server as the run time server. Data were generated by the message generator method in the WSProvider class at the server side upon receiving a request from a client. Metrics such as the message (payload), message size, response time and other overhead were captured and recorded at both sides. Gradually, message size was increased and the metrics were consequently monitored, measured and recorded. Figure 4.8 illustrates the flow of the process of executing the two web services.

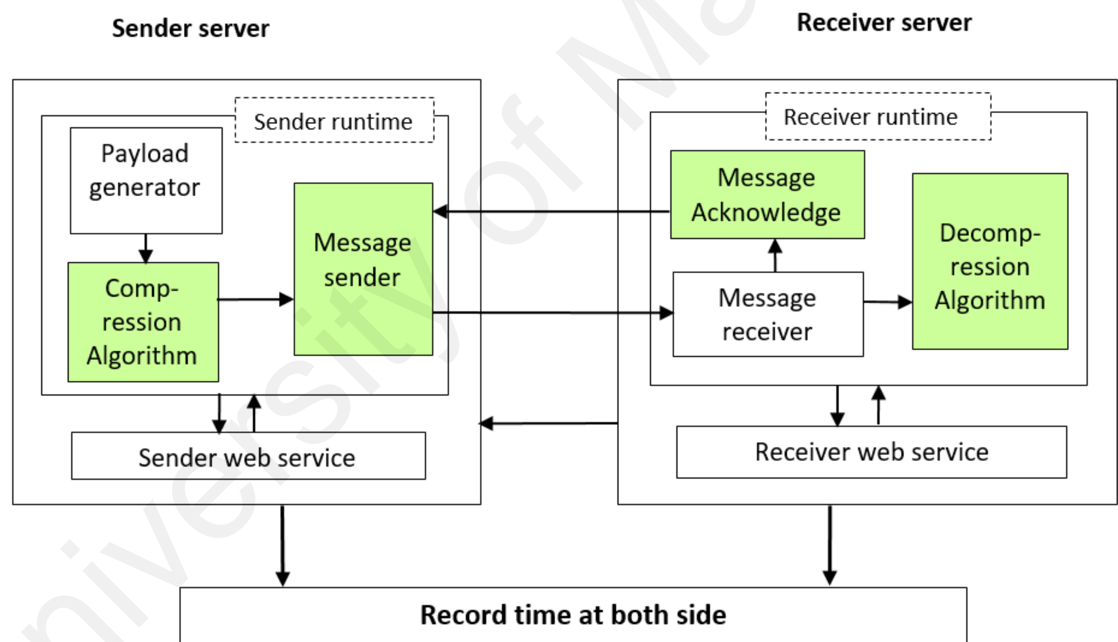


Figure 4.8: SOAP messaging architecture for large volume with modified LZ77 compression algorithm (Brydon and Singh, 2010)

University of Malaya

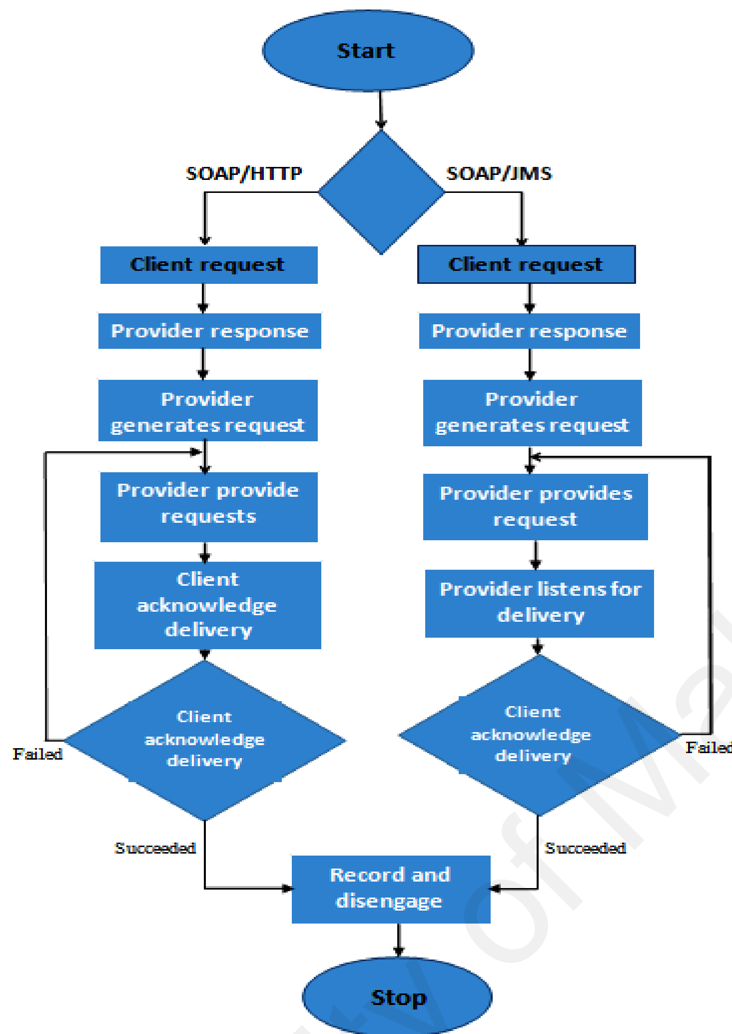


Figure 4.9: Flowchart for the implementation of SOAP/HTTP and SOAP/JMS

4.3.2 System Execution and Evaluation

The exchange of XML input is performed for the two web services the: SOAP over HTTP and SOAP over JMS. Each transaction is executed 50 times. The client requests for services from the server and the server provides the services by generating the payload. The server exchanges the generated payload via the exchange protocol (HTTP or JMS). Figure 4.10 shows how individual metric was captured during the exchange between the server and the client.

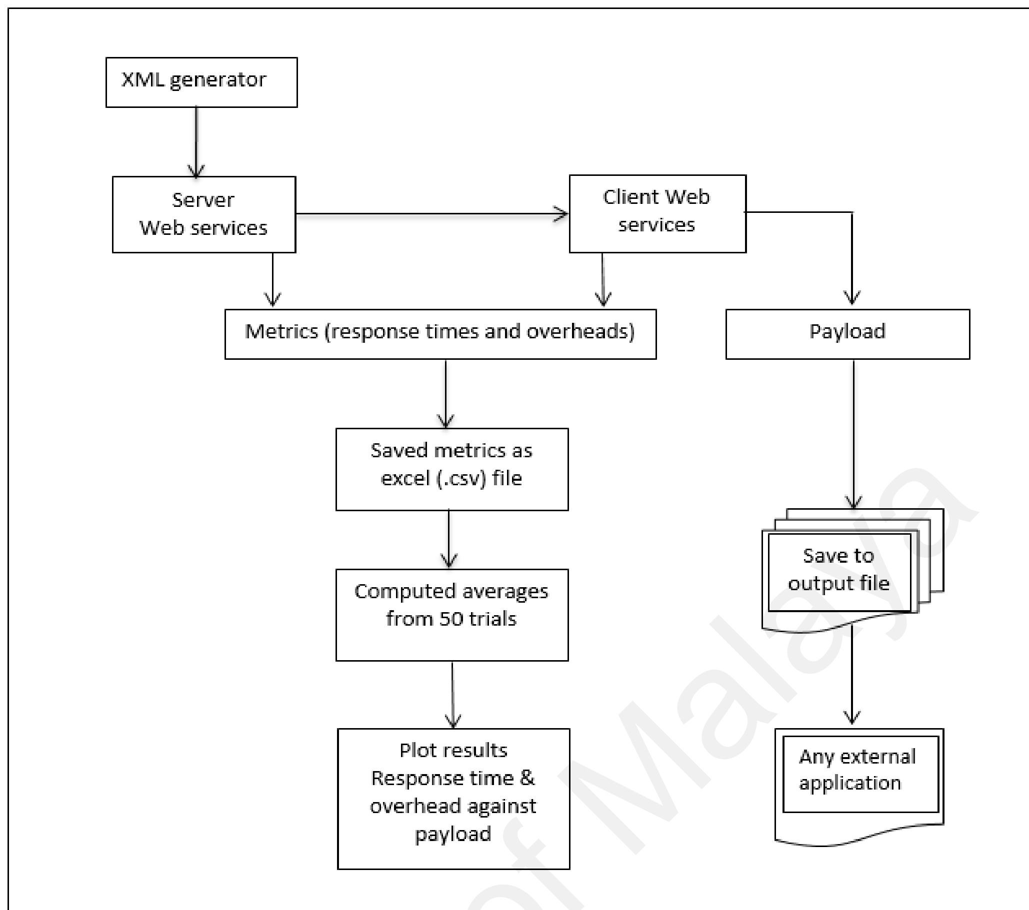


Figure 4.10: Workflow for the Web services showing how the endpoints interact and how the metrics for the services were captured for the analysis

The metrics: server response time, client response time, payload overhead, compression overhead and decompression overhead were measured and recorded for every transaction. The metrics were automatically sent by the application and saved as excel (.csv) file for each transaction, while the payloads for all transactions were appended at every transaction and saved in notepad file which can be accessed for further use. After running, measuring and recording the Web services for 50 times, the excel data were collected into a single file with 10 sheets each containing 5 transactions. The average for each metric except the payload was calculated using an excel formula for average.

The average for the metrics were calculated by referring to the results in each of the 10 excel sheets. For instance, to calculate the average for a particular metric is;

=AVERAGE(Sheet1!B2, G2, H2, K2, O2, Sheet2!B2, G2, H2, K2, O2, Sheet3!B2, G2, H2, K2, O2, Sheet4!B2, G2, H2, K2, O2, Sheet5!B2, G2, H2, K2, O2, Sheet6!B2, G2, H2, K2, O2, Sheet7!B2, G2, H2, K2, O2, Sheet8!B2, G2, H2, K2, O2, Sheet9!B2, G2, H2, K2, O2, Sheet10!B2, G2, H2, K2, O2).

The result will produce average of all the values of that particular metric contained in the 10 sheets.

Table 4.1: Performance metrics calculations shows the formulae used for calculating the metrics and subsection 4.3.2.1 explains the metrics and how they are calculated. Samples raw data of the metrics for both protocols (HTTP and JMS) are shown in appendices E1-F2. The averages for the results are presented in Tables 5.1 – 5.7 in Chapter 5. The results were copied and paste into a software (Origin lab, version 6.4) for plotting engineering and scientific graphs. The graphs for the response times and overheads were plotted against payloads for all the transactions in both SOAP over HTTP and SOAP over JMS. The graphs are presented in Figures 5.1 – 5.7 in Chapter 5.

Table 4.1: Performance metrics calculations

Long name	Formula
Payload overhead	$T_{pgt} = t_{pgt1} - t_{pgt0}$
Server response time	$T_{srt} = t_{srt1} - t_{srt0}$
Client response time	$T_{crt} = t_{crt1} - t_{crt0}$
Compression overhead	$T_c = t_{c1} - t_{c0}$
Decompression overhead	$T_{dc} = t_{dc1} - t_{dc0}$
Transaction response time	$T_{rt} = \sum T_{crt} + \sum T_{srt}$

4.3.2.1 The Web Services calculation for response time and overheads

a) **Payload generation time:** This is the time taken for the sender to generate the payload. The payload is generated as a message and send to the client each time the service is requested by the client. The payload generation time is calculated

automatically at the server as the difference between the time when the server application triggers the module to generate the message and the time when the message is generated. The payload is considered as overhead and is isolated from the actual exchange time.

This is calculated as payload generation time = payload generation stop time - payload generation start time and measured in milliseconds (ms) and is given as:

$$T_{pgt} = t_{pgt1} - t_{pgt0}$$

b) **Compression time:** This is the time taken by the server to compress the generated message before sending to the client. The payload compression time is calculated automatically in milliseconds (ms), at the server as the difference between the time when the server starts compressing the generated payload and the time when the server finishes compressing the generated payload. This is calculated as payload compression time = payload compression stop time - payload compression start time. and given as:

$$T_c = t_{c0} - t_{c1}$$

c) **Server response time**: Time taken by the server to successfully exchange the payload. The server generates the payload and sends in normal or compressed format as the case may be, through the Web services. The send time is the time at the server to communicate with other server application services, process and send the payload. Therefore, the send time is the total time taken at the server to provide the required services. The send time is comprised partly the payload generation time and the compression time (for compressed version), which are regarded as overhead.

Server response time is automatically calculated at the server as the time difference when the server receives a request from the client and the time when the server provides the request to the client. This is precisely the start and the

University of Malaya

stop of the server for every exchange, measured in millisecond (ms) and given as:

$$T_s = t_{srt1} - t_{srt0}$$

d) **Decompression time**: Time taken by the client to decompress the received compressed payload. The payload decompression time is calculated automatically in milliseconds (ms), at the client side by the client application. Decompression time is the difference between the time when the client starts decompressing the compressed payload and the time when the client finishes decompressing the compressed payload. This is calculated as payload decompression time = payload decompression stop time - payload decompression start time. and given as:

$$T_{dc} = t_{dc1} - t_{dc0}$$

e) **Client response time**: This is the time utilized by the client to receive and process the sent payload. This may include the decompress time (in case of compression). Receive time is automatically calculated at the client side by isolating the overhead. Receive time is an absolute time after the payload received is decompressed. This is calculated as receive time = client side stop time – client side start time and is given as:

$T_r = (t_{crt0} - t_{crt1})$ for normal payload

$T_r = (t_{crt0} - t_{crt1}) - T_{dc}$ for compressed payload,

where T_{dc} is the decompression time.

f) **Transaction response time**: Is the total time taken at the server and the client processes. This response time measure the request-response elapsed by both endpoints to request, process and provide the required services. Transaction response time or total response time is obtained as the time at the client side and

University of Malaya

the time taken at the server side to produce the desired web service result. The transaction response time involves server response time, client response time, the compression overhead and decompression overhead.

Transaction response time = time taken by the server to process and produce services + time taken at the client to receive and process the services. The metric is measured in milliseconds (ms). The response time is calculated as:

$$T_{rt} = \sum T_{crt} + \sum T_{srt}$$

$$T_{rt} = (T_s + T_c) + (T_r + T_{dc})$$

4.4 Summary

This chapter discusses the research problem requirement analysis, solution design for the system and the implementation of the problem solution. Use case, sequence diagram and activity diagram were used to model the requirements for the Web service. Class diagram and Component diagram were used to design the logical solution to the Web service. The implementation put the system into work by building a running Web service. Finally, the web services were executed 50 times to evaluate the effectiveness. Metrics for determining and fulfilling the research objectives were fetched and utilized to form the result of this study. The results of the metrics calculations and the graphs for the response times and the payloads are presented in Chapter 5.

University of Malaya

CHAPTER 5: EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Introduction

The experimental results for each Web services protocol are presented and discussed in this chapter. This chapter discusses the results of both implementations using the two different message formats: normal and compressed. The results are presented in graph and table form. The purpose of this study was examined by designing and implementing a SOAP over HTTP and SOAP over JMS web services and incorporating a modified LZ77 text compression algorithm for payload exchange.

The presentation of the findings is categorized as response times for normal (uncompressed) and compressed payloads for the benchmark system (SOAP over HTTP) and the experimental system (SOAP over JMS). The performance comparison between the two systems is vis-à-vis presented and discussed here.

5.2 SOAP over HTTP Protocol

The SOAP web services with HTTP binding protocol experimental results are shown in Table 5.1 and Table 5.2. The graphs of the experimental results are shown in Figure 5.1 and Figure 5.2. The following subsections 5.2.1 and 5.2.2 discuss the findings of the results for SOAP over HTTP binding.

5.2.1 Normal Payload Response Time

Table 5.1 shows the response time for exchange of message with normal payload using the SOAP over HTTP protocol. Figure 5.1 shows the graph of the payload generation overhead and response time for server, client and the transaction for the normal payload. As shown in the table, the transaction response time has the highest response time of 1841ms for the highest payload of 22.2MB.

The transaction and the client response times between payload 1.3MB and 9.9MB increase with the increment of the payload. However, as the payload increased to

11.9MB, the client response time decreased to 524.67ms, but then picked up again to 760.33ms when the payload was 14.2MB. Again both the transaction and the client respose times fall to 796.67ms and 637ms respectively when the payload was increased to 16.6MB. Then, the time picked up again for the last two points.

Table 5.1: Response time for normal payload (SOAP over HTTP)

Payload size (bytes)	Payload overhead time (ms)	Server response time (ms)	Client response time (ms)	Transaction response time (ms)
1339772	2.00	6.67	176.00	184.67
2153205	3.00	9.33	172.33	184.67
3158034	5.00	19.00	209.00	233.00
4354259	13.67	38.33	252.33	309.33
5741880	15.67	39.33	299.33	354.33
7320897	12.67	49.00	355.67	417.33
9896768	16.00	72.67	531.33	620.00
11938326	21.00	98.33	524.67	644.00
14171279	24.33	141.00	760.33	925.67
16595628	15.00	144.67	637.00	796.67
19211374	15.67	473.67	1062.00	1551.33
22249785	20.67	610.67	1209.67	1841.00

University of Malaya

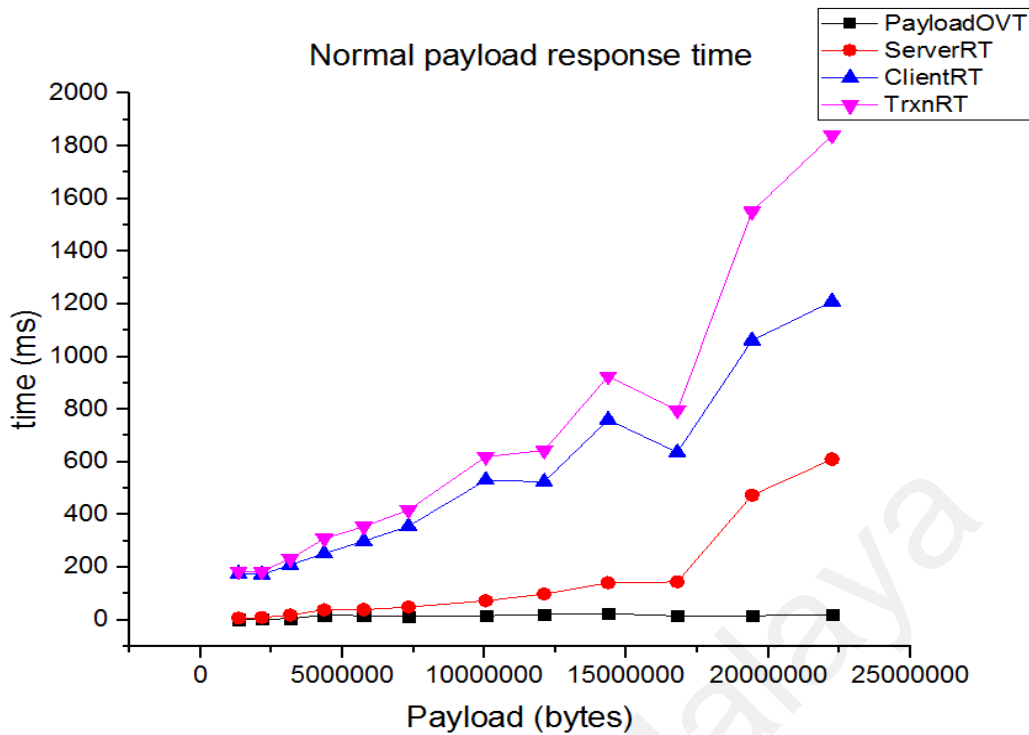


Figure 5.1: SOAP over HTTP response time for normal payload transaction

In contrast to the client response time, the server response time was steady and maintained a low response time with the increasing payload as indicated in the server response time trend in Figure 5.1. The time to respond for payloads between 1.3MB to 16.6MB were 150ms, but when the payload size was increased to 19.2MB, the response time shot up to 473.67ms, almost 329ms increment from the previous response time of 144.67ms. The server response time increased to 610.67ms as the payload was increased to 22.2 MB.

From the startup of the transaction, the payload overhead time is increasing in accordance with payload, but when the payload was 7.3MB, the overhead time decreases to 12.67ms, but then the trend went 21.00ms when the payload is increased to 11.9MB. Again, the overhead decreases to 15.00ms when the payload was 16.6MB and continued to slighlty go down before shooting up again the end of the process. The trend in the payload overhead is sporadic and produces peaks in the transition.

University of Malaya

Key findings:

The client response time, in comparison with the server response time revealed to be high due to HTTP request. The client as the requesting service always repeatedly checks the server for any interim message. As a result, the client monopolizes the transaction thread.

The transaction response time indicates to be ascending with the payload as seen in the pattern in Figure 5.1. This is possible due to the client response time that dominated the entire web service transaction.

The server response time tends to increase gradually with the increase in the payload but rises as the payload increases to 19.2MB. The possibility is that, the payload of 19.2MB might be regarded as huge at that level to the CPU process, therefore more space is needed to continue processing more incoming messages. Peaks were noticed in the payload overhead time. This surge might be due to memory swapping by the JVM to allow space for an incoming message (Nakamoto and Akiyama, 2015).

5.2.2 Compressed Payload Response Time

Table 5.2 shows the response time for exchange of message with compressed payload using SOAP over HTTP protocol.

University of Malaya

Table 5.2: Response time for compressed payload (SOAP over HTTP)

Payload (bytes)	Payload overhead time (ms)	Server response time (ms)	Client response time (ms)	Compre-ssion time (ms)	Decompre-ssion time (ms)	Transaction response time (ms)
6203	2.00	49.33	70.67	14.67	7.33	144.00
9802	3.33	61.33	86.00	16.00	10.33	177.00
14248	8.33	79.00	93.67	23.33	12.33	216.67
19552	8.00	112.00	140.00	31.67	21.33	313.00
25676	10.00	163.00	191.67	41.67	22.33	428.67
32642	17.33	236.00	298.00	55.00	59.00	665.33
42582	18.00	365.67	418.33	72.33	46.00	920.33
51448	26.00	482.33	544.67	89.33	54.67	1197.00
61175	20.33	747.67	731.00	104.00	62.33	1665.33
71737	27.00	963.67	1387.33	121.00	292.00	2791.00
83144	30.00	1276.33	2186.33	235.33	359.00	4087.00
95394	35.67	1457.00	2656.00	160.67	89.00	4398.33

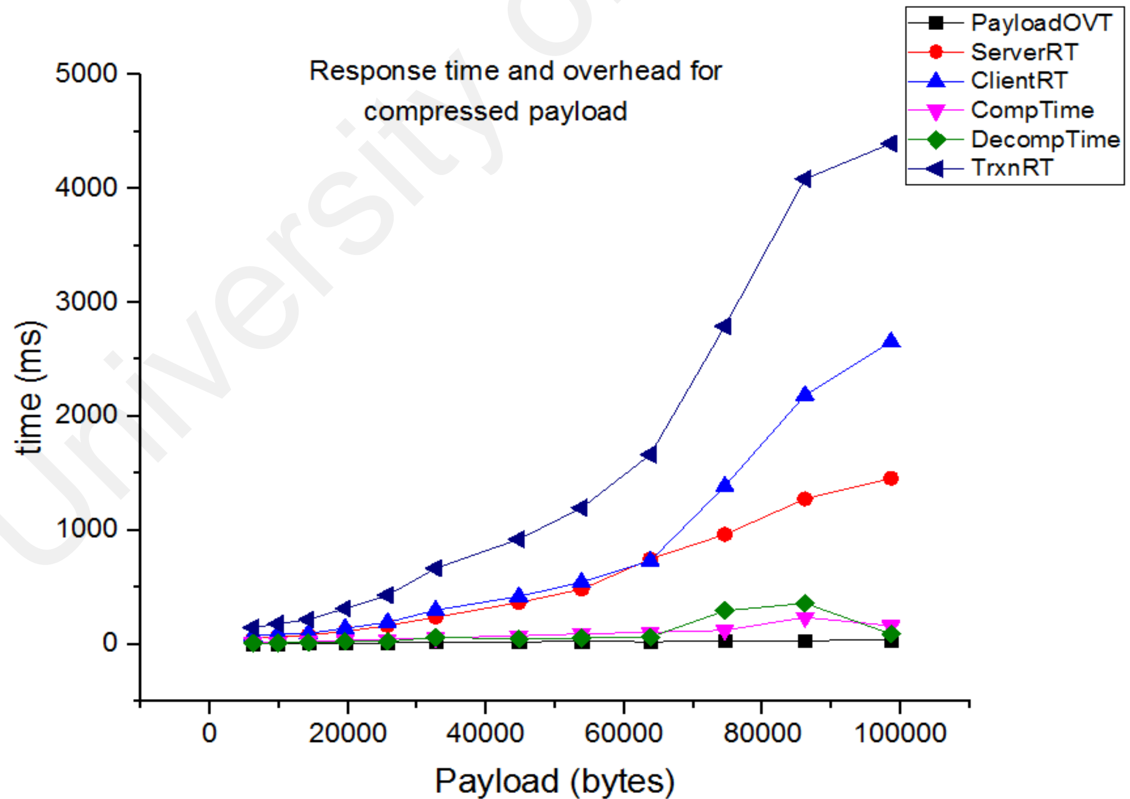


Figure 5.2: SOAP over HTTP server, client, compression, decompression and payload overhead for compressed payload transaction response times

University of Malaya

Figure 5.2 shows the payload overhead, compression, decompression delay and response times for the transaction, the server and client. From the figure, it can be seen that the payload generation time for the web services was progressing with payload increase. It moved almost smoothly parallel to the payload X-axis, except that when the payload was 61.2KB, the overhead made a downrise to 20ms.

From the initial stage of the compression overhead time till the payload of 7.2KB, the overhead increased accordingly with the payload. But then the overhead time went up when the payload was 8.3KB and then come down at the final stage to maintain linear transition. From the start, the decompression overhead time increased with the increase in the payload but abruptly ascended when payloads were 7.2KB and 8.3KB respectively.

The server and client response times both inclined to almost upward trend from the beginning of the Web services start up until the payload of 6.12KB. Both response times surged with client time rising significantly to 1387.0ms while server time increased to 963ms at payload of 7.2KB. Client response time continued to rise but the server time grew modestly with the payload increase.

The transaction response tim e for the Web services is higher than all the response times as seen in the figure. This transaction started and carried on with gradual upward bearing throughout the transaction.

Key findings:

In this transaction, the client and server response time trend together exponentially until the payload was 6.4 KB. As seen in Figure 5.2, there is a sudden drift by the client. The possible cause of this might be due memory management such as paging or memory swapping by the client JVM.

University of Malaya

Compression and decompression overheads were shown to be low as they incur little computation overhead of the total execution. In addition, compression is costly than the decompression due to a large number of comparisons in finding match patterns during encryption process in the LZ77 algorithm as mentioned by Kreft and Navarro (2010) and Mouli and Rajendra (2012).

Payload overhead time ascending slowly throughout the transaction is an indication that the overhead increases with the payload size. The payload generator is a method that is always called to produce same string of XML, as such less execution time is needed to commit the process.

It is noticed that in all the three overhead times, there is increase in one or two transaction(s), this might be attributed to memory paging or resource sharing by the CPU internal processes (H. Li et al., 2013).

5.3 SOAP over JMS Protocol

The SOAP Web services with JMS binding experimental results are shown in Tables 5.3 – 5.5. The graphs of the experimental results are shown in Figures 5.3 – 5.5.

JMS, unlike HTTP, is not a request-based protocol. It is an API with an abstraction of some interfaces and classes required by a client to communicate with the server in the messaging services. In JMS, server exposes shared services to the remote service client for consumption. In essence, the server writes the message to the queue and the client reads the data from the queue in a stateful mode. The following subsections 5.3.1 and 5.3.2 discuss the findings of the results for SOAP over JMS binding.

5.3.1 Normal Payload Response Time

Table 5.3 shows the response time for exchange of message with normal payload using SOAP over JMS protocol. Figure 5.3 shows payload overhead and response

University of Malaya

times of server, client and transaction for normal payload. As shown in the figure, the server response time revealed to be the lowest time in the web services communication. It started and continued upright with payload increase. Although there is a slight rise in the trend when the payload was increased to 19.2MB but it went down and maitained the movement.

Table 5.3: Response time for normal payload (SOAP over JMS)

Payload Size(bytes)	Payload overhead time(ms)	Server response time(ms)	Client response time(ms)	Transaction response time(ms)
1339772	25.75	4.00	46.86	76.61
2153205	47.75	8.86	71.29	127.89
3158034	74.50	3.00	77.86	155.36
4354259	91.38	12.43	128.57	232.38
5741880	107.63	8.14	131.71	247.48
7320897	165.75	13.14	223.71	402.61
9896768	260.88	10.86	233.43	505.16
11938326	387.63	8.71	269.43	665.77
14171279	402.63	11.00	299.71	713.34
16595628	493.63	11.57	341.71	846.91
19211374	594.63	44.71	388.71	997.05
22249785	763.63	9.14	453.29	1261.05

University of Malaya

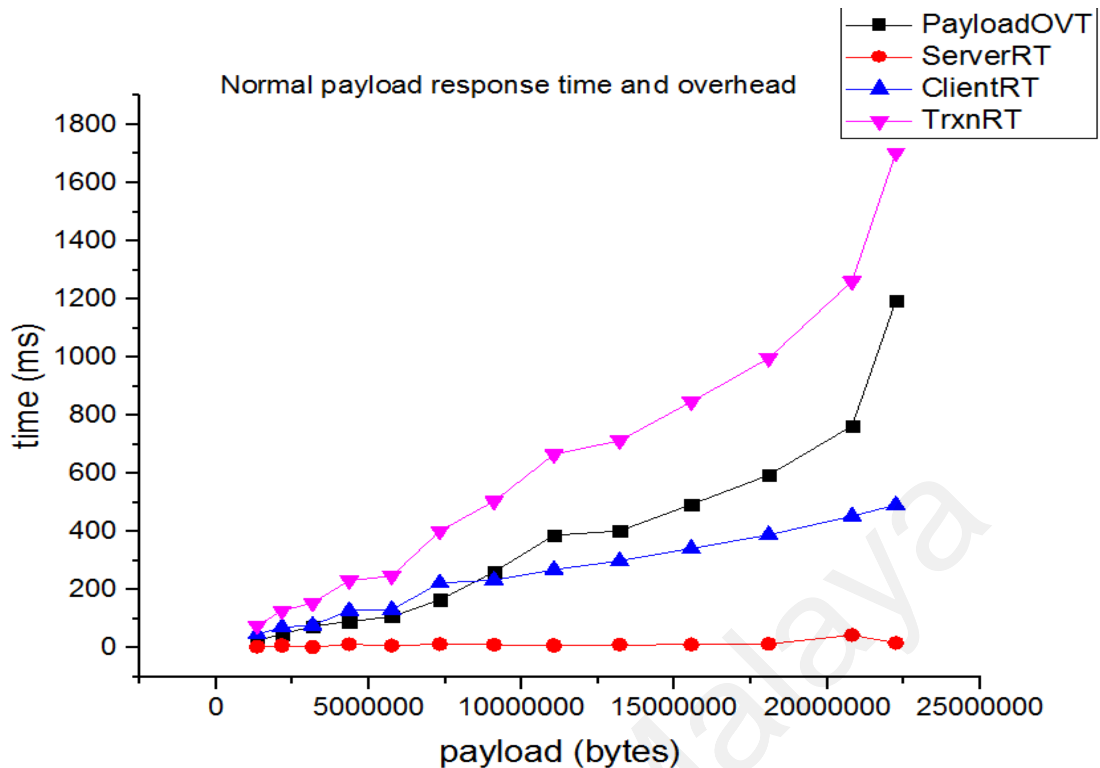


Figure 5.3: SOAP over JMS response time for normal payload transaction comprising payload generation overhead time, server, client and overall transaction response times

The transaction response time is revealed to be high as the payload increases as seen in Figure 5.3. This is possible due to the payload overhead time that dominated the entire web service transaction. The transaction response time varies accordingly at the different payloads until the load was 11.9MB and caused the trend to rise slightly higher at 665.8ms. It then maintained the bearing linearly, but this pattern changed when the payload was 22.2MB when the trend shot up from 997.1ms to 1261.1ms.

The payload overhead time generally rises with the increment in the payload till the end of the transaction. The overhead moves normal with the payload but have an upward change when the payload was increased to 11.9MB and 22.2MB. Both the transaction response time and the payload overhead were revealed to be high compared to other metrics of the transaction.

This client response time trend indicates upward transition with the payload. The response time continue to grow as the payload is increase. The transition formed a perpendicular linear progression up to the end of the system transaction. Except in one point of a payload of 7.3MB, the response time made a sudden slight change upward to 223ms but came down to maintain the course.

But for the server, the response time shows a linear progression as the payload is increased. Here payloads are produced and queued, little effort is needed by the server, specifically the payload queuing effort, in this case. JMS is stateful and connected hence none of these requirements is needed after the initial connection.

Key findings:

The transaction response time reveals to be high as the payload increases as seen in Figure 5.5. This is possible due to the payload overhead time that dominated the entire web service transaction. The payload overhead time generally rises with the increment in the payload. Possible cause is that the normal payload is not constraint by any format during the exchange process as such the payload is not cached at the JVM. Any time the client is reading the message, it has to start over and fetches and concatenate the payload through the loop.

The client response time is higher than the server response time. This claim is evident from client response time trend line that indicates upward transition with the payload. The client starts the communication by obtaining a JNDI connection to the messaging server which provides the access to the connection factory and the queue. The server produces and place the payload on the queue while. The client always reads the data from queue one by one. This process is time demanding for the messaging client.

University of Malaya

5.3.2 Compressed Payload Response Time

Table 5.8 shows the response time for exchange of message with compressed payload using SOAP over JMS protocol.

Table 5.4: Response time for compressed payload (SOAP over JMS)

Payload (bytes)	Payload overhead time (ms)	Server response time (ms)	Client response time (ms)	Compre-ssion time (ms)	Decomp-ression time (ms)	Transaction response time (ms)
6203	2.00	1.80	5.60	19.40	10.75	39.55
9802	8.40	1.80	9.20	33.40	13.25	66.05
14248	3.20	2.20	23.00	36.60	29.25	94.25
19552	15.20	2.00	44.40	50.60	46.25	158.45
25676	5.60	2.60	25.20	66.60	38.50	138.50
32642	6.20	3.00	25.60	100.80	33.75	169.35
42582	12.80	2.60	85.20	103.60	94.00	298.20
51448	9.80	2.40	39.60	151.80	95.00	298.60
61175	17.20	2.80	53.80	168.00	111.25	353.05
71737	27.00	21.20	153.80	217.40	167.50	586.90
83144	33.60	8.40	57.60	271.80	75.75	447.15
95394	13.60	3.20	148.40	235.00	176.00	576.20

University of Malaya

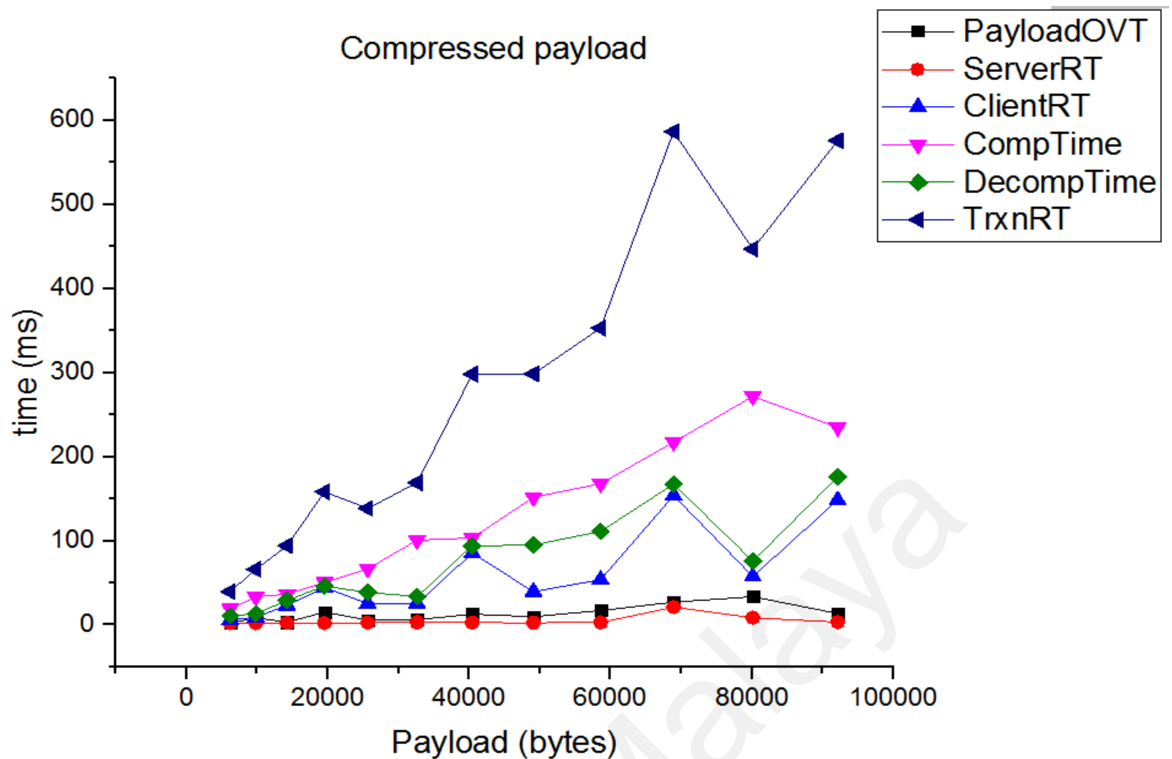


Figure 5.4: SOAP over JMS response time for compressed payload transaction comprising payload generation overhead time, server, client, compressed, decompress and overvall transaction response times

Figure 5.4 shows response times and overheads for SOAP over JMS compressed payload. In the compressed payload, except payload overhead time and server response time, all other metrics did not show a consistent pattern.

The client response time is higher than the server response time throughout the transaction process. server response time maintained a low level with a smooth regular trend except in two points when the payloads were 71.2KB and 83.1KB as it rises to 21.20ms and 8.40ms respectively. The payload overhead time on the other hand, fluctuated throughout the transaction process. The overhead time in the trend rises, especially at the payload of 9.8KB, 19.6KB, 72KB and 83KB with time as13.6ms, 8.4ms, 27ms and 33,6ms respectively.

Compression and decompression time are clearly not equal. The compression time is higher than the decompression time throughout the transaction.

University of Malaya

As can be seen from the figure, the transaction response time was irregular with peak time of 586ms at the payload size of 71.7KB. The trend rose to 298.2ms, 298.6ms and 586.9ms at the payload of 42KB, 51KB and 71.7KB respectively. The response time also rose to 576.2ms with payload of 95.4KB. The transaction response time showed wide time disparity compared to all the other metrics.

Key findings:

Both compression/decompression processes take CPU resources during encoding and decoding of the message. Compression is expensive due to a large number of comparisons in finding match patterns during encoding process in the LZ77 algorithm as proved by Kreft and Navarro (2010) and Mouli and Rajendra (2012), enforcing an encoding overhead thus utilizes a lot of server resources. Decompression, even though faster than the encoding it takes clients' resources. The spikes in the decompression overhead might be a delay due to space allocation for new incoming messages or delay by the messaging client JVM during garbage collection (Shams M. Imam, Vivek, & Sarkar, 2014).

The communication process is mostly handled by the client runtime by making request and accessing messages on the queue. The client invokes the send method of the messaging server to trigger the compression algorithm, and then decompress the message when upon received at the client side. This makes the client to have high response time compared to the server. The spikes in the client response time are caused possibly by the decompression activity to accommodate increasing memory loads by reclaiming unused memory or garbage collection by the client JVM as suggested by Du et al., (2013).

The server response time maintains a low linear transition indicating slight but constant overhead. In JMS, the plain underlying principle is, messaging server, once

University of Malaya

received an established a connection through the JNDI, it produces the message and put it on to the queue and allow the rest of the process to the client. As such, the server utilizes very little resources in the entire process.

Like server response time, the payload overhead time is low compare to other metrics. Likely, the message is either cached at the server messaging provider or the messaging client cache to avoid frequent request to the server. As such, less utilization is expected from the server.

5.4 Comparison between SOAP over HTTP and SOAP over JMS

Compared experimental results of normal payload for SOAP with HTTP binding and SOAP with JMS binding is shown in Table 5.5 and the graph of the compared bindings is depicted in Figure 5.5. While comparison of payload overhead of SOAP with HTTP binding and SOAP with JMS is shown in Table 5.6 and the graph is depicted in Figure 5.6.

Normal and compressed messages formats for both HTTP and JMS protocols were compared based on the experimental findings and the results are discussed in the following subsections 5.4.1 and 5.4.2.

5.4.1 Normal Payload for SOAP over HTTP vs SOAP over JMS

The following section shows the comparison of normal payload for the two protocols: SOAP over HTTP vs SOAP over JMS.

Table 5.5 shows the normal payload transaction response time for the exchange using SOAP over HTTP and the SOAP over JMS protocol. From Figure 5.5, both Web services started with entirely different transaction response times and moved through the transition. Although the SOAP over HTTP shows higher response time, But the SOAP over JMS response time for was higher than the SOAP over HTTP at two points

University of Malaya

when the payloads were 11.9MB and 16.6MB, with the differences of 21.8ms and 50.2ms respectively.

Table 5.5: Normal payload transaction response times for SOAP over HTTP vs SOAP over JMS.

Normal Payload (bytes)	SOAP over HTTP (ms)	SOAP over JMS (ms)
1339772	184.67	76.61
2153205	185.33	127.89
3158034	233.00	155.36
4354259	309.33	232.38
5741880	354.33	247.48
7320897	417.33	402.61
9896768	620.00	505.16
11938326	644.00	665.77
14171279	925.67	713.34
16595628	796.67	846.91
19211374	1551.33	997.05
22249785	1841.00	1261.05

University of Malaya

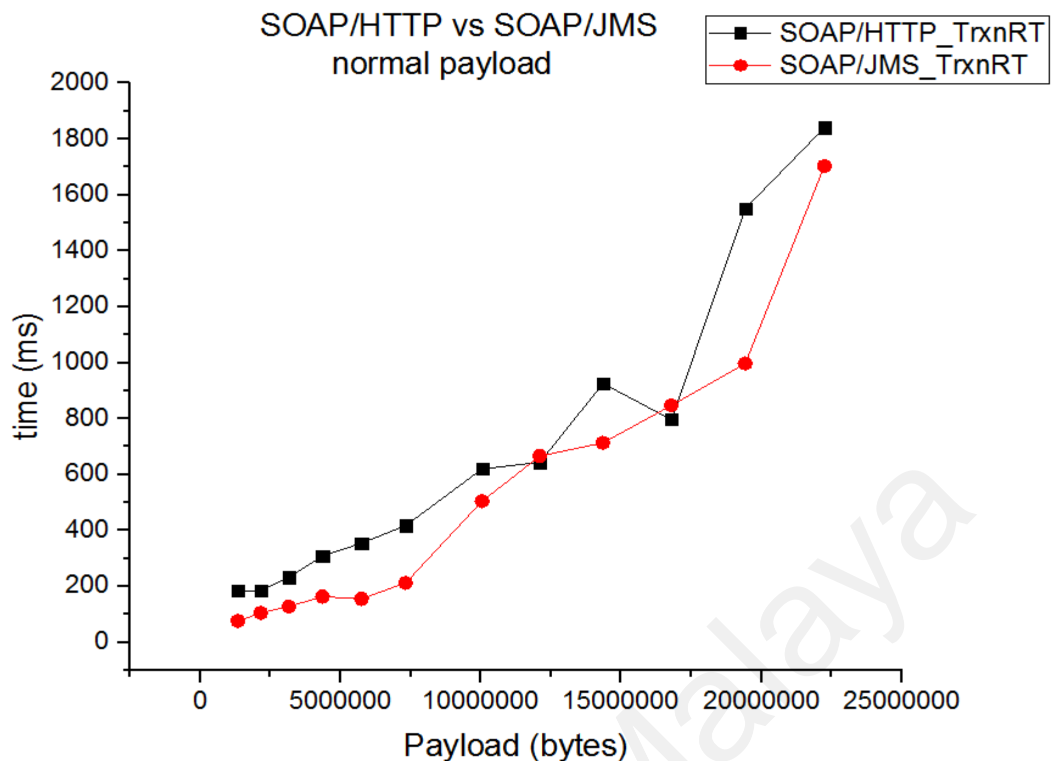


Figure 5.5: SOAP over HTTP vs SOAP over JMS response time for normal payload transactions response times

The SOAP over HTTP transaction response time began to be irregular at a point when the load size was increaed to 14.2MB. The time shot up to 925.6ms and then recessed to 796.7ms against a load size of 16.6MB, but went up significantly to 1551.3ms against the payload of 19.2MB. The SOAP over JMS response time was also irregular, though maintaned a steady movement between payload sizes of 1.3MB and 7.3MB. It then rose normally with the payloads were increased, but when the payload size was increased to 22.2MB, the trend apparently shot up.

Key findings:

As seen from Figure 5.5, the response time of normal payload for SOAP over HTTP revealed to be high in comparison with the SOAP over JMS. HTTP is a request-based protocol and connectionless, the client request makes a request to the server and disconnects once a request is made. The server processes and the client requests by re-establishes the connection. The process of making request each time transaction is to be

made informed the SOAP over HTTP busy and resource-intensive (Massimiliano et al., 2013).

JMS is an API with an abstraction of some interfaces and classes required by a client to communicate with a server in the messaging services. Connection is established once and endpoints stay connected in a stateful mode throughout the communication process. The client establishes the request and reads the messages one by one from the queue. This operation revealed to be expensive for the client and affects the performance of the messaging web service. The client is the side that makes most of the process by reading the payloads one at a time from the server.

Considering the effects of the two binding protocols, SOAP over JMS is better than SOAP over HTTP. The response time in SOAP over JMS is low, and HTTP binding incurs overhead as a result of message encapsulation and the constant check for awaiting message by the client.

University of Malaya

5.4.2 Compressed Payload for SOAP over HTTP vs SOAP over JMS

The following section shows the comparison for compressed payload for the two protocols: SOAP over HTTP and SOAP over JMS.

Table 5.6: Compressed payload transaction response times for SOAP over HTTP and SOAP over JMS

Compressed payload (bytes)	SOAP over HTTP (ms)	SOAP over JMS (ms)
6203	144.00	39.55
9802	177.00	66.05
14248	216.67	94.25
19552	313.00	158.45
25676	428.67	138.50
32642	665.33	169.35
42582	920.33	298.20
51448	1197.00	298.60
61175	1665.33	353.05
71737	2791.00	586.90
83144	4087.00	447.15
95394	4398.33	576.20

University of Malaya

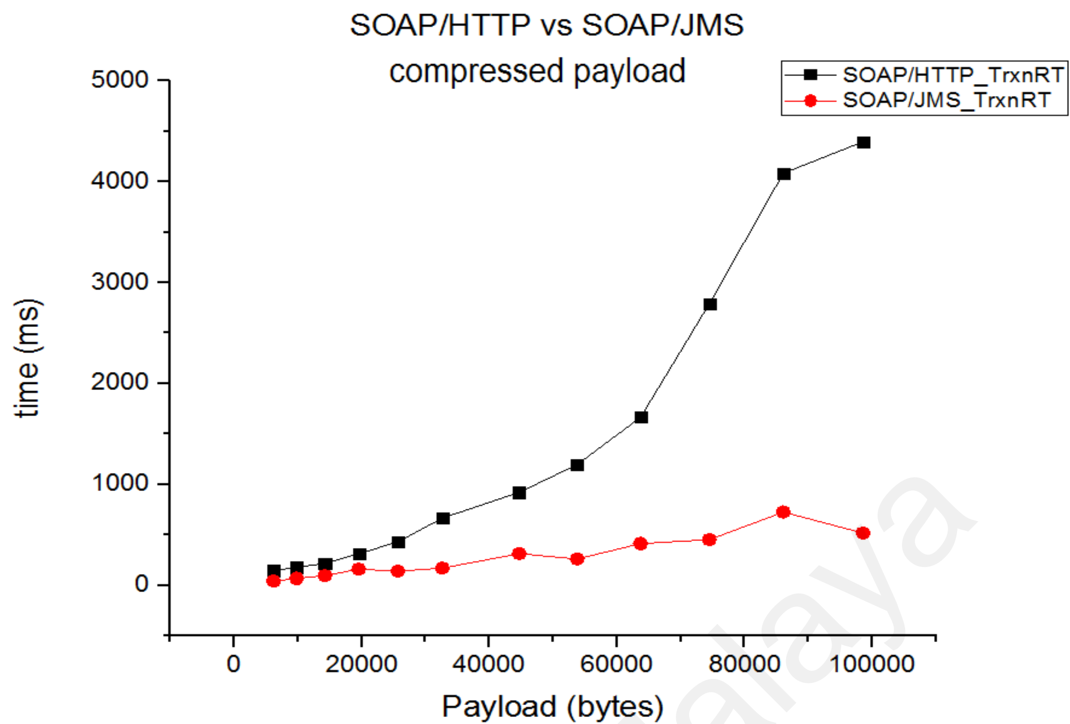


Figure 5.6: Compressed payload transactions response times for SOAP over HTTP vs SOAP over JMS protocols.

As seen in Table 5.6 and Figure 5.6, the SOAP over JMS compressed transaction has low response time while the SOAP over HTTP is higher. Both trends appeared to be in a regular transit throughout the web services communication. SOAP over HTTP response time rose at a higher rate as the payload size was increased. On the other hand, the SOAP over JMS response time grew slower along the transition with slight differences in the response times. From the early stage of the transaction of payload size between 6.2KB and 19.6KB, the disparity was small. The disparity was much widened when the payload was increased to 51.4KB and continued to grow wider until the end of the transaction.

Key findings:

Figure 5.6 shows the compressed payload for both binding protocols. It revealed that the transaction response time for the SOAP over HTTP binding is significantly higher than that of SOAP over JMS binding. The HTTP binding is almost four times higher

than the JMS binding at averagely around 26.6%. This reason might be related to the stateless mode of HTTP request and the GET and POST methods that demands every transaction to be connected again. In addition, HTTP constantly checks for awaiting message by the client. This process utilizes a lot of CPU resources and causes degradation on the entire performance of the messaging system, especially the client side.

The trend of the SOAP over JMS binding protocol reveals to be good with almost 75.4% less response time than the HTTP binding. The major reason for the lower response time is the fact that JMS is stateful and stay connected once the connection services are established. The client establishes the request and reads the messages from the queue in succession. This reduces amount of access to the CPU for a request.

Considering the effects of the two binding protocols, SOAP over JMS is better than SOAP over HTTP. The HTTP binding incurs overhead as a result of message encapsulation for the HTTP GET and POST (Butek, 2005) and the constant check for awaiting message by the client.

5.5 Messaging Communication Delivery Analysis

Both messaging Web services were executed 50 times each and the numbers of success message delivery were recorded. Table 5.7 shows the number of successful deliveries of the messages from the messaging server to the messaging client for both binding protocols.

University of Malaya

Table 5.7: Normal and compressed payloads success rate for SOAP/HTTP and SOAP/JMS

Normal payload (bytes)	No. of success		Compressed payload (bytes)	No. of success	
	HTTP	JMS		HTTP	JMS
1339772	50	50	6203	50	50
2153205	50	50	9802	50	50
3158034	50	50	14248	50	50
4354259	50	50	19552	50	50
5741880	50	49	25676	50	50
7320897	50	50	32642	49	50
9896768	50	50	42582	50	49
11938326	50	50	51448	50	50
14171279	49	50	61175	49	50
16595628	49	50	71737	50	50
19211374	48	49	83144	48	49
22249785	48	50	95394	49	50

University of Malaya

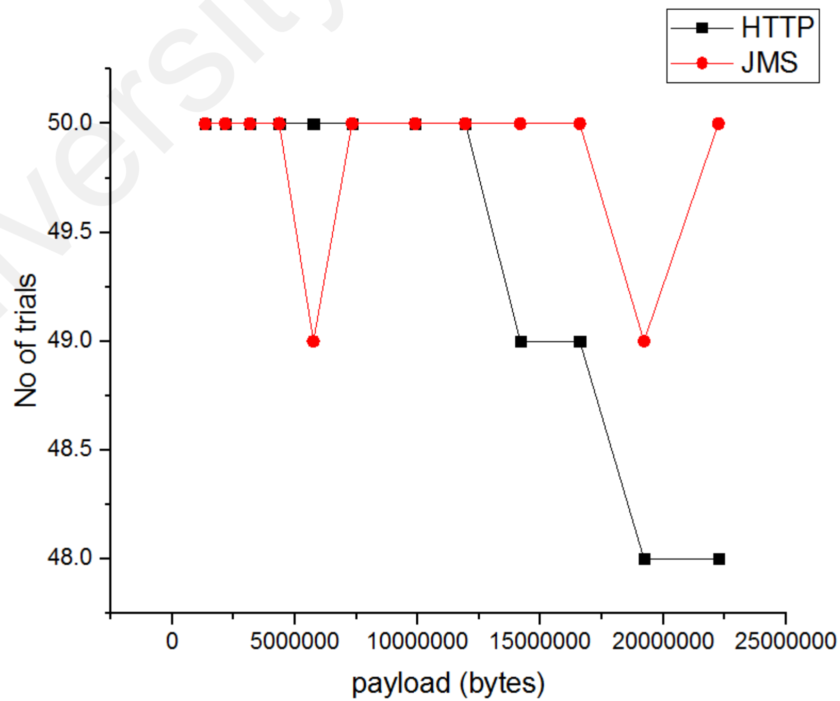


Figure 5.7: Normal payload successful delivery for SOAP over HTTP and SOAP over JMS

Figure 5.7 shows the successful messaging services for a normal payload. The graph reveals that the JMS binding on SOAP performed well as the trend indicates a smooth transition except at two points missing to deliver the payload. HTTP binding delivered successfully in the beginning but failed one time when the payload was 14.2MB and 16.6MB. And also, when the payload is 19.2MB and 22.4MB, the HTTP binding failed to deliver the payload two times at each point.

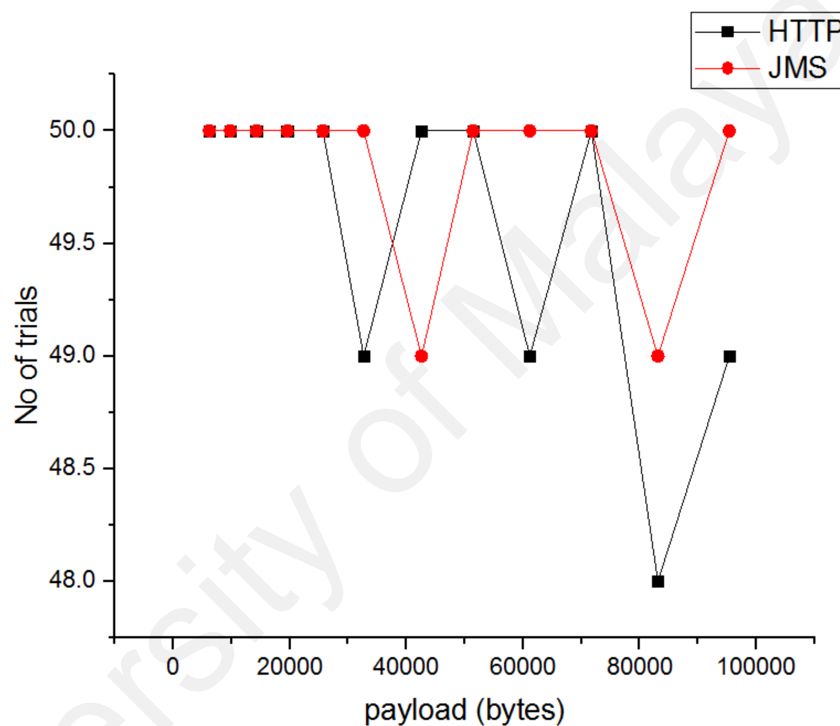


Figure 5.8: Compressed payload success rates for SOAP/HTTP and SOAP/JMS.

University of Malaya

Figure 5.8 shows the successful messaging services for a compressed payload. The graph revealed that the JMS binding on SOAP performed well as the graph indicates that the exchange failed two times, one at the payload of 42.6KB and also one time at the payload of 83.1KB. In the HTTP binding, the payloads were as well delivered but failed 5 times. One time each at the payload of 32.6KB, 42.7KB, 61.2KB and 95.4KB respectively. Also, the HTTP binding failed to deliver two times at the payload of 83.1KB.

Key findings:

Figure 5.7 shows that the highest number of failures recorded is from HTTP binding normal payload for SOAP over HTTP failed to deliver 6 times. As revealed from earlier analysis, HTTP is a request-based protocol as such it might be overloading the server with a lot of requests. This might force the server to go out of resources as result, the server could befall unresponsive when the JVM cannot withhold the produced message due to memory allocation or swapping.

The normal payload for SOAP over JMS failed to deliver 2 times. This happens likely when the CPU is overwhelmed by internal processes and breaks the I/O activity for a long time and the JVM misses the track of its activity.

Figure 5.8 shows the compressed payload for the two binding protocols. HTTP binding failed to deliver the payload 5 times. This failure is related with the fact that HTTP always requests service from the server and overload the server with full of request and increased message. This process will compel the server to be saturated with demand and payloads and subsequently run out of computing resources and become passive due to many garbage or swapping memory location or allocation.

The compressed payload for SOAP over JMS failed to deliver 2 times. This might be caused by the client when allocating new memory for incoming messages. The client may fail to register and acknowledge the message and will be sent to error destination. Another possible cause of failure in the delivery is from the compression algorithm. The algorithm is programed to increase the buffers of search and look-ahead on-the-fly. However, modifying this parameter can cause the JVM to suddenly allocate memory and this impromptu request can delay or cease the I/O activity (Hines, Gordon, Silva, Da Silva, Ryu, & Ben-Yehuda, 2011).

University of Malaya

5.6 Summary

In this chapter, the result of the research on the effect of high payload on the performance of SOAP Web services is presented and discussed. Two message formats: normal and compressed (modified LZ77 algorithm) SOAP payload were exchanged using HTTP and JMS bindings protocols. The results for these formats were compared and analyzed to observe the effect of the two binding protocols at different scenario. Response time and overhead time were discussed and possible reasons were deduced and explained. The assessment of number of delivered messages is discussed and reasons of delivery failure in some transactions were discussed.

University of Malaya

CHAPTER 6: CONCLUSION

6.1 Introduction

The previous chapter has discussed and analyzed the findings of the research. This chapter concludes the study of the SOAP performance for high volume messaging Web services by providing evidences from the previous chapters supporting the research questions of this research. The chapter provides the main findings based on the related researches and methods applied in implementation of this research and direction for potential research in the future.

6.2 Research Aims and Objectives

In order to conclude the process of this research, the research objectives were reviewed with respect to the findings of the research in Chapter 5.

6.2.1 Research objective 1

To implement an approach for high payload exchange in SOAP Web services.

With a view to examine the effect of SOAP payload on Web Services' delivery time in terms of response time, two web services were implemented: SOAP with HTTP binding as the benchmark and SOAP with JMS binding as the experimental study. Look-ahead buffer of LZ77 compression algorithm was modified to accommodate high number of bits. The algorithm was introduced into the two web services. Figure 4.8, Figure 4.9 and Figure 4.10 illustrate the Web services.

6.2.2 Research objective 2

To determine the high payload response time and overheads in the implemented approach.

University of Malaya

To assess the performance of payload on the web services, same amount of payloads in normal and compressed format were transmitted using both web services. Payloads from 1.3MB to 22.5MB were subsequently exchanged and the response time and overheads time per each transaction were monitored and automatically recorded from both ends. Figure 3.3 shows the flow of execution of the Web services and how the results were captured for the analysis. The raw results collected from the exchange of the two Web services are shown in Appendices E1 – F2.

While communicating the normal payload, the HTTP binding messaging client constant requests cause the overall messaging response time to be high. In the exchange with the JMS binding, the messaging client makes most of the transaction effort in establishing connection and constantly reading queue messages. Thus, the client overhead becomes high and eventually affect the overall execution time. Analysis of the findings is an evidence for JMS binding on SOAP messages to performed better due to less response time and overhead time.

JMS binding on compressed SOAP payload got some spikes in the client response time and the decompression overhead time as seen in Figure 5.4. This might be due to CPU process as the client side where JVM regularly claimed unused memory. Messaging client connecting and requesting the server causes high overhead for the client. Despite soaring compression time, the server overhead was still less than the client time. Contrary to the HTTP binding, the JMS binding proved that compression/decompression overhead takes more of the CPU resources than the server and client processes. Compression is costlier due to search/match during compression process. Server overhead is smooth and less in the messaging process. Spikes during the compression affect the overall response time.

University of Malaya

6.2.3 Research objective 3

To evaluate the performance of the implemented approach and prototype in terms of response time and the overhead.

The Web services transaction was executed 50 times in order to determine the successful delivery of the exchanged payloads. Acknowledgement was sent by the client and recorded at the server after every successful delivery in all transactions.

From 50 trials of the message exchange process, in the HTTP binding normal payload failed to deliver 6 times and compressed payload failed to deliver 5 times. Reasons for the HTTP binding to record higher failure rate might be attributed to the constant HTTP requests by the client that can lead the server to go out of resource and become unresponsive.

For the JMS binding, normal payload failed to deliver 2 times and the compressed payload failed to deliver 2 times. This might be due to server CPU out of resources and JVM become unresponsive due to I/O delay. Other causes of this failure might be that the messaging server was saturated with requests and paused. Conjointly, JMS binding may miss delivery as a result of buffer delay by compression algorithm or JVM paused due to I/O delay.

The overall findings of these results observe that using the modified LZ77 algorithm, SOAP over JMS has proved to outperform the SOAP over HTTP. The JMS binding protocol reveals to be impressive with almost 75.4% less response time than the HTTP binding. Compressed version of 22.2MB was exchanged at 0.6 seconds. Improving the Java heap size *"USER_MEM_ARGS"* of the server can improve the overall messaging performance.

University of Malaya

Compressing the SOAP message using the modified LZ77 algorithm over the JMS binding has yielded a remarkable performance. This put together the combination to be a good candidate for messaging when considering low response time and assurance for delivery of high payload is needed.

6.3 Contributions

The findings of this study have vital contributions to numerous areas of Web services engineering. This study has identified several requirements for improving SOAP message performance delivery.

Very important finding that is beneficial to enterprises and business-to-business solutions is the improvement in the performance of the messaging system. This finding reduces the response time and assures the successful delivery of high payload. Research objectives one and three demonstrated these assertions. This research is also valuable in communicating critical mission payload with low and unreliable bandwidth.

The modified LZ77 compression algorithm has offered the capacity for large data by reducing the number of checks by the search buffer during encoding while JMS offers asynchronous and loose coupling in the implementation. This is established in research objective one. This combination materialized a stable landscape that reduces network influence on the communication system.

Programmers and performance analysts seeking to identify system performance will find the evidence of CPU utilization in data compression, message parsing and computing time analysis. This research provides an insight on how system resources are utilized in the management of payload with normal and compressed message formats.

University of Malaya

This research will also be of benefit to programmers and software architects interested in analyzing and evaluating load performance and its general effect on both server and client nodes. The web services have offered an intuition on how the payloads are processed at both the server and the client ends.

6.4 Limitations

The main aim of this study is to improve SOAP performance. Two SOAP web services were implemented to test the three research questions related to the main aim. Findings were obtained and analyzed. The findings found to be significant but have some limitations.

Performance degradation is high at the client side. Most of the computing activities occur at the client end as such the response time and overhead are revealed to be high. Spikes resulting from compression increase the latency of the communication system and eventually degrade the overall performance. The implementation did not cover the wire aspect of the communication system.

6.5 Future Work

Implementing a cache at the client end will aid in reducing the latency and increasing the speed of request/response. Incoming payload will be compared with the first one and similar parts of the payload will be used instead of processing as new. A procedure for selecting message format by the client can also improve the performance at the client endpoint.

Future research in this area should also take the compression algorithm into cognizance. The algorithm needs to be optimized to make the search buffer more effective during the search/compare process when compressing the payload. If optimized, it will enhance the compression cost and the server-side response time.

University of Malaya

Another area of further consideration is the implementation of this research using different network strength. This will be vital to ascertain the system in different scenarios to further identify the level of efficiency of these research findings. This could include both WAN and LAN.

University of Malaya

REFERENCES

Aali, S. H., & Farkhady, R. Z. (2011). *A Combination Approach for Improving Web Service Performance.* Presented at the Proceedings of the International Multi Conference of Engineers and Computer Scientists, Delhi, 2011, India: Prentice Hall.

Abbas, A. M., Bakar, A. A., & Ahmad, M. Z. (2014). *Fast dynamic clustering SOAP messages based compression and aggregation model for enhanced performance of Web services.* Journal of Network and Computer Applications, *2(41)*, 80-88.

Abhaya, V. G., Tari, Z., & Bertok, P. (2012). *Building Web services middleware with predictable execution times*. World Wide Web-Internet and Web Information Systems, *15*(5-6), 685-744.

Abu-Ghazaleh, N., & Lewis, M. J. (2005). *Differential checkpointing for reducing memory requirements in optimized SOAP deserialization.* Conference on Grid Computing, 2005, New Jersey: IEEE.

Abu-Ghazaleh, N., & Lewis, M. J. (2006). *Lightweight Checkpointing for Faster SOAP Deserialization.* Conference on Web Services, New Jersey: IEEE.

Abu-Ghazaleh, N., Lewis, M. J., & Govindaraju, M. (2004). *Performance of Dynamically Resizing Message Fields for Differential Serialization of SOAP Messages.* Conference on Internet Computing, 2004, Madrid, Spain: IJCA.

Ahmad, F., Sarkar, A., & Debnath, N. C. (2014). *Analysis of dynamic web services.* Conference on Computing, Management and Telecommunications, 2014, Stockholm, Sweden: IEEE.

Aihkisalo, T., & Paaso, T. (2012). Latencies of service invocation and processing of the rest and SOAP web service interfaces. Conference on Web services, (pp. 100-107), Birmingham, UK: IEEE.

Al-Shammary, D., & Khalil, I. (2010). *SOAP web services compression using variable and fixed length coding.* Conference on Network Computing and Applications (NCA), , 2010, Athens, Greece: IEEE.

Appel, S., Frischbier, S., Freudenreich, T., & Buchmann, A. (2012). Eventlets: Components for the integration of event streams with SOA. Presented *IEEE International Conference on service-Oriented Computing and Applications (SOCA),* (pp. 1-9), Lisbon: IEEE.

Arteaga, D., & Zhao, M. (2014). Client-side flash caching for cloud systems. *Proceedings of International Conference on Systems and Storage* (pp. 1-11): ACM.

Banditwattanawong, T., & Uthayopas, P. (2013). *Improving cloud scalability, economy and responsiveness with client-side cloud cache.* Conference on Computer, Telecommunications and Information Technology (ECTI-CON), Rome, Italy: ECTI.

University of Malaya

Barrington, A., Feldman, S., & Dechev, D. (2015). A scalable multi-producer multi-consumer wait-free ring buffer. *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (pp. 1321-1328): Delhi, India.

Belazzougui, D., Kärkkäinen, J., Kempa, D., & Puglisi, S. J. (2016). Lempel-Ziv Decoding in External Memory. *International Symposium on Experimental Algorithms* (pp. 63-74), Springer: Cham.

Bell, T. C., Cleary, J. G., & Witten, I. H. (1990). *Text compression 5*(348), 79-90, US: Prentice Hall Englewood Cliffs.

Bonetta, D., Peternier, A., Pautasso, C., & Binder, W. (2012): A Scripting Language for High-Performance RESTful Web Services: ACM *Sigplan Notices, 47*(8), 97-106.

Bosin, A., Dessì, N., & Pes, B. (2011). Extending the SOA paradigm to e-Science environments. *Future Generation Computer Systems, 27*(1), 20-31.

Bou, S., Amagasa, T., & Kitagawa, H. (2014). Keyword search with path-based filtering over XML streams. *Symposium on Reliable Distributed Systems (SRDS)* (pp. 337-338), Anakra, Turkey: IEEE.

Bu, H. (2011). Metrics for service granularity in Service Oriented Architecture. *Conference on Computer Science and Network Technology (ICCSNT),* Vol. 1, 491-494, Toronto, Canada: IEEE.

Bulus, H. N., Carus, A., & Mesut, A. (2017). A new word-based compression model allowing compressed pattern matching. *Turkish Journal of Electrical Engineering & Computer Sciences, 25*(5), 3607-3622.

Brydon, S. P., & Singh, I. (2010). *U.S. Patent No. 7,702,724*. Washington, DC: U.S. Patent and Trademark Office.

Bzoch, P., & Safarink, J. (2013). *Simulation of client-side caching policies for distributed file systems*, Conference on Distributed systems, Rome, Italy :IEEE.

Cao, M. D., Dix, T. I., Allison, L., & Mears, C. (2007). *A simple statistical algorithm for biological sequence compression.* Conference on Data Compression, Delhi, India: IJCA.

Chakraborty, D., Ghosh, D., & Ganguly, P. (2015). A Dictionary based Efficient Text Compression Technique using Replacement Strategy. *International Journal of Computer Applications, 116*(16), 116-123.

Chow, M., Meisner, D., Flinn, J., Peek, D., & Wenisch, T. F. (2014). The Mystery Machine: End-to-end Performance Analysis of Large-scale Internet Services. Journal of Computing and Internet Services *23(6)*217-231.

Dhore, S. R., Gangwar, H., Mishra, P., Sharma, R., & Singh, R. (2012). *Systematic approach for composing Web Service using XML.* Conference Computing Communication & Networking Technologies, Wolverhampton, UK: Springer.

University of Malaya

Du, Y. N., Zhao, Y. L., Han, B., & Li, Y. C. (2013). Optimistic parallelism based on speculative asynchronous message passing. *Journal of the Chinese Institute of Engineers, 36*(1), 35-47.

Eugène, E. C., & Fréjus, L. A. (2012). Asynchronous Message Exchange System between Servers based on Java Message Service API, *IEEE letters, 35*(3), 534-542.

Fei, S., Ke, Y., Lin, Z., & Xiaofei, W. (2010). *A performance evaluation method and it's implementation for web service.* Paper presented at the ic-bnmt Conference, 2010: Beijing, China.

Fiala, E. R., & Greene, D. H. (1989). Data compression with finite windows. *Communications of the Acm, 32*(4), 490-505.

Fu, C. Y., Belqasmi, F., & Glitho, R. (2010). RESTful Web Services for Bridging Presence Service across Technologies and Domains: An Early Feasibility Prototype. *Ieee Communications Magazine, 48*(12), 92-100.

Gerić, S., & Vrček, N. (2009). *Prerequisites for successful implementation of Service-Oriented Architecture.* Paper presented at the Information Technology Interfaces, 2009. ITI'09. Proceedings of the 2009 31st International Conference on Information Technology Interface, 2009, Ankara, Turkey: ITT.

Girtelschmid, S., Steinbauer, M., Kumar, V., Fensel, A., & Kotsis, G. (2014). On the application of Big Data in future large-scale intelligent Smart City installations. *International Journal of Pervasive Computing and Communications, 10*(2), 168-182.

Hines, M. R., Gordon, A., Silva, M., Da Silva, D., Ryu, K., & Ben-Yehuda, M. (2011). Applications know best: Performance-driven memory overcommit with ginkgo. Conference on *Cloud Computing Technology and Science* (pp. 130-137). Athens, Greece: IEEE.

Hansen, A., & Lewis, M. C. (2018). Modified Huffman Code for Bandwidth Optimization Through Lossless Compression *Information Technology-New Generations 4*(3), 761-763.

Hong, Y., Zhang, S., Wang, R. Y., Li, Z., & Liu, D. X. (2016), Text Compression and Decompression, *US20160197621A1*, EMC Corp: Google Patents.

Hu., S. X. (2006). Interoperability at the SOAP message level, retrieved on 17th January, 2016 from http://www.ibm.com/developerworks/library/ws-soa-intersoap/

Iqbal, R., Shah, N., James, A., & Cichowicz, T. (2013). Integration, optimization and usability of enterprise applications. *Journal of Network and Computer Applications, 36*(6), 1480-1488.

Isaac, S., & Devi, V. U. (2014). Isaac, S. G. C., & Devi, V. U. (2014). Efficient Querying and SOAP Based Streaming of Multimedia Content Using WEB Services. *Conference on Intelligent Computing*, 37-41, Toronto, US: IEEE.

University of Malaya

Jendrock, E., Cervara-Navarro, R., & Evans, I. (2017). Java EE 6 Tutorial. *https://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html.* 6th. Retrieved on 16/08/2017.

Juric, M. B., Rozman, I., Brumen, B., Colnaric, M., & Hericko, M. (2006). Comparison of performance of Web services, WS-Security, RMI, and RMI–SSL. *Journal of Systems and Software, 79*(5), 689-700.

Kalyani, K. (2012). Recent Trends and Challenges in Enterprise Application Integration, *International Journal of Application or Innovation in Engineering & Management*, *1*(4), 62-71.

Kanoun, K., & Van der Schaar, M. (2015). Big-data streaming applications scheduling with online learning and concept drift detection. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, 1547-1550, EDA Consortium: Shefield, UK.

Kärkkäinen, J., Kempa, D., & Puglisi, S. J. (2016). Lazy Lempel-Ziv factorization algorithms. *Journal of Experimental Algorithmics (JEA), 21*, 2.4.

Kiran, D., & Andresen, D. (2003). SOAP optimization via parameterized client-side caching. *Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, 785-790.

Koulouzis, S., Cushing, R., Karasavvas, K., Belloum, A., & Bubak, M. (2012). Enabling web services to consume and produce large datasets. *Internet Computing, IEEE, 16*(1), 52-60.

Kruse, H., & Mukherjee, A. (1997). Data compression using text encryption. *Data Compression Conference, 1997. DCC'97.* (p. 447), UT, USA: IEEE.

Kumari, S., & Rath, S. K. (2015). Performance comparison of soap and rest based web services for enterprise application integration, *International Conference on Advances in Computing, Communications and Informatics (ICACCI),* 1656-1660: IEEE.

Kumawat, H., & Chaudhury, J. (2013). Optimization of LZ77 Data Compression Algorithm. *International Journal of Computer Engineering and Technology, 4*, 42-48.

Lam, G., & Rossiter, D. (2013). A web service framework supporting multimedia streaming. *IEEE Transactions on Services Computing*, *6*(3), 400-413.

Larsson, N. J., & Moffat, A. (2000). Off-line dictionary-based compression. *Proceedings of the IEEE, 88*(11), 1722-1732.

Liu, W., Mei, F., Wang, C., O'Neill, M., & Swartzlander, E. E. (2018). Data Compression Device Based on Modified LZ4 Algorithm. *IEEE Transactions on Consumer Electronics, 64*(1), 110-117.

Liv, J., Wang, Y., & Zhong, Y. (2015). Efficient XML Document Compressing Method Based on Internet of Things. *Conference on Intelligent Systems Research and Mechatronics Engineering, Zhengzhou, China*: IEEE.

University of Malaya

Mahmood, A., Islam, N., Nigatu, D., & Henkel, W. (2014). DNA inspired bi-directional Lempel-Ziv-like compression algorithms. *8th International Symposium on Turbo Codes and Iterative Information Processing (ISTC),* 162-166, *2014*, Bremen, Germany. IEEE.

Malladi, S. K., Murphy, R. F., & Deng, W. (2017). *U.S. Patent No. 9,852,116*. Washington, DC: U.S. Patent and Trademark Office.

Menarini, M., Seracini, F., Zhang, X., Rosing, T., & Krüger, I. (2013). Green web services: Improving energy efficiency in data centers via workload predictions. *2013 2nd International Workshop on Green and Sustainable Software (GREENS),* (pp. 8-15), May 2013,Warsaw, Poland: IEEE.

Mohamed, A. W., & Zeki, A. M. (2017). Web services SOAP optimization techniques. *Conference on Engineering Technologies and Applied Sciences* (pp. 1-5), Vienna, Australia: IEEE.

Mutange, K., Okeyo, G., Cheruiyot, W., Sati, A., & Kalunda, J. (2014). A Review of SOAP Performance Optimization Techniques to Improve Communication in Web Services in Loosely Coupled Systems. *International Journal of Computer Science Issues, 11*(1), 142-158.

Nakamoto, Y., & Akiyama, S. (2015). A Proposal for Mobile Collaborative Work Support Platform Using an Embedded Data Stream Management System. 23-28.

Nitin N., Paul J., Davies P., & David N. (2016). Native Web Communication Protocols and Their Effects on the Performance of Web Services and Systems. Conference on Computer and Information Technology (219-225), Nadi, Fiji: IEEE.

Oswald, C., & Sivaselvan, B. (2018). An optimal text compression algorithm based on frequent pattern mining. *Journal of Ambient Intelligence and Humanized Computing*, *9*(3), 803-822.

Pahl, C. (2001). Components, contracts, and connectors for the unified modelling language UML, *International Symposium of Formal Methods Europe* (pp. 259-277), 2001, March, Berlin, Heidelberg: Springer.

Pavan Kumar, P., Sanjay, A., Karthikeyan, U., & Zornitza, P. (2013). Comparing Performance of Web Service Interaction Styles: SOAP vs. REST. *Journal of Information Systems Applied Research, 6*(1) 118-129.

Pawar, S., & Chiplunkar, N. N. (2017). Open source apis for processing the XML result of web services. *Conference on Advances in Computing, Communications and Informatics* (pp. 1848-1854). IEEE.

Perez-Castillo, R., de Guzman, I. G. R., Caballero, I., & Piattini, M. (2013). Software modernization by recovering Web services from legacy databases. *Journal of Software-Evolution and Process, 25*(5), 507-533.

Pinto, S. H., Anand, G., Truong, B., Sundaresan, S. R., & Vk, K. S. (2017). *U.S. Patent No. 9,594,846*. Washington, DC: U.S. Patent and Trademark Office.

University of Malaya

Pirnau, M. (2010). Implementing Web Services Using Java Technology. *International Journal of Computers Communications & Control, 5*(2), 251-260.

Policriti, A., & Prezza, N. (2016). Computing LZ77 in run-compressed space. Conference on *Data Compression (DCC),* (pp. 23-32). Houston TX, US: IEEE.

Saab, C. B., Coulibaly, D., Haddad, S., Melliti, T., Moreaux, P., & Rampacek, S. (2012). An integrated framework for web services orchestration. In *Innovations, Standards and Practices of Web Services: Emerging Research Topics* (pp. 306-335): IGI Global.

Salomon, D., & Motta, G. (2010). Dictionary Methods. In *Handbook of Data Compression* (pp. 329-441). Springer, London.

Sarkas, N., Das, G., Koudas, N., & Tung, A. K. (2008). Categorical skylines for streaming data. In the *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 239-250), 2008, Christchurch, New Zealand: ACM.

Sayood, K. (2002). *Lossless compression handbook*: Elsevier.

Sha, F., Yu, K., Zhang, L., & Wu, X. (2010). A performance evaluation method and it's implementation for web service. In *Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference on* (pp. 218-222). IEEE.

Shanmugasundaram, S., & Lourdusamy, R. (2011). A comparative study of text compression algorithms. *International Journal of Wisdom Based Computing, 1*(3), 68-76.

Simon, B., Goldschmidt, B., Kondorosi, K. (2013), A Metamodel for the Web Services Standards, Journal of Grid computing (11)4, 735-752.

Sriwiroj, C., & Banditwattanawong, T. (2015). An economic model for client-side cloud caching service. *7th International Conference on Knowledge and Smart Technology (KST),* (pp. 131-136), Dublin, Ireland,: IEEE.

Stevens, P. (2001), On use cases and their relationships in the Unified Modelling Language, *International Conference on Fundamental Approaches to Software Engineering* (pp. 140-155), 2001, Berlin, Heidelberg,: Springer.

Suzumura, T., Takase, T., & Tatsubori, M. (2005). *Optimizing Web services performance by differential deserialization.* Paper presented at the IEEE International Conference on Web Services, 2005, Orlando, FL, USA: IEEE.

Tapang, C. C. (2001). Web Services Description Language (WSDL) Explained, 2001, USA: *Microsoft Developer Network*.

Tekli, J. M., Damiani, E., Chbeir, R., & Gianini, G. (2012). SOAP Processing Performance and Enhancement. *Services Computing, IEEE Transactions, 5*(3), 387-403. doi: 10.1109/TSC.2011.11.

University of Malaya

Uemura, T., Kusumoto, S., & Inoue, K. (2001). Function-point analysis using design specifications based on the Unified Modelling Language. *Journal of software maintenance and evolution: Research and practice*, *13*(4), 223-243.

Val, P. B., Garcia-Valls, M., & Estevez-Ayres, I. (2009). Simple Asynchronous Remote Invocations for Distributed Real-Time Java. *Ieee Transactions on Industrial Informatics, 5*(3), 289-298.

Vandikas, K., Quinet, R., Levenshteyn, R., & Niemöller, J. (2011). Scalable service composition execution by means of an asynchronous paradigm. A paper presented at the *2011 15th International Conference on Intelligence in Next Generation Networks (ICIN),* (pp. 157-162), Berlin, Germany: IEEE.

Vasilakis, C., Lecznarowicz, D., & Lee, C. (2009). Developing model requirements for patient flow simulation studies using the Unified Modelling Language (UML). *Journal of Simulation*, *3*(3), 141-149.

Vernadat, F. (2002). UEML: towards a unified enterprise modelling language. *International Journal of Production Research*, *40*(17), 4309-4321.

W3Schools, (2014a), Introduction to Web Services, Aug, 2015, retrieved from http://www.w3schools.com/webservices/ws_wsdl_intro.asp.

W3Schools. (2014b). SOAP Introduction, Aug, 2015, retrieved from http://www.w3schools.com/webservices/ws_soap_intro.asp.

Wagner, S., Roller, D., Kopp, O., Unger, T., & Leymann, F. (2013). Performance Optimizations for Interacting Business Processes*. 210-216.

Williams, R. N. (1991). An extremely fast Ziv-Lempel data compression algorithm, *Data Compression Conference,* (pp. 362-371), *1991,* Snowbird, UT, USA: IEEE.

Wilson, J. (2010). Using WSDL Generator and SOAP with Cloud Computing for Enterprise Architectures, February 2010, Florida, US: Springer.

Yu, S. C., & Chen, R. S. (2003). Web Services: XML-based system integrated techniques. *The Electronic Library*, *21*(4), 358-366.

Zhang, W., Cao, J., Zhong, Y., Liu, L., & Wu, C. (2008). *An integrated resource management and scheduling system for grid data streaming applications.* Paper presented at the International Conference on Grid Computing, 2008 9th IEEE/ACM, Tsukuba, Japan: IEEE.

Zimmermann, O., Tomlinson, M., & Peuser, S. (2012). *Perspectives on Web Services: Applying SOAP, WSDL and UDDI to Real-World Projects, 2012, 45-53*: Springer Science & Business Media.

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE transactions on Information Theory, 23*(3), 337-343.

University of Malaya

Zolfi, H., Lakdashti, A., & Vahidi, J. (2014), A Method for Performance Evaluation of SOAP Protocol, *Journal of Computing Academic Research, 6*(4)*,* 217-226.

University of Malaya