Enhanced Web Services Performance by Compression and Similarity-based Aggregation for SOAP Traffic

A thesis submitted for the degree of
Doctor of Philosophy

Dhiah Eadan Jabor Al-Shammary
B.Sc, M.Sc,
School of Computer Science and Information Technology,
Science, Engineering, and Technology Portfolio,
RMIT University,
Melbourne, Victoria, Australia.

November, 2013

Declaration

I certify that except where due acknowledgment has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

Dhiah Eadan Jabor Al-Shammary
School of Computer Science and Information Technology
RMIT University
November, 2013

Acknowledgments

I would like to extend my sincere thanks to my two supervisors Dr. Ibrahim Khalil and Professor Zahir Tari for their support and guidance over the period of my candidature.

I wish to thank the School of Computer Science and Information Technology at RMIT University for their material support, providing an excellent work environment for me to pursue my research interests.

I would like to thank all the staff and research students in the Distributed Systems and Networking (DSN) discipline for providing a peer group where support and feedback are readily available.

The experience was extremely valuable and fruitful.

Finally, I wish to thank my parents, brothers, sisters-in law, and my beloved wife for their patience, support and good humor during this period.

Credits

Portions of the material in this thesis have previously appeared in the following publications:

- Dhiah Al-Shammary, Ibrahim Khalil, Zahir Tari, "A distributed aggregation and fast fractal clustering approach for SOAP traffic", Journal of Network and Computer Applications, Available online 12 October 2013, ISSN 1084-8045, http://dx.doi.org/10.1016/j.jnca.2013.10.001.
- Dhiah Al-Shammary, Ibrahim Khalil, Zahir Tari, Albert Y. Zomaya, "Fractal self-similarity measurements based clustering technique for SOAP Web messages", Journal of Parallel and Distributed Computing, Volume 73, Issue 5, Pages 664-676, May 2013.
- Dhiah Al-Shammary, Ibrahim Khalil, "Redundancy-aware SOAP messages compression and aggregation for enhanced performance", Journal of Network and Computer Applications, Volume 35, Issue 1, Pages 365-381, January 2012.
- Dhiah Al-Shammary, Ibrahim Khalil, and Loay Goerge, "Clustering SOAP Web Services on Internet Computing Using Fast Fractals", Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on, pp.366-371, 25-27 August 2011.
- Dhiah Al-Shammary, Ibrahim Khalil, "Dynamic Fractal Clustering Technique for SOAP Web Messages", Services Computing, 2011. SCC 2011. IEEE International Conference on, pp.96-103, 3-9 July 2011.
- Dhiah Al-Shammary, Ibrahim Khalil, "Compression-based aggregation model for medical Web Services", Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE , pp.6174-6177, Aug. 31 2010-Sept. 4 2010.

- Dhiah Al-Shammary, Ibrahim Khalil, ”A new XML-aware compression technique for improving performance of healthcare information systems over hospital networks”, Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE , pp.4440-4443, Aug. 31 2010-Sept. 4 2010.

- Dhiah Al-Shammary, Ibrahim Khalil, ”SOAP Web Services Compression Using Variable and Fixed Length Coding”, Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on , pp.84-91, 15-17 July 2010.

Contents

List of Figures

List of Tables

Abstract

Many organizations around the world have adopted Web services, server farms hosted by large enterprises, and data centers for various applications. Web services offer several advantages over other communication technologies. However, it still has high latency and often suffers congestion and bottlenecks due to the massive load generated by large numbers of end users for Web service requests. Simple Object Access Protocol (SOAP) is the basic Extensible Markup Language (XML) communication protocol of Web services that is widely used over the Internet. SOAP provides interoperability by establishing access among Web servers and clients from the same or different platforms. However, the verbosity of the XML format and its encoded messages are often larger than the actual payload, causing dense traffic over the network.

Improving the performance of Web services by compressing SOAP messages is considered an important issue. Compression ratios achieved by most existing techniques are not high enough, and even a minute improvement could save tremendous amounts of network bandwidth in emerging cloud and mobile scenarios. This thesis tries to achieve this objective by proposing three innovative techniques capable of reducing small, as well as very large, messages. General XML trees and binary trees are constructed that support the encoding algorithm by removing the closing tags. Instead of encoding the characters of XML messages individually, fixed length and variable length (Huffman) encodings are developed to deal with XML tags as individual input items.

Furthermore, new redundancy-aware SOAP Web message aggregation models (Binary-tree, Two-bit, and One-bit XML status trees) are proposed to enable the Web servers to aggregate SOAP responses, and send them back as one compact aggregated message, thereby reducing the required bandwidth and latency, and improving the overall performance of Web services. XML message compressibility, the Jaccard-based clustering technique, and

the vector space model (VSM) are three similarity measurements that are developed to cluster SOAP messages based on their degree of similarity. Clustering based on similarity measurements enables the aggregation techniques to potentially reduce the required network traffic by minimizing the overall size of messages.

Fractal as a mathematical model provides powerful self-similarity measurements for the fragments of regular and irregular geometric objects in their numeric representations. Fractal mathematical parameters are introduced to compute SOAP message similarities that are applied on the numeric representation of SOAP messages. Furthermore, SOAP fractal similarities are developed to devise a new unsupervised auto-clustering technique. An extension of the aggregation model is proposed to further improve the performance of SOAP. A new distributed aggregation is developed to support aggregation of messages from multiple Web servers that share the path of SOAP responses over the Internet. On the other hand, a fast fractal similarity based clustering technique is proposed with the aim of speeding up the computations for the selection of similar messages to be aggregated together in order to achieve greater reduction.

Extensive experiments have shown high performance of the proposed compression techniques, with high compression ratios obtained, ranging from 7.38 to 8.31 for large-sized XML documents to 11.57 to 14.70 for very large-sized documents. Furthermore, the processing times of both compression and decompression were found to be extremely promising for several computing platforms, such as personal computers (PCs), laptops, netbooks, and personal digital assistants (PDAs). Experiments have also shown significant performance of aggregation techniques that achieved compression ratios as high as 25 for aggregated SOAP messages. Moreover, the fractal clustering technique is evaluated and experiments prove that it significantly improves the performance of Web services, and exceeds other clustering standards, such as K-means and principal component analysis (PCA) combined with K-means. Lastly, results have shown that the distributed aggregation has outperformed regular aggregation, resulting in a 100 percent higher compression ratio, and the fast fractal clustering has reduced the required processing time by 85 percent of the classical fractal clustering technique.

Chapter 1

Introduction

Web services are middleware that establish access to their components over the Internet using some network mechanisms and protocols, such as HTTP and TCP [Davis and Parashar, 2002]. A Web services interface is defined and described using Extensible Markup Language (XML) [Heinzl et al., 2006]. In recent years, usage of Web services has increased dramatically over the Internet due to their features, such as interoperability, the self-describing nature of XML-based interfaces, and message format [Andresen et al., 2004]. Moreover, the fact that they use successful Internet standards, such as HTTP, as a transport protocol is considered to be a major reason for proliferation of Web services over the net [Abu-Ghazaleh and Lewis, 2005]. Simple Object Access Protocol (SOAP) is the basic communication protocol for most regular Web services and Cloud Web services [Devaram and Andresen, June 23 - 26, 2003; Dikaiakos et al., 2009] as it supports Remote Procedure Call (RPC) and message semantics [Davis and Parashar, 2002].

Cloud Web services represent a new supplement, consumption and delivery model that provides a variety of services over the Internet [Werner and Buschmann, 2004]. Generally, Cloud provides dynamically scalable and usually virtual resources as a Web service that is available on demand over the Internet. Recently, adoption of Cloud Web services has increased significantly by many network organizations with the aim of providing the required services without investing heavily in the computing infrastructure [M. Nakagawa and Shimojo, 2006].

However, SOAP Web services inherit the disadvantages of XML, such as messages being larger than the actual payload of services [Werner and Buschmann, 2004], creating high network traffic. The use of XML strings for encoding SOAP messages is producing larger messages where only a very small part of the encoded SOAP message represents the real payload [Werner and Buschmann, 2004], and the remainder of these messages are XML and protocol overhead. Therefore, Web services, and especially the most popular ones, often suffer congestion and bottlenecks as a result of the large Web requests made by users for their services [Devaram and Andresen, June 23 - 26, 2003]. At the same time, Web services suffer the long execution times that are required for parsing and processing large XML messages [Andresen et al., 2004].

This research aims to provide effective solutions that could improve Web services' performance. The latency of Web services is investigated and analyzed in detail, with a concentration on the fact that Web services produce larger-sized messages than the actual payloads. Moreover, Web services are verbose in nature, resulting in high network traffic which keeps the network generally busy, but not available for all client requests. The processing time and network traffic reduction are considered main metrics for the proposed improvements (solutions). Furthermore, large numbers of requests/responses are considered in the proposed models to satisfy the growing network requirements.

1.1 Scope and goals

Improving the performance of Web services is a significant concern, and there are some serious problems being discussed and analyzed in this thesis that can broaden the knowledge of the theoretical and practical issues surrounding Web services. This research provides some significant benefits to both community and industry by creating new development aspects that could be applied by industrial fields to improve the current state of Web services middleware. Consequently, the community will receive better services through the use of more efficient Web services.

This thesis provides some scientific improvements to Web services and supports commercial Web applications, such as Banking and Quote stock systems, which require more effective tools (i.e. Web services). This research motivates commercial and economical systems to be more involved in using enhanced Web services for their users. For example, one important and familiar industry that could benefit from high-performance Web services is the travel sector, such as airline reservations and car rental services. It is a well-known fact that Web services have poor performance, and therefore, it is discouraging for commercial and economical applications to use them as their basic communication tool. Technically, improving the performance of Web services satisfies industry requirements in terms of providing a highly qualified infrastructure for IT industrial applications. This research is motivating industrial fields to potentially move towards information technology (IT) by providing more efficient Web services and better quality of service (QoS).

Practically, data compression provides some important benefits for Web service applications, especially for clients that have poor network connectivity [Werner and Buschmann, 2004]. Web compression-based applications use the lossless compression family as a technique that promises reducing the total size of a service message, and guarantees reconstructing the exact original message on the receiver's side. In this research, a set of compression techniques is proposed with the utilization of the tree data structure of SOAP messages with both fixed length and variable length (Huffman) encoding.

Aggregation of SOAP messages is another significant contribution of this thesis as it is based on utilizing the compression concepts with the aim to potentially reduce the size of aggregated messages. This new proposed solution will motivate future researchers to work on activating the multicast protocol to significantly minimize the required network bandwidth. This can be achieved by sending the aggregated message over the network with the development of powerful routing algorithms that have the ability to parse the compact aggregated message, and extract the required responses at the closest hop to the destination (client) address.

Furthermore, this study introduces new research topics to the Web services area, such as fractal models that have never been studied as a Web service technique. In fact, this will motivate other researchers to investigate these concepts and concentrate on the contributions they can make based on analyzing fractal Web services and proposing new solutions. Fractal is a well-known technique that has been applied in many applications, especially multimedia; however, it is a completely new aspect to Web services. Fractal can be applied in Web service applications by using the fractal self-similarity principle and other characteristics. In this thesis, the proposed fractal models suggest utilizing fractal characteristics in Web services after creating their numeric form. Numeric forms for Web services can be computed using XML tree indexing, Huffman binary tree coding, or Shannon-Fano coding, in addition to the TF-IDF scheme. Technically, fractal is proposed as a new alternative to the SOAP message similarity measurements to support aggregating messages with a greater capability of size reduction.

1.2 Research questions

Much work has been accomplished on improving SOAP performance and several solutions have been suggested to address SOAP Web services' congestion and bottlenecks. They used several concepts: compression by reducing the XML message size to reduce the network traffic [Werner and Buschmann, 2004; Davis and Zhang, 2002], building a caching system to increase the locality of messages at the client side [Liu and Deters, 2007], server side [Abu-Ghazaleh and Lewis, 2005], or both client/server sides [Andresen, 2005], binary encoding by encoding XML messages and transmitting them in binary instead of textual format [Julio Cezar Estrella and Monaco, September 22-24, 2008; Lu et al., 2006], optimizing the SOAP run-time implementation by improving the SOAP kernel performance using some effective optimizations, or multicasting based on the similarity of SOAP messages and grouping the similar ones in one message via a multicast protocol [Phan et al., 2008; Damiani and Marrara, October 31 - 31, 2008].

Although Web services are mainly built with tree data structures, only few studies have proposed the compression concept utilizing tree structures with XML messages. As such, this research has proposed new compression techniques that try to utilize both binary- and general-based tree structures of SOAP messages. The first two research questions relate to the Web service message tree structure, and attempt to explore the potential efficiency of developing both binary- and general-based tree compression techniques to minimize Web service volumes over the network. The second two questions relate to clustering SOAP messages based on detecting a high degree of similarity and a distributed aggregation (multi-Web server), and minimizing the required processing times for these tasks when handling large numbers of SOAP messages at the server side.

1. **How to reduce the size of SOAP messages sent/received over the network?** Minimizing the Web service messages' size has been widely studied and some unique compression models have been proposed. However, the outcomes of these models provide limited significant solutions to the large size of SOAP messages. Few studies have tried to utilize the general-based tree structure of XML messages since the concept of using these models has not been developed in a professional and technical way. Further, they usually add overhead parameters in order to keep the order of the tree representation that may increase the size of the encoded messages. Nor has the development of binary-based tree transformations for XML messages been proposed. Can a binary-based tree structure of SOAP messages in compression models potentially reduce the

total size of SOAP messages? Could the compression models using general-based tree structures outperform the binary-based tree models?

2. **How to utilize the compression techniques (redundancy-aware) in developing SOAP aggregation models?**
   Many studies have been done on reducing the total network traffic created by SOAP messages in the context of Web services. Several solutions have been proposed, such as caching of SOAP messages at the client side, server side or both, and multicast aggregated packets of pairs of SOAP messages. The results of these studies are still limited, in addition to the fact that they consume large amounts of both client and server storage space. Would the development of the compression concepts in aggregating SOAP messages group-wise, and not only as pairs, improve the reduction of total network traffic for SOAP messages?

3. **What are the cost-effective similarity parameters for clustering SOAP messages?**
   With the aim to aggregate SOAP messages, multicasting would be activated to minimize their total network traffic. Similarity parameters play a vital role in improving the outcomes of the aggregation models. While Jaccard coefficients and vector space models (VSM) are widely used on the Internet to compute the similarity of textual messages, they consider very simple cost metrics, such as the intersection of XML items and the cosine similarity of messages. In order to aggregate more than two messages in one packet, general clustering techniques, such as K-Means and principal component analysis (PCA) combined with K-Means, can be considered. However, these models are not designed as XML-aware clustering. According to the self-similarity principle of the fractal mathematical model, they can be considered for the development of a new clustering model. Can fractal outperform both simple cost metrics (Jaccard and VSM) and other standard (K-Means and PCA combined with K-Means) clustering models,

in terms of finding a high similarity degree of SOAP messages and the clustering time?

4. **What are the cost-effective methods to develop a distributed Web server-based aggregation model for large numbers (hundreds and thousands) of SOAP messages?**
   Aggregation of SOAP messages is an efficient solution to minimize network traffic. However, the previously proposed aggregation models are designed to aggregate only a few messages at the same Web server in one compact packet. The network is rapidly growing and large numbers of messages are sent and received simultaneously. However, the processing time of aggregating large numbers of messages is very long, and can become an expensive task. Technically, current aggregation models cannot be used for aggregating messages at several servers and share the service with a distributed technique. On the other hand, this costly aggregation task still requires high computations for selecting similar messages, which is another processing overhead.

1.3 Research contributions

In response to the research questions posted in Section 1.2, this thesis makes a number of contributions to the current state-of-the-art of research in improving the performance of the SOAP communication protocol in the context of Web services. The main contributions are:

1. **Binary and general tree-based structure compression techniques**
   New binary and general tree-based transformations are proposed and implemented for SOAP messages as an important technical part of new compression techniques. The tree-based transformed SOAP messages are encoded by either fixed length or variable (Huffman) encoding techniques. Technically, these transformations (binary and general tree) are found to be very supportive for compression as they remove all the duplicated closing tags of XML items and keep only one copy of each in the trans-

formed tree. Generally, developers deal with fixed and variable (Huffman) encoding techniques in compressing messages by using individual symbols or characters as the basic input parameters. However, in this research, the whole tags of SOAP messages are manipulated individually as basic parameters for both fixed- and variable-length techniques. The results are promising as they minimize the network traffic by reducing the size of sent/received messages over the network. The performance of the proposed techniques is found to be significantly better than other standard techniques, such as gzip, bzip2, XMill, and XbMill compression techniques. Furthermore, the compression/decompression times of these techniques are tested on personal computers (PCs), laptops, netbooks, and personal digital assistants (PDAs). Bandwidth-constrained mobile communication environments and Cloud Web services are two scenarios that are likely to benefit from the proposed compression techniques.

2. **Redundancy-aware aggregation and group-based similarity measurements**
New aggregation models are introduced that are mainly based on utilizing the compression concepts by exploiting the redundancies of SOAP messages. The basic objective of the proposed model is to provide an efficient aggregation that could potentially reduce the size of messages. XML tree compression-based aggregation techniques aim to enable Web servers to aggregate a group of messages that have a certain degree of similarity, and send them as one compact message in order to minimize the network traffic. Aggregated messages of SOAP responses are extractable at the closest routers to the receivers (clients) to deliver only the required response to clients. Furthermore, three similarity measurements of SOAP messages are introduced in order to investigate the highest similarity degree of SOAP message groups (i.e. not only pairs) to enable the aggregation techniques to potentially achieve significant message size reduction. Compressibility measurements, Jaccard coefficients, and VSM are developed to cluster SOAP messages based on their similarity. The results have proven that the size reduc-

tion by aggregating messages is significantly better than compressing them separately. Moreover, the proposed similarity measurements can significantly support aggregation to increase compression ratios.

3. **Dynamic fractal similarity-based clustering model**
   Fractal mathematical parameters are introduced as new efficient similarity measurements for SOAP messages, and developed as a new significant unsupervised auto-clustering technique. Fractal clustering would be an alternative to the SOAP similarity measurements for selecting the most similar messages by clustering them based on SOAP message fractal parameters. The similarity measurements are based on computing fractal coefficients of numeric objects that together construct one main numeric form. The generated dataset for SOAP messages is actually a set of numeric vectors showing the weights of XML items, which are broken up into blocks of equal size. Fractal coefficients of the vector blocks are computed and compared with others arranged in the same order in other vectors to investigate their fractal similarity and cluster them according to their similarities. Essentially, this model is mainly supporting the aggregation of SOAP messages with the aim to improve the compression ratio by grouping them according to a higher degree of similarity. Experimental evaluation has shown that the fractal clustering technique outperformed other standards (K-Means and PCA combined with K-Means) in terms of the resultant compression ratios and clustering time.

4. **Distributed aggregation and fast fractal similarity measurements**
   A novel distributed aggregation technique is proposed to aggregate messages at multi-Web servers located on the same path of the aggregated responses. Huffman binary tree encoding is developed to include more messages in the encoded format of the previously aggregated messages. The structure of the compact packet is modified to be adaptive for aggregating more messages at other Web servers. Furthermore, new mathematical

models are developed with the aim to speed up the performance of classical fractal computations. It is basically introducing new mathematical factors that are computed in advance of the fractal coefficient computations in order to segment data object blocks based on these factors. Fractal coefficients, such as scale, offset, and root mean square error (RMS), are computed for only those blocks that have the same fractal factor value. For clustering SOAP messages, fast fractal factors are computed in advance of the fractal coefficients that belong to the same segment. Finally, clustering SOAP messages are based on the maximum histograms of fast fractal indexes in every single message. The results have shown a tremendous minimization of the clustering time for the fast fractals in comparison with the classical fractals. Furthermore, fast fractal clustering models have outperformed K-means and PCA combined with K-means for large numbers (hundreds and thousands) of SOAP messages.

1.4 SOAP dataset

In fact, we have used more than one thousand and eight hundred SOAP messages for evaluating our proposed models in this thesis. Technically, we have used two datasets of SOAP messages: 160, and 1800 messages and that is to provide a suitable evaluation to investigate both the ability to reduce the network traffic and the processing time for regular compression, aggregation, and clustering in addition to the fast clustering and distributed aggregation.

The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth of the Web. SOAP messages are completely based on the Web Service Description Language (WSDL) schema as it represents one of the most used bindings provided by WSDL. All the considered messages in our evaluations are built based on the WSDL schema and the structure of the SOAP binding schema. The generation of SOAP messages was designed to create a variety of messages in terms the application (such as Travel Agent and Stock Quote Market applications) and size of messages. We consider different sizes: small (from only 140 byte to 800 bytes), medium (800-3000 bytes), large (3000-20000 bytes), and very large (20000-55000 bytes). WSDL schema and SOAP binding (both schemas published by W3C http://www.w3.org) are listed in Appendices A and B respectively.

1.5 Thesis structure

The rest of this thesis is structured as follows.

- Chapter 2 analyzes related work in Web services, specifically focuses on related studies that aim to improve the performance of Web services in some specific areas, such as compression, aggregation, clustering, caching, binary encoding, and run-time optimizations for SOAP engines. Moreover, limitation of existing solutions is presented.

- Chapter 3 presents three new compression techniques for SOAP messages. Binary tree and general tree structures are used to develop new textual expressions for XML-based messages. Fixed- and variable-length encoding are proposed to compress the generated textual expressions. The proposed techniques are evaluated and compared with existing compression models (gzip, bzip2, XMill, and XbMill).

- Chapter 4 presents a novel redundancy-aware aggregation model for aggregating SOAP messages. This chapter illustrates the development of the compression techniques presented in Chapter 3, and shows how to utilize their redundancy search strengths in developing a potential aggregation model that could achieve high reduction in the aggregated SOAP messages. Furthermore, traditional similarity measurements, such as Jaccard coefficients and VSM are developed and presented.

- Fractal similarity-based technique is presented in Chapter 5. Term Frequency with Inverse Document Frequency (TF-IDF) is proposed to build a numeric representation for XML documents in order to generate the dataset. The proposed clustering technique is evaluated and compared with other standard clustering techniques, such as K-means and PCA combined with K-means.

- A novel distributed aggregation technique that is aggregating SOAP messages at several Web servers over the network is presented in Chapter 6. Moreover, new development of the classical fractal model with the aim to provide significantly fast similarity measurements is presented. The proposed models are evaluated and promising results are obtained.
- Finally, Chapter 7 summarizes the research contributions presented in this thesis and discusses possible directions for future research.

Chapter 2

Related Work

This chapter considers some of the important existing solutions for developing the performance of Web services. This chapter starts by briefly exploring the general Web services architecture and layers to help understanding the strategies and goals of existing studies. Then, the strengths and weaknesses of the considered studies are examined. Section 2.1 explores the technical architecture and communication layers used to build Web services. Section 2.2 reviews the existing compression techniques proposed to compress Web messages in order to reduce the overall network traffic. Section 2.3 shows the proposed models for aggregating Web messages and the similarity measurements required by the aggregation models. Moreover, section 2.4 discusses existing clustering techniques used to cluster Web messages in order to illustrate their measurements and general structures. Section 2.5 considers the caching studies that aim to store the complete or partial copy of the sent/received Web messages and improve the response time by using the cached copies without generating them again. Section 2.6 illustrates the proposed models for the binary encoding of Web messages as an efficient alternative to the textual mode of Web services. Furthermore, this section presents few studies that have proposed serialization/deserialization improvements for XML-based messages. Section 2.7 reviews the run-time optimizations for SOAP engines.

Finally, section 2.8 illustrates the limitations of existing solutions.

2.1 Web services

In this section, the fundamental architecture, communication layer, and protocols used to build Web services are presented. Web services are XML (eXtensible Markup Language) based software systems designed and developed to be interoperable (interoperable machine to machine interaction) [Elfwing et al., 2002]. Web services interoperability is derived from a set of XML-based standard protocols such as WSDL (Web Service Description Language), SOAP (Simple Object Access Protocol), and UDDI (Universal Description, Discovery and Integration) [Davis and Parashar, 2002; Tian et al., 2003]. Web services are defined, located, and published by the common approaches that are introduced by these XML-based protocols [Liu and Deters, 2007; Tian et al., 2004].

2.1.1 Web service architecture

As a result of the dramatic increase in the number of Web services over the Internet, an emerging approach that can locate the required services has appeared. The Service Oriented Architecture (SOA) arranges locating desired services that could provide access to the targeted function or data [Liu and Deters, 2007]. SOA is a component model that defines Web services architecture [Elfwing et al., 2002]. The basic concept of SOA is to inter-relate its three components: service provider, service registry, and service requester with three operations: publish, find, and bind to provide automated discovery for services and maintain their use [Tian et al., 2003; Liu and Deters, 2007]. The interaction among the main three operations is depicted in Fig. 2.1.

These three main components can be described as the following:

- Service provider is responsible for publishing services to a registry, making them available on the network for the consumer's request.

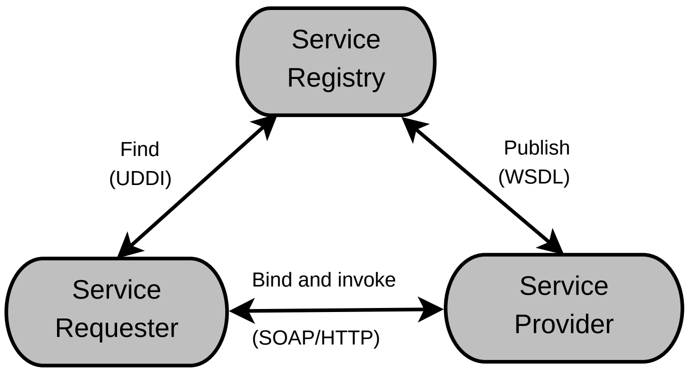


Figure 2.1: Web Service Architecture

- Service requester is responsible for finding a service description that is published to a service registry, as well as to bind or invoke these services hosted by a service provider.
- Service Registry supports the service provider and the service requester to find each other by replying to the service requester's queries on the availability of the desired service.

2.1.2 Web service layers

Web service layers is a collection of XML-based open protocols that support sophisticated communications between different nodes in a network [Elfwing et al., 2002; Tian et al., 2004]. Web service layers are placed between the application layer and the transport layer as shown in Fig. 2.2.

- Discover protocol organizes the Web service into a common registry. The service provider can use the Universal Description, Discovery and Integration (UDDI) specification to advertise the availability of Web services. The service requester can use the same UDDI specification to search and discover the desired services in a registry.
- Description protocol: Web service is defined by the Web Service Description Language (WSDL), the syntax of input and output documents, the communication protocol,

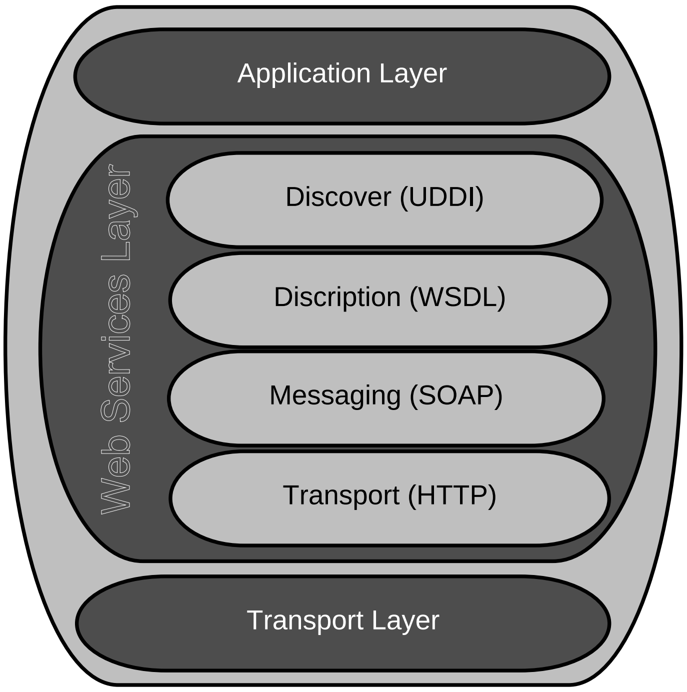


Figure 2.2: Web service protocol stack

and the location of these services. Moreover, WSDL specifies the public interface to a specific service.

- Messaging protocol: XML is the common format for encoding messages so that all nodes can understand each other. Simple Object Access Protocol (SOAP) is the standard format for exchanging services over HTTP.

- Transport protocol is responsible for transporting Web service messages between various applications over the network. Hyper Text Transport Protocol (HTTP) is the main transport protocol for Web services as a result of its capability to pass through firewalls. However, there are some other transport protocols such as TCP, UDP, SMPT and FTP that could be used instead.

2.2 Compression techniques

Compression is a popular strategy to reduce the overall size of large Web service messages. It is proposed by several studies to manage the poor connectivity for clients with resource constrained devices or resource constrained environments (e.g wireless environment).

Liefke in 2000 [Liefke and Suciu, 2000b] designed and implemented a compressor (XMill) and a decompressor (XDemill) for XML data. This technique was based on three steps. First step involves separating XML tags from the data items, and therefore, XML tree structure and groups of data items are compressed separately. Next, it distributes the items into separate containers with related meaning, where each container is compressed separately by exploiting similarities of the considered data values. Finally, some semantic compressors are applied to each container. The XMill method achieves a good performance and can compress messages twice more than what gzip can achieve. For messages that had more data and less text, XMill without any semantic compressor reduced them about 45%-60% of gzip, while with semantic compressors the resultant size is about 35%-47% of gzip's.

Another research by Werner in 2004 [Werner and Buschmann, 2004] evaluated the performance of gzip and bzip2 compressors by comparing them with three XML compressors (XMILL, xmlppm and XBXML). In this study, a test bed is set up with 182 files of eight different methods (XML(uncompressed), XML(bzip2), XML(gzip), XMILL(bzip2), XMILL(gzip), XMILL(ppm), xmlppm and wbxml). The evaluation shows gzip compression is more effective than bzip2 in achieving better compression ratio but XMILL(ppm) outperforms both. Moreover, in order to reduce the compression overhead, a differential encoding for both web requests and responses has been suggested. This was achieved by encoding only the differences between the current message and the previous sent/received by a web service. A skeleton for the previous message is generated to compute the differences with the current message.

In 2006, Werner [Werner et al., 2006] introduced a new approach using generated single deterministic pushdown automaton (PDA), that can represent the XML schema. The proposed algorithm first converts the XML schema into Regular Tree Grammar (RTG), then converts RTG into a set FSA (Finite State Automaton). Finally, the considered algorithm constructs the pushdown automaton that is based on the RTG. The compression scheme of this research is based on the conversion of XML document as a sequence of transitions on the path of PDA. The proposed algorithm achieves compression ratios between 5 and 10 for large XML documents.

Catalin Rosu in 2007 [Rosu, 2007] proposed an XML dictionary compression schema that enables both sender and receiver to build the same dictionary for XML tags. On the sending endpoint, new tags are inserted in the dictionary and replaced by their indexes. On the receiving endpoint, newly received tags are inserted in the dictionary with the same indexes used at the sender side. Practically, any reused tag in the send/receive messages will be replaced with its corresponding index.

Performance evaluation study for XML compression techniques was proposed by Ericsson in 2007 [Ericsson, 2007] and measured their effects on SOAP Web services. This research was based on testing all the considered techniques on two platforms: desktop computers and different small wireless devices. These platforms were involved in this study to investigate the performance of XML compression techniques in different real-world scenarios including the low bandwidth environments. Five encodings (BXML, XMill, XMLPPM, Gzip, and bzip2) were investigated by this study. While testing on small wireless devices, performance of BXML coupled with Gzip was specified as the best in comparison with others, for all documents. For document size ranging from 2-300 KB, XBMill and XMill outperformed all other encodings and BXML was the worst performer. Moreover, all encodings except BXML achieved almost the same compression ratio for document size ranging from 0.7-130 KB. For documents ranging from 0.2-9MB, the resultant compression ratio for all encodings were almost the same, except for BXML which is considerably worse.

Luoma in 2007 [Luoma and Teuhola, 2007] presented a novel modeling technique that is based on traversing XML tree of the document, and adding an empty tag to each node in the whole XML tree. The objective of the empty tag is to preserve the nested structure of the tree, so XML tree can be regenerated again. Preorder and level-order are used to traverse the XML tree. Different XML tree models are discussed and evaluated such as children-depth-first, ancestor-path, and left-sibling-parent techniques. In addition, the proposed techniques are evaluated and compared with the performance of gzip and XMill compressors. XMill coupled with gzip outperform both level-order and preorder methods. However, when XMill is coupled with bzip2, preorder technique slightly outperforms both level-order and XMill.

A discussion on how the proposed preprocessing transformation can improve the performance of generic compression techniques, in terms of getting better compression ratios than XML-aware encodings, is presented by Skibinski in 2007 [Skibinski et al., 2007]. The first preprocessing in this work requires removing single spaces that are located before the encoded words, and generating them again during the decompression process. The second step of the proposed transformation is replacing every sequence of digits with two adjacent codes. The first code is a character that shows the length of the second code, in terms of number of consumed bytes. The digit sequence is represented by the second code using base-256 numerical representations. The test shows that PPM efficiency has been improved using the proposed textual transformation. Based on this work, gzip improves its compression ratio by about 30% and LZMA by about 41%. LZMA as a generic encoding technique outperformed the gzip based XMill compression ratio by about 27%.

Johnsrud in 2008 [Johnsrud et al., 2008] evaluated compression methods on resource constrained mobile platforms with limited processing power. Measurements have been performed in both simulated environments and on wireless mobile devices. The evaluation differentiates between generic and XML-aware compression techniques. As an XML-aware encoding, XMLPPM achieved the best compression ratios in comparison with other XML compressors. On the other hand, gzip achieves the best compression ratios in comparison with other generic techniques. The study shows that XMLPPM requires a high amount of processing power and memory in comparison with gzip. Therefore, XMLPPM is not a promising encoding method in mobile networks with limited resources. Moreover, ZLIB and EFX are considered in this study as generic and XML encodings respectively. The response time for both techniques are almost the same, with EFX being slightly better. Both encoding methods considerably reduce the overall response time of large samples.

Cezar Estrella in 2008 [Julio Cezar Estrella and Monaco, September 22-24, 2008] discussed how the compression techniques can decrease the time for data transfer over the network by improving the response time and network traffic volume. Then, they introduced a heuristic to provide preliminary information about implementing compression for Web services, and to decide whether the SOAP message should be compressed or not. The experimental simulation shows that the proposed heuristic can improve the service response time based on the model scenario.

2.3 Aggregation and similarity measurements

Similarity-based aggregation models for XML-based Web messages have been proposed by few studies, which aim to reduce the overall network traffic by exploiting Web message similarities. Furthermore, similarity measurements have been examined and proposed by several studies.

Leung in [Leung et al., 2005] introduced a new sequential mining-based XML documents similarity computation in order to develop a novel technique for finding the semantic correspondence of XML documents. The proposed sequential mining scheme uses preorder traversal to traverse XML trees of XML documents in order to extract more hierarchical information, such as the paths and positions of XML items. A postprocessing step is proposed for reutilizing the mined patterns of XML items with the aim to investigate the similarity of unmatched elements, and produce another metric for similarity measurements. The semantics of XML elements are estimated by considering their subtree or leaf nodes. The experimental results show that the proposed technique provides more stable and reasonable similarity measurements for XML documents.

Another similarity measurements study by Flesca in [Flesca et al., 2005] introduced a new similarity measurements approach for XML messages. It is mainly investigating the structural similarities of XML documents in the generated time series representation. The basic strategy of the proposed technique consists of linearizing the structure of XML documents by encoding XML tags into signal pulses in order to transform XML document into a numeric form. Then, the XML documents are distributed into clusters according to the analysis of their numerical sequences. Discrete Fourier Transform (DFT) is proposed to effectively compare the encoded XML documents (frequency domain). The experimental results show the effectiveness of the proposed technique in comparison with tree-edit function based techniques.

Ma and Chbeir [Ma and Chbeir, 2005] have addressed the problem of XML documents similarity measurements by considering the asymmetric similarity in addition to both semantic content and the XML document structure. In this research, XML schema was investigated as XML documents may have the same schema. However, they may have different structure like different number of occurrences for the same element. Furthermore, XML tree structure has been addressed with ascending and descending weight for branch from bottom to up and from up to bottom respectively. Several similarity measurements have been considered such as instance similarity of simple and complex elements. A Java based prototype called SimXML was implemented in order to validate the proposed approach.

A hybrid approach for XML similarity has been proposed by Tekli in 2007 [Tekli et al., 2007]. Their approach integrates information retrieval (IR) semantic similarity in the traditional edit distance function. The hybrid model has introduced the semantic relatedness of

XML elements/attribute labels in edit distance computations. With the aim to evaluate the proposed similarity measurement technique, real and generated XML documents in addition to a number of hierarchical taxonomies are considered. Moreover, the results have shown a positive impact of the semantic meaning on improving the similarity measurements for XML documents.

XML-aware aggregation model for SOAP Web messages after computing their similarities using Jaccard and Levenshtein similarity measurements is developed by Phan in 2008 [Phan et al., 2008]. The aggregated messages are delivered using multicast protocol in order to avoid sending the SOAP responses separately. This efficiently minimizes the network traffic. The generated compact message includes all the addresses of clients as strings in the header part. The structure of the aggregated messages consist of two parts: the common section that contains message structures and common values of the messages, and the distinctive section that contains the non-redundant values of the messages. Intermediary routers parse the message header and create groups of client addresses based on the next hop in order to forward only the required message along the next hop.

Wang in [Wang et al., 2009] introduced a novel approach for similarity measurement for XML documents. The main technique is based on a weighted element tree model. The proposed model classifies XML elements as the center, subtrees as main parts, and the weight of the subtrees that represents the connection among elements. Basically, similarity of XML documents is computed with respect to common XML tree features that have been proposed in this research. Moreover, the proposed technique was evaluated using two different datasets, and compared with the tree edit distance algorithm. The performance of all models was investigated with respect to two metrics: the processing cost in terms of the processing time and accuracy of the similarity measurements for XML documents.

2.4 Clustering models

Clustering of XML-based Web messages is proposed by many studies with the aim to facilitate several advanced Web applications such as information retrieval, data integration, structure analysis for Web messages, and documents classification. Liu in [Liu et al., 2004] developed a new XML clustering approach using Principal Component Analysis (PCA) technique. The proposed technique first extracts features from XML documents by constructing an ordered labeled XML tree and transform them into vectors. The resulted vectors contain the occurrences of the considered features in XML documents. PCA is developed to minimize the dimensions of the dataset vectors by summarizing all of the considered features and generating new reduced dimension vectors. Then, the K-Means clustering technique is used to cluster XML documents based on the minimized features. In order to evaluate the performance of the proposed approach, two sets of XML documents are considered as input datasets. The performance of the developed PCA technique is compared with K-Means technique without reducing the dimensionality of the dataset vectors. The experiments show that the PCA has significantly improved the accuracy of K-Means clustering.

Lian [Lian et al., 2004] has proposed a hierarchical based algorithm (S-GRACE) for clustering XML documents using the structural similarities of XML trees. The study discussed that a group of XML documents could have different structures while an appropriate clustering technique alleviates the fragmentation problem. The proposed clustering algorithm is based on a distance metric, which is developed on the graph structure notation in order to provide a minimal summary of content in the considered documents. Furthermore, with the aim to investigate the performance of the proposed clustering technique, DBLP database was used as a dataset and the proposed model showed a higher performance speed-up in comparison with other techniques.

Particle Swarm Optimization (PSO), as a fast and high quality clustering algorithm for text documents, was proposed by Cui [Cui et al., 2005]. In this research, PSO was implemented in addition to K-Means clustering technique, in order to produce an accurate comparison for the performance of the proposed technique with a well-known clustering method. The dataset was generated using the Vector Space Model (VSM). Although experiments showed that PSO is significantly better than K-Means in terms of the clustering time, K-Means is still more efficient for large documents than PSO. Therefore, a hybrid PSO was presented in this study that can take the advantage of K-Means to replace the refine stage in the proposed PSO technique. K-Means, PSO and hybrid PSO techniques have been applied on four generated datasets with different number of documents for each one. The hybrid PSO showed the best results in comparison with other clustering techniques.

Another XML documents clustering technique by considering the weight of frequent structures in XML trees was proposed by Hwang and Gu [Hwang and Gu, 2007]. The strategy of the proposed technique was completely based on the ability to recognize the highly frequent items in XML documents in order to cluster them according to these items. The criterion for clustering XML documents include the path details of all XML tags and data items. Technically, the proposed technique first computes the average frequencies of the structures in XML tree. Second, it groups any new XML document according to the same average factor. With the aim to evaluate the performance of the developed technique, a comparison was considered with the Hierarchical Agglomerative Clustering (HAC) and K-Means techniques. The experiments showed that the proposed technique has outperformed both HAC and K-Means techniques.

Yongming [Yongming et al., 2008] introduced a novel technique for the measurements of XML similarities, and then cluster them based on both the structure and content. The contribution of this research is a new development to the traditional Vector Space Model (VSM) by adding the structural similarity measurements as the main technical step in the clustering process of XML documents. The leaf path and nested elements are the main XML features that are extracted in order to build the dataset. Moreover, VSM is technically based on computing the weight of XML tags and data elements as the features for clustering purposes. In order to evaluate the proposed techniques, entropy and purity have been computed for two different datasets. The results have shown that the new developed vector space model outperformed the traditional version as it has shown higher purity and lower entropy.

Homogeneous XML documents clustering technique was implemented by Nagwani and Bhansali [Nagwani and Bhansali, 2010] using weighted similarity measurements on the XML attributes. A new distance measurement methodology was proposed for XML documents. They have implemented the proposed technique using several open source technologies such as Java, JIXB, and JAXP. The processing of the proposed technique starts by retrieving XML documents from the repositories. Then, all the retrieved XML documents are parsed and the required information will be kept in the Java collections (API). Next, the similarity measurements are applied based on the structure and style-sheet in addition to the content of XML documents. Finally, K-means is applied to cluster the documents using the resultant measurements. The Wikipedia XML Corpus is used as a dataset for the experimental results.

2.5 Caching systems

Several caching strategies are proposed by several studies [Devaram and Andresen, June 23 - 26, 2003; Terry and Ramasubramanian, 2003; Liu and Deters, 2007] to improve the performance of Web services. Mainly, caching supports disconnected operations as it enables clients and servers to retrieve disconnected requests and responses. Partial caching model was proposed by Devaram and Andersen [Devaram and Andresen, June 23 - 26, 2003] to cache SOAP messages at the client side. All SOAP messages are cached at the client when they are first sent. The cached payloads of SOAP messages are reused to generate future requests, and only the new parameters/values are replaced. The experimental results show higher performance in comparison with the non-caching system (conventional SOAP messaging), and specially for small number of XML tags and values. On the other hand, the proposed partial caching strategy degrades with large number of XML tags and values as a result of the large number of accessing file (i.e. I/O operations).

Another caching strategy was proposed by Terry and Ramasubramanian [Terry and Ramasubramanian, 2003] which is based on developing a HTTP proxy server between the Web service provider and the consumer, in order to cache the Web SOAP messages at the proxy server. The proposed proxy caching model supports the disconnected operations. This study discusses the benefits that could be provided on the server side. Technically, for the disconnected operations, the proxy machine sends a copy of the previously stored responses to the client. Furthermore, all the received requests from the clients are stored in a write back queue and would be sent to the server in case the request was sent during the disconnected mode. However, the study has highlighted several technical issues such as consistency and availability of offline access to Web requests/responses. Moreover, the proposed model suffers from recognizing the required service to be played back.

A dual caching strategy for mobile Web services was proposed by Liu and Deters [Liu and Deters, 2007]. The proposed strategy is mainly to resolve the loss of connectivity problem between the server and client. The caching model resides on both sides (server and client). A cache manager is proposed to arrange the coordination between the service provider and consumer. The meta-data is described using an ontology Web language on both caching sides including client workflow, service description, and disconnectivity description. With the aim to develop the caching strategy, new annotations are proposed in the WSDL specification to express some semantic information such as cacheability, life time, and default response.

2.6 Binary encodings and Serialization/Deserialization models

With the aim to reduce the size of SOAP messages, binary encodings have been proposed by several studies. SOAP messages are transmitted using a binary format (similar to CORBA) instead of the textual format. Technically, both textual and binary XML schemes are supported by SOAP engines.

SOAP MTOM (Message Optimization Mechanism) [Gudgin et al., January 25, 2005a] was introduced in a new released specifications by W3C XML Protocol Working Group and XML binary Optimized Packaging (XOP) [Gudgin et al., January 25, 2005b]. The target of these specifications was to facilitate the communication for multimedia data such as BMP, JPEG, and GIF as well as data that has digital signature. Moreover, these specifications represent a technical definition of the XML Infoset serialization. An extensible packaging format includes the serialization of XML Infoset has created the XOP. The mechanism of XOP and how it is layered into SOAP HTTP transport is described by MTOM.

A new mobile Web service architecture called Handheld Flexible Representation (HHFR) was proposed by Oh and Fox [Oh and Fox, 2005]. Their proposed architecture is supported by a binary messaging stream in order to provide optimized SOAP messaging communication. HHFR architecture is mainly based on separating the XML syntax of SOAP messages from the contents of the same messages. The syntax of SOAP body is characterized by an XML schema at the initiation of the message stream to achieve the separation process. As a consequence to the fact that sent/received SOAP messages between any two end-points have similar structure and possibly similar content, the proposed architecture (HHFR) is technically suited for end-points using the same Web application. SOAP messages structure and type are transferred only once and only the payloads are transmitted for the future transactions. The experimental results show higher performance to the SOAP messaging in comparison with other conventional architectures.

Lu [Lu et al., 2006] has built a new binary encoding scheme for XML documents called BXSA (Binary XML for Scientific Applications). The proposed BXSA encoding supports both converting the textual form of XML into a binary XML format and vice versa. SOAP messages are first modeled using bXDM model (an XML data model for scientific data that is developed by the same authors) instead of the XML Infoset. After modeling SOAP messages using the bXDM model, the encoding policy provider is invoked and the message is serialized into an octet stream. Then, the generated octet stream is transferred by the policy provider. On the other hand, a reverse processing is used when a message is received. Experimental results showed that both SOAP over BXSA/TCP model and SOAP with HTTP data channel have similar performance. BXSA transport can be bound to multiple TCP streams in order to carry larger messages.

Differential encoding is another efficient solution proposed by several studies to utilize the fact of having similar SOAP messages in content. Differences of SOAP messages are the only parts sent over the network. Abu-Ghazaleh [Abu-Ghazaleh et al., 2004] developed a differential serialization model at the server side with the aim to optimize SOAP performance. SOAP messages are serialized only in the first time as the serialized form of SOAP messages are stored at the server. Later on, the proposed model serializes only the different elements that have changed or not been serialized before, and reuses the main serialized parts of the previous messages. Furthermore, the evaluation shows higher performance to the proposed model in terms of the sending time by a factor of four to ten for Web applications that resend the same messages repeatedly. Moreover, significant improvement was noticed for resending SOAP messages with similar structure and have different values.

Another study by Suzumura et al. 2005 [Suzumura et al., 2005] proposed a differential deserialization strategy for SOAP messages with similar structure. First, a new XML parser was proposed in [Takase et al., 2005] that supports the differential deserialization process by recognizing the differential regions between the current active XML message and the previous messages. Technically, only the differential portions are parsed. The transitions state is stored by a state machine during the parsing process of XML messages. This strategy is developed to build a deserialization framework in the Web service architecture. The experimental results show a significant optimization to the processing time at the receiver side.

2.7 Run-time optimizations

Run-time optimizations are proposed by few studies in order to develop real time performance for SOAP engines. Helander and Sigurdsson [Helander and Sigurdsson, 2005] introduced a stochastic quality sampling driven scheduler and pre-planned matching mechanism to achieve a real-time improvement to SOAP Web services. This study proposes a self tuning planning mechanism that predicts the required resources for the requested SOAP services by using online sampling driven statistical technique. Technically, the proposed model separates the temporal behavior (temporal behavior is represented by patterns that describe the required resources for each node involved in the requested task) from the actual functions. The required resources are described in a high-level language using the XML syntax. Several parameters (time constraints) related to the service are provided in advance to determine the execution time. Furthermore, execution and control flows are separated, which a continuation rendezvous mechanism combines them later. The evaluation part shows that the proposed model can be applied in a micro controller with 256KB of ROM and 32KB of RAM.

Another study by Jun [Jun et al., 2006] developed a real-time improvement to the performance of SOAP engine by proposing a new data mapping template. The study identified the main negative impact factor on the performance of SOAP Web services and that is the data model mapping between XML data and Java data. Therefore, a new data mapping model was proposed (Dynamic Early Binding) that avoids Java reflections by reserving the mapping data and actions in a template that is dynamically generated. Furthermore, push-down automation with output actions is used to develop a context-free grammar template. The proposed mapping template was applied on SOAP engine, and showed significant improvement to the SOAP performance in comparison with Apache Axis 1.2.

Gamini Abhaya [Gamini Abhaya et al., 2010] presented a set of guidelines, algorithms, and techniques for building a Web service with predictable service execution. The proposed guidelines can be classified into three requirements. First, in order to meet SOAP requests deadline, they should be explicitly scheduled. Second, SOAP requests are based on requests laxity. Finally, the proposed model needs to be supported with an operating system that provides the access to system features for achieving predictability. The study discussed the required enhancement to standalone SOAP engine to achieve service predictability and improve the service response time. The proposed guidelines were evaluated and proved the ability to achieve service predictability by using the proper techniques in SOAP engines.

2.8 Limitations of existing solutions

Although the proposed studies have improved the performance of Web services, they missed some significant links to achieve higher performance.

- Most of the XML-aware compression techniques are still using some generic compression techniques such as gzip and bzip2 in specific portions of the compact message. This strategy reduces the opportunity to increase compression ratio of messages.
- Technically, aggregation of SOAP messages is based on reducing the overall size of the aggregated messages by exploiting the similarities inside them. However, the existing techniques do not use the potential of redundancy encoding techniques (compression). This could potentially result in higher size reduction.
- Clustering techniques are generally based on iterative processing which is a time consuming strategy. Therefore, a one pass clustering technique is a requirement for improving the clustering time of Web messages.
- The proposed caching strategies degrade with large sized Web messages. Moreover, although caching is resolving the disconnectivity problem for clients and servers, it suffers from inconsistency and the recognition of available offline access.

- Binary encoding solution is generally affecting the interoperability of Web services, which is a significant goal that Web services try to achieve. Moreover, binary encoding is proposed to reduce the size of Web messages, where the potential of compression concepts in reducing message size was not used.

- Run-time optimizations require significant support from operating systems to meet the suggested guidelines to improve the real time performance of SOAP engines. This technical obstacle may result in expensive requirements for the proposed run-time solution.

Chapter 3

XML-aware Compression Techniques for SOAP Messages

This chapter answers the first research question posted in section 1.2. The main performance problem of SOAP communication protocol is outlined. Furthermore, the proposed compression techniques as an efficient solution in terms of minimizing the required bandwidth over the Internet for SOAP requests and responses is presented. Section 3.1 describes the problem outlines and its negative impact on the network. Then, section 3.2 shows the motivation for this research with the drawbacks of existing compression standards. In section 3.3, we have described the technical structure of the proposed compression techniques. Section 3.4 shows the analysis and evaluation of the proposed techniques with the comparison with other standards. Finally, section 3.5 summarizes the chapter.

3.1 The problem statement

As a result of the dramatic increase in the number of Web services over the Internet, an emerging approach called Service Oriented Architecture (SOA) has appeared [Davis and Zhang, 2002] that is capable of locating desired services and providing access to the targeted function or data. It is a component based model that defines the Web services Architecture [Wang et al., 2006]. The protocol behind the success of SOA is Simple Object Access Protocol (SOAP) which is a communication technology that supports interoperability by enabling applications to communicate with each other from the same or different platforms [Davis and Parashar, 2002]. XML (Extensible and Markup Language) is an encoding scheme for SOAP Web services to translate requests/responses and send them over HTTP transport layer [Heinzl et al., 2006]. As a result of SOAP Web service features such as interoperability, self-describing nature of the XML as an encoding language, and HTTP being used as a transport layer protocol, usage of Web services has increased dramatically over the Internet [Andresen et al., 2004; Abu-Ghazaleh and Lewis, 2005], and will likely be the protocol for massive data transactions in the emerging mobile, cloud and cluster environments [Cardellini et al., 2002; Levy et al., 2003; Yanping et al., 2005; Brebner and Liu, 2011; Dikaiakos et al., 2009; AjayKumar et al., 2009].

However, SOAP Web services require more bandwidth compared with other communication technologies such as Java-RMI and CORBA [Werner and Buschmann, 2004]. The major reason of this drawback is the use of XML as an encoding language which is verbose in nature and has high overhead [Julio Cezar Estrella and Monaco, September 22-24, 2008]. Consequently, as a result of the high network traffic and the significant increase of users demand for Web services, many organizations may suffer congestion, bottlenecks, and degraded performance of Web applications.

As an example of high network volume, medical Web scenarios have shown high SOAP message loads over the Internet. Figures 3.1 and 3.2 are both medical scenarios that show large number of patients and costumers daily transactions over the Internet. Furthermore, Fig. 3.3 represents only a small part of a large size patients report that usually medical organizations exchange with each other periodically. Figure 3.4 shows the structure of the large size patients report.

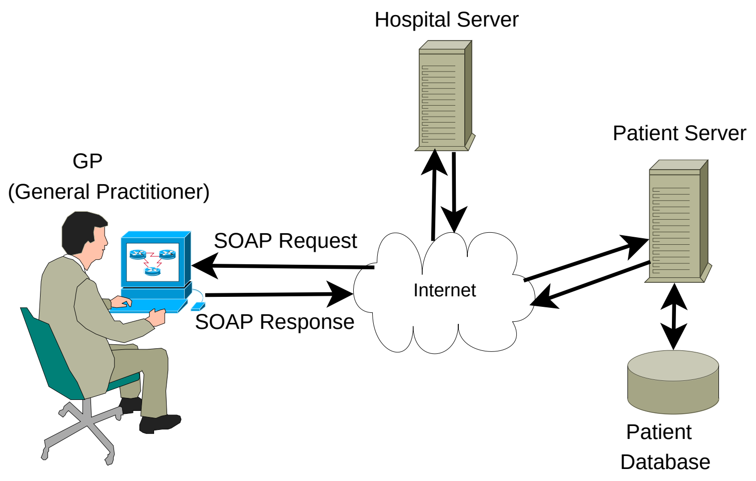


Figure 3.1: Medical SOAP Web service scenario

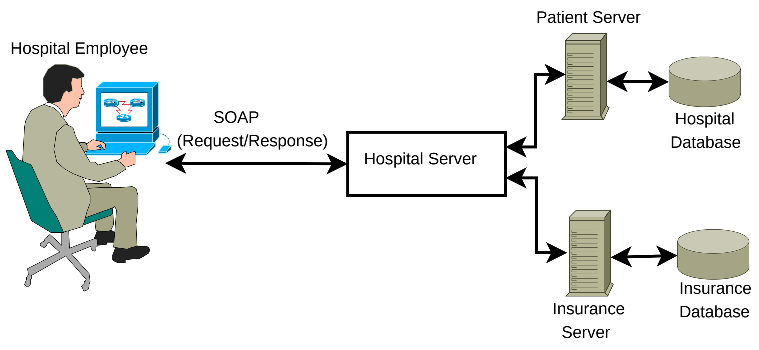


Figure 3.2: Patients medical Web services insurance scenario

3.2 Motivation

Compression of SOAP has been suggested as an efficient mechanism to reduce bandwidth requirement and improve performance of Web services by a number of researchers using different concepts and strategies. These concepts have exploited the strategy of advocating the segregation of XML tags from data items and compressing them separately using semantic compressors [Liefke and Suciu, 2000a]. Moreover, traversing XML tree [Luoma and Teuhola, 2007] is another strategy of Web service compression [Skibinski et al., 2007] that requires pre-processing steps for Web service messages in order to enable the compressors to achieve high compression ratio.

```
< Clinic_Data >
< HSD_Organisation_ID >
24811 < /HSD_Organisation_ID >
< Clinic_Name >
XxxxXxxx < /Clinic_Name >
< Clinic_Address_1 >
54 Xxxxx Street
< /Clinic_Address_1 >
< Clinic_Address_2/ >
< Clinic_Address_3/ >
< Suburb > Pascoe Vale < /Suburb >
< State > VIC < /State >
< Postcode > 3144 < /Postcode >
< Clinic_Email/ > xxx@xxx.xxx < /Clinic_Email >
< Telephone_List >
< Telephone >
< Type > Voice/LandLine < /Type >
< Area_Code > 03 < /Area_Code >
< Number > xxxxxxxx < /Number >
< /Telephone >
< Telephone >
< Type > Fax < /Type >
< Area_Code > 03 < /Area_Code >
< Number > xxxxxxxx < /Number >
< /Telephone >
< /Telephone_List >
< Practice_Principals_ID >
24811 < /Practice_Principals_ID >
< Practice_Principals_Name >
XxxxXxxx < /Practice_Principals_Name >
< Last_Updated >
21/09/200911 : 05 : 41AM < /Last_Updated >
< Status > A < /Status >
< /Clinic_Data >
```

Figure 3.3: Single patient report message

Although previous studies have achieved some performance improvements to the Web services, they still have limited performance and drawbacks as:

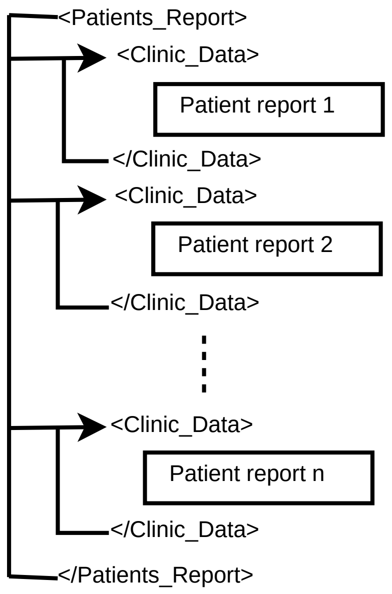


Figure 3.4: Web message structure of bulk patients report

- Most of the proposed techniques could not achieve a very high compression ratio because they are basically based on generic compressors such as gzip and bzip2.
- Web service scenarios such as mobile and Cloud Web applications are not significantly supported by these techniques in reducing the required bandwidth.
- The compression and decompression time of some of them is considered to be significantly high and unsuitable for devices with limited resources (e.g Netbooks and PDAs) over the Internet.

Although lossy compression mechanisms can achieve potentially high compression ratios, web applications cannot tolerate any loss as Web service messages must be completely reconstructable in order to be meaningful to the applications. Therefore, the challenge here is to develop compression techniques that can achieve high compression ratio and at the same time can keep their lossless feature to guarantee the translation of compressed messages to the original ones.

Both Cloud Web service scenarios (see Fig. 3.5) and bandwidth constrained mobile communication environment (see Fig. 3.6) are two of the most Web services based scenarios and applications that are likely to benefit from the proposed compression techniques.

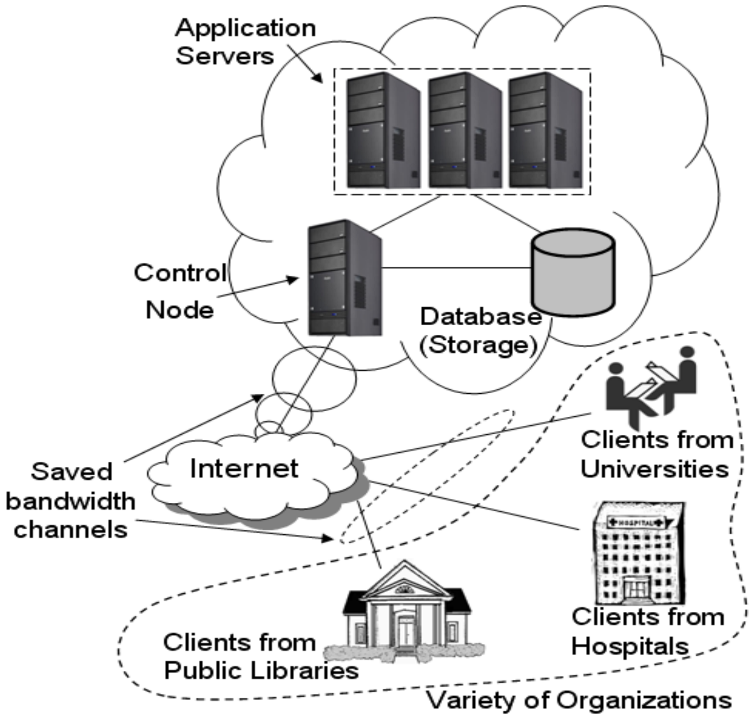


Figure 3.5: Compression supports of Web services: Cloud Web services environment

3.3 The proposed compression models

In this chapter, three XML-aware compression techniques are proposed that are technically based on three novel assignment mechanisms for XML tree of SOAP messages. The proposed assignment mechanisms aim to remove all the duplicated XML tags as XML messages have high degree of redundancy. Binary tree transformation is proposed to the general XML tree using first child/first sibling method. Binary tree based, pre-order, and level-order tree traversals are developed to traverse XML tree and assign the resultant order with combinations of either two or one bit codes. Then, the resultant assigned XML trees are encoded

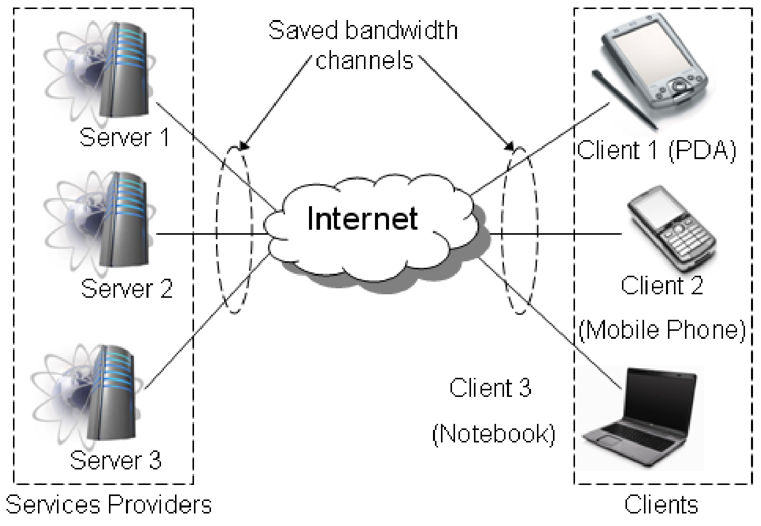


Figure 3.6: Compression supports of Web services: low bandwidth Web services environment

by either fixed-length or Huffman (variable-length) techniques in order to reduce the overall size of SOAP messages.

Fixed-length and Huffman encodings are developed to encode XML tags and data leaf as individual input parameters since both encodings generate bit code strings for all tags and data leaf. In both encodings, instead of computing a lookup table that has a bit code string for every individual symbol in the XML message, the lookup would include a bit code string for every XML tag and data leaf as separate items. Figure 3.7 shows the main technical components of the Binary-tree compression techniques. Figure 3.8 shows the main technical components of both the proposed One-bit and Two-bit status tree compression techniques for SOAP messages.

The gains resulted from the proposed techniques are promising and show higher performance in terms of high compression ratios when they compared with other well known XML-aware encodings such as XMill and XBMill as well as generic compression techniques like gzip and bzip2. Furthermore, the processing time for both compression and decompression are found to be significantly low on PC. Moreover, the general XML tree based compression techniques have shown significant processing time on limited performance de-

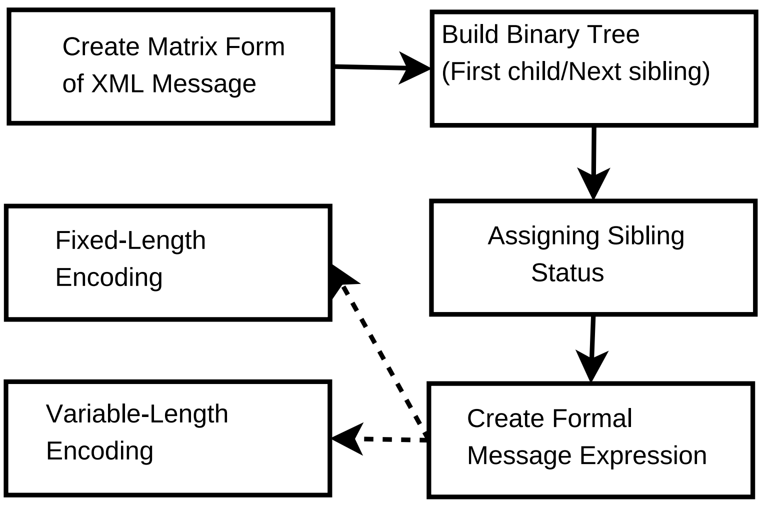

Create Matrix Form of XML Message
Build Binary Tree (First child/Next sibling)
Assigning Sibling Status
Create Formal Message Expression
Fixed-Length Encoding
Variable-Length Encoding

Figure 3.7: Binary-tree main model components

vices such as Laptop, Netbook, and PDA.

3.3.1 Computing XML tree

Generally, compression is one of the major performance enhancement techniques that can efficiently reduce network traffic of Web service applications over the Internet [Julio Cezar Estrella and Monaco, September 22-24, 2008]. Technically, compression improve the performance of Web services and protocols by supporting end users to send and receive compact size of Web messages. Basically, Web service messages are based on tree structure which is a serious motivation for utilizing tree data structure in developing the core of the proposed compression techniques. These models enable both clients and servers to send compressed messages and have the ability to decompress the received requests or responses. All the proposed techniques are based on assigning XML tree and then encoding it using variable and fixed length compression techniques. A stock quote application is used as an illustrative example to show the practical steps of the proposed compression techniques. SOAP message given in Fig. 3.9 is a response to the operation getStockQuote(WBC, ANZ). Generally, stock quote applications involve a large number of transactions requesting the available market quotes based on the requirements of the clients.

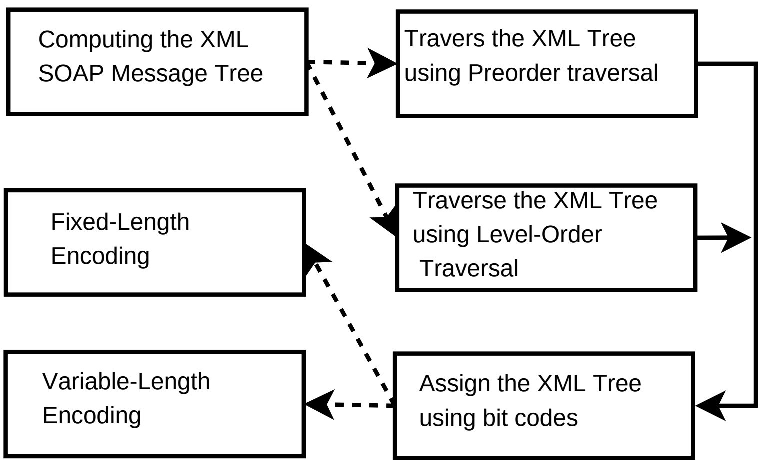

Computing the XML SOAP Message Tree
Travers the XML Tree using Preorder traversal
Fixed-Length Encoding
Traverse the XML Tree using Level-Order Traversal
Variable-Length Encoding
Assign the XML Tree using bit codes

Figure 3.8: One-bit and Two-bit status tree main models components

$< StockQuoteResponse >$
$< ArrayOfStockQuote >$
$< StockQuote >$
$< Company >$ WBC $< /Company >$
$< QuoteInfo >$
$< Price >$ 24.74 $< /Price >$
$< LastUpdated >$ 22/01/2010 $< /LastUpdated >$
$< /QuoteInfo >$
$< /StockQuote >$
$< StockQuote >$
$< Company >$ ANZ $< /Company >$
$< QuoteInfo >$
$< Price >$ 21.1 $< /Price >$
$< LastUpdated >$ 22/01/2010 $< /LastUpdated >$
$< /QuoteInfo >$
$< /StockQuote >$
$< /ArrayOfStockQuote >$
$< /StockQuoteResponse >$

Figure 3.9: A SOAP response message to the getStockQuote(WBC, ANZ) request

Building the XML tree is the initial step of all the proposed compression models which is preparing XML message for the encoding process. XML tree (denoted as T) is a finite set of one or more nodes such that R is called the root of XML tree including both types of nodes: leaf and non-leaf nodes. Furthermore, XML items of SOAP messages can be classified into simple and complex nodes as they defined in definitions 3.2 and 3.3.

The nodes of XML tree except the root node (R_0) are partitioned into $K \geq 0$ number of disjoint subsets $T_1, T_2, ..., T_K$, each of which is a subtree that has its own root $R_1, R_2, ..., R_K$, respectively, and at the same time they are children of the main root R_0 of the considered XML tree. Considering all the subtrees, there are $N \geq 0$ complex nodes $(X_1, X_2, ..., X_N)$ in the whole XML tree. In addition, the last two levels of each subtree (T_i), there is a number of simple nodes that are $M \geq 0$ such that $M \geq 0$ $(S_1, S_2, ..., S_M)$ in all the subtrees of the XML tree.

Definition 3.1 A SOAP message tree (denoted as T) is a finite set of one or more complex nodes X_i where $i \leq N - 1$ and one or more simple nodes S_j where $j \leq M - 1$. N and M are numbers of complex and simple nodes respectively. R_0 is considered to be the root of XML tree and each complex node (R_i) is a root of a subtree of XML tree $T\{R_0\}$

Definition 3.2 Simple nodes $\{S\}$ is a set of data leaf elements and the parent nodes of all data leaf elements.

Definition 3.3 Complex nodes $\{X\}$ is a set of elements $X_i \notin \{S\}$

Based on definitions 3.1, 3.2, and 3.3, XML tree can be represented by the following formula:

$$T\{X_0\} = \{X_0, X_1, ..., X_{N-1}\} \cup \{S_0, S_1, ..., S_{M-1}\} \tag{3.1}$$

Technically, computing the matrix form of the XML messages that is based on transformation of XML string into a general XML tree has resulted in reducing the overall number of XML tags. For every non-leaf tag, there is a closing tag and the matrix form of XML tree has one tag occurrence of every two occurrences of non-leaf tag. The overall number of tags (T_{Nodes}) can then be computed as:

$$T_{Nodes} = C \times 2 + L \tag{3.2}$$

Where L and C are the number of leaf and non-leaf tags respectively. For example, the given message in Fig. 3.9 has 12 non-leaf tags ($C = 12$), and the message has 6 leaf nodes ($L = 6$). Therefore:

$$T_{Nodes} = (12) \times 2 + 6 = 30 \tag{3.3}$$

The number of non-leaf tags is reduced by 50% and the final reduced number of the tags can be computed as:

$$R_{Nodes} = \frac{T_{Nodes} - L}{2} \tag{3.4}$$

Where R_{Nodes} is the number of the reduced tags, and by applying this equation for the same example before:

$$R_{Nodes} = \frac{30 - 6}{2} = 12 \tag{3.5}$$

Finally, the resultant number of nodes (S_{Nodes}) for the generated matrix form can be computed as:

$$S_{Nodes} = T_{Nodes} - R_{Nodes} = 18 \tag{3.6}$$

As shown in Fig. 3.10, the XML tree of the given XML message (Fig. 3.9) has only 18 tags while the message had 30 where 12 tags are removed. Technically, building XML tree is converting the XML message from the text form into a matrix data structure. The matrix form of XML tree has two columns, the first is for tags (X_i and S_j) content while the second column is for recording the parent index for the considered tag in the same matrix.

Figure 3.11 shows the generated matrix form of XML tree of SOAP messages. The procedure to build the XML tree is summarized in algorithm 3.1 and the matrix form can be expressed as:

$$XML\ Tree = \begin{pmatrix} X_0 & - \\ X_1 & P_{X_1} \\ . & . \\ . & . \\ . & . \\ X_{N-1} & P_{X_{N-1}} \\ S_0 & P_{S_0} \\ S_1 & P_{S_1} \\ . & . \\ . & . \\ . & . \\ S_{M-1} & P_{S_{M-1}} \end{pmatrix}$$

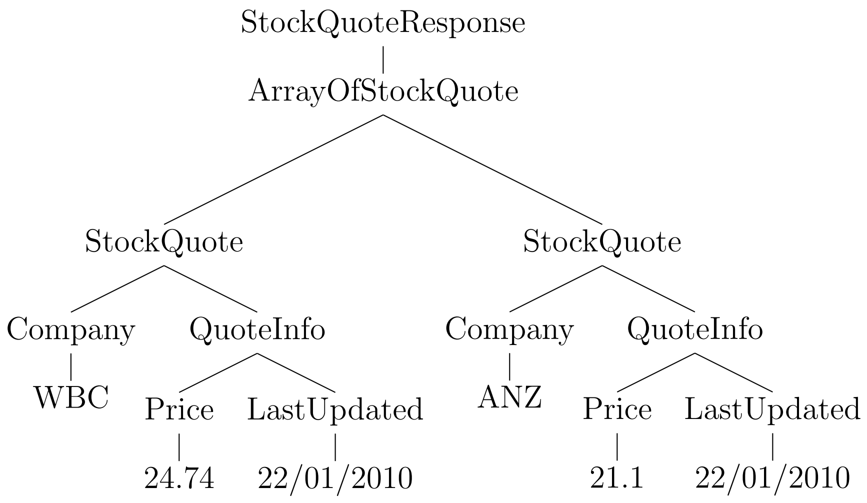


Figure 3.10: SOAP response message tree

Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	ArrayOfStockQuote	0
2	StockQuote	1
3	StockQuote	1
4	Company	2
5	QuoteInfo	2
6	Company	3
7	QuoteInfo	3
8	WBC	4
9	Price	5
10	LastUpdated	5
11	ANZ	6
12	Price	7
13	LastUpdated	7
14	24.74	9
15	22/01/2010	10
16	21.1	12
17	22/01/2010	13

Figure 3.11: Matrix form of the SOAP response

In the worst case, the number of operations is controlled by the matching of parent and children nodes in the XML tree in addition to the bubble sort $= O(n^3)$, where n is the number of nodes and data values in the XML tree.

3.3.2 Binary-tree SOAP expression

The proposed Binary-tree based model has been designed for generating the compact version of SOAP messages by first building their XML tree and then create the transformed expression using assignment binary codes. Then, SOAP expression is encoded by either fixed-length or Huffman (variable-length) encodings.

Algorithm 3.1 *Build the XML tree*

```
01: //**Notation Description:**
02: //$P$ holds the parent index
03: //$X$ holds the input XML text
04: //$D$ holds the current node content
05: //$X_{Tr}[][]$ holds the XML tree
06: $D \leftarrow getNode(X)//Get\ Root\ Node$
07: $i \leftarrow 0//Counter\ Initialization$
08: $X_{Tr}[i][0] \leftarrow D$
09: $P \leftarrow i$
10: for all $parent\ nodes\ (P)\ of\ XML\ message$ do
11:     for all $children\ of\ node\ P$ do
12:         $D \leftarrow getNextChild(X)$
13:         $i \leftarrow i + 1$
14:         $X_{Tr}[i][0] \leftarrow D$
15:         $X_{Tr}[i][1] \leftarrow P$
16:     end for
17:     $P \leftarrow P + 1$
18:     $sort\ all\ X_{Tr}\ from\ P\ to\ i$
19: end for
```

Binary XML tree transformation

Generally, the structure of XML tree is known as ordered tree such that each node in the tree can have an arbitrary number of child nodes. It is well-known that any ordered tree can be converted into a binary tree [Shaffer, 1997]. "First child/next sibling" is one of the encoding methods that can transform any ordered tree and represent it as a binary tree. Technically, first child of each node in the ordered tree becomes the left child for the same node, and the next sibling of each ordered node becomes the right child for the new left child node. Basically, the architecture of full binary tree using matrix form is based on index measurements to allocate specific locations to both left and right children of each parent node. Based on this discussion, left and right children allocation indexes can be computed as:

$$L_{Child} = P_{index} \times 2 + 1 \tag{3.7}$$

$$R_{Child} = P_{index} \times 2 + 2 \tag{3.8}$$

Where P_{index} is the index of the parent node. Equations 3.7 and 3.8 are used to allocate the positions of both left and right children respectively. The architecture of the binary tree is illustrated in Fig. 3.12. Figure 3.13 shows the resultant binary tree of XML tree (shown in Fig. 3.10) of SOAP message. Finally, to create full binary tree, any missing child (left or right) is filled with "Nil-leaf". This procedure is summarized in algorithm 3.2 which is required to achieve the XML binary tree construction. Furthermore, binary tree of SOAP messages can be encoded as an expression: {*StockQuoteResponse (ArrayOfStockQuote (StockQuote (Company (WBC, QuoteInfo (Price (24.74, LastUpdated (22/01/2010, Nil)), Nil)), Stock-Quote (Company (ANZ, QuoteInfo (Price (21.1, LastUpdated (22/01/2010, Nil)), Nil)), Nil)), Nil), Nil)*}. The complexity of algorithm 3.2 is $O(n^2)$, where n is the number of nodes and data values in the XML tree.

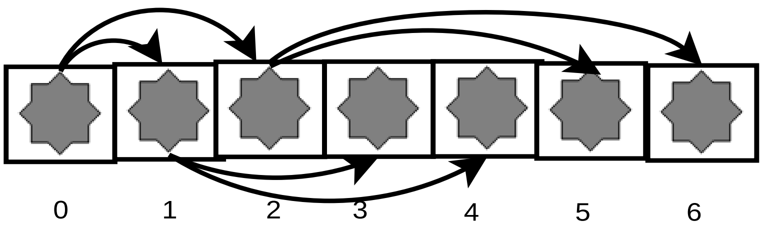


Figure 3.12: Binary tree architecture

Algorithm 3.2 *Build the XML binary tree*

```
01: //**Notation Description:**
02: //$P_{index}$ holds the parent index
03: //$X_{Tree}[][]$ holds the XML tree
04: //$B_{Tree}[]$ holds the binary XML tree
05: $P_{index} \leftarrow 0$
06: $B_{Tree}[P_{index}] = X_{Tree}[P_{index}][0]$
07: **repeat**
08:     $Ch_{index} \leftarrow P_{index} \times 2 + 1$
09:     $B_{Tree}[Ch_{index}] \leftarrow getFirstChild(P_{index})$
10:     $OCh_{index} = Ch_{index}$
11:     **repeat**
12:         $NCh_{index} = OCh_{index} \times 2 + 2$
13:         $B_{Tree}[NCh_{index}] \leftarrow getNextChild(P_{index})$
14:         $OCh_{index} = NCh_{index}$
15:     **until** No more children of $P_{index}$
16:     $P_{index} \leftarrow get\ Index\ Of\ Next\ Parent$
17: **until** No more parent nodes
```

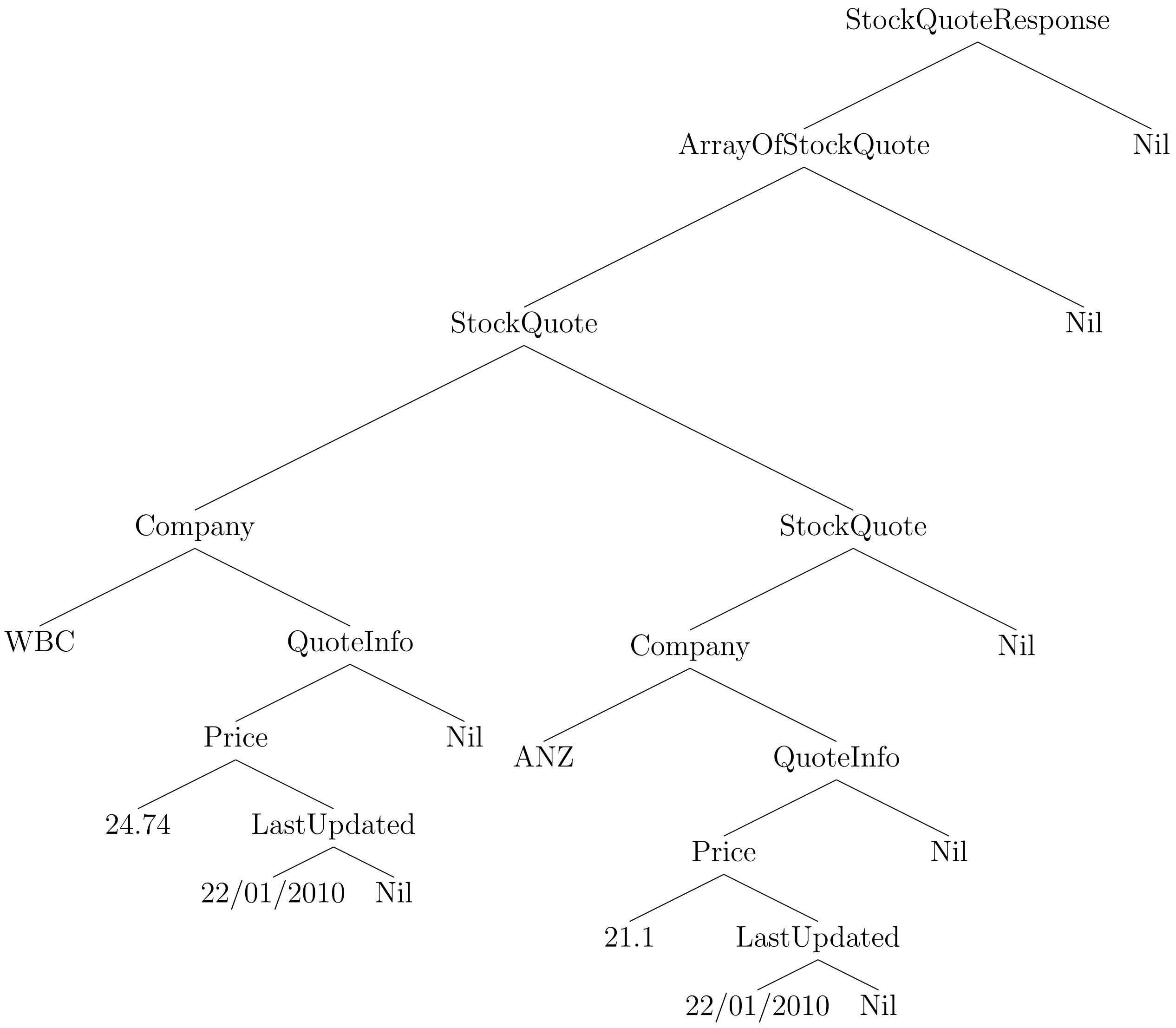


Figure 3.13: The binary tree of the SOAP response message

Assigning siblings status

In fact, building the XML binary tree of SOAP messages has resulted in producing large number of Nil-leaf which seems wasteful. Technically, Nil-leaf can be avoided by representing them with two bits. Therefore, we have four bit-codes for four cases: "11" prefix is denoting a binary node that has two binary nodes (left and right children), "10" prefix is for a binary node that has left child and right Nil-leaf (right child is nil), "01" representation is referring to a binary node that has left Nil-leaf and an ordinary right binary child, and finally, the prefix "00" is to show the ordinary leaf of the tree for which they do not have any children. Algorithm 3.3 is describing the procedure that is required to achieve the assignment process. This assignment algorithm requires only $O(n)$ (linear), where n is the number of both nodes and data values in the binary XML tree.

Consequently, message expression of binary tree for SOAP message can be encoded by applying algorithm 3.3 as {***10****StockQuoteResponse* ***10****ArrayOfStockQuote* ***11****StockQuote* ***11****Company* ***00****WBC* ***10****QuoteInfo* ***11****Price* ***00****24.74* ***10****LastUpdated* ***00****22/01/2010* ***10****StockQuote* ***11****Company* ***00****ANZ* ***10****QuoteInfo* ***00****Price* ***00****21.1* ***10****LastUpdated* ***00****22/01/2010*}

3.3.3 Two-bit status tree based SOAP expression

Two-bit status XML tree based SOAP expression is the second proposed assignment mechanism that aims to remove the duplicated closing tags directly on the general structure tree without any transformation. At the same time, the assignment mechanism generates some combinations of bit codes that would enable the receiver to reconstruct the XML tags through the capability of recognizing the correct order for tags position.

There are three suggested bit codes for assigning each node in the considered XML tree including the data leaf items. Algorithm 3.4 is required to assign the one and two bits prefix codes using the depth-first traversal technique. The first bit code "0" assigns all the non-leaf elements of the XML tree. Second code is a combination of two bits "11" to assign the right end leaf element of any complex node X_i. The second code is used to recognize the closing tag of the complex element which is considered to be the root of the tree or a subtree. Third code "10" assigns the rest of the leaf elements as it is used to refer to the data leaf elements and closing tags for their parents. The Depth-First traversal assignment algorithm is a linear process as it requires only $O(n)$, where n is the number of nodes and data values in the general XML tree.

The assigning algorithm is mainly based on the depth-first traversal. Generally, tree traversals refer to the target of visiting each node in the tree in such a specific order that depends on the considered traversal technique. Therefore, the bit code assignments are decided to be inserted due to traversing the tree. In depth-first traversal of a tree, once a node is visited, all its subtrees would be traversed next. The nodes of any subtree will be traversed completely before starting traversal of the next subtree nodes. For each non-leaf node the algorithm will assign it with "0" and for all the leaf nodes, they will be assigned with "10" except the last visited leaf node of each subtree as it will be assigned by "11".

As a result of this assigning process, the XML message would be transformed again into a textual form but in a different representation. For example, the assigned textual form for the status XML tree (Fig. 3.14) would be in the following form: {*__0__StockQuoteResponse __0__ArrayOfStockQuote __0__StockQuote __0__Company __10__WBC __0__QuoteInfo __0__Price __10__24.74 __0__LastUpdated __11__22/01/2010 __0__StockQuote __0__Company __10__ANZ __0__QuoteInfo __0__Price __10__21.1 __0__LastUpdated __11__22/01/2010*}

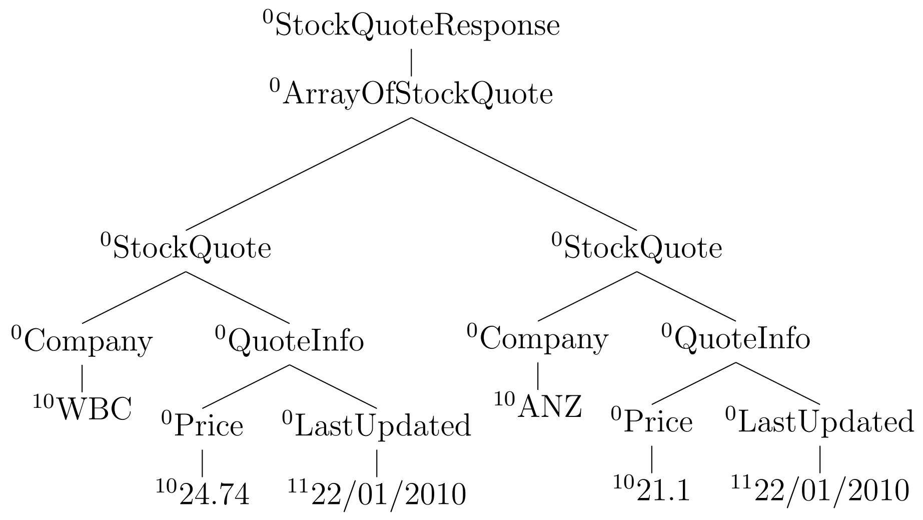


Figure 3.14: Two-bit assigned SOAP message tree

This assignment of bit codes would enable the decompression algorithm to recognize the correct positions of each tag. To build the XML tree again, the order would be divided into two queues, the first queue starts with the root node of that tree and ends with the first node that is assigned as ”11”, where it means the end of the current level. As described before, both bit codes ”10” and ”11” refer to data leaf items and the parent is the closest node to them at the left hand side of the queue which assigned with ”0”.

Algorithm 3.3 *Assigning siblings status*

```
01: //Notation Description:
02: //index holds the nodes index
03: //$B_{Tree}$[] holds the binary XML tree
04: //$L_{Ch}$ and $R_{Ch}$ hold the indexes of nodes children
05: for $index = 0$ to $n - 1$
06:   $L_{Ch} \leftarrow index \times 2 + 1$
07:   $R_{Ch} \leftarrow index \times 2 + 2$
08:   if $Both\ B_{Tree}[L_{Ch}]\ and\ B_{Tree}[R_{Ch}] \neq Nil$
09:     $Assign\ B_{Tree}[index] \leftarrow 3\ ((11)_2)$
10:               $//has\ two\ children$
11:   else if $(B_{Tree}[L_{Ch}] = Nil)\ and\ (B_{Tree}[R_{Ch}] \neq Nil)$
12:     $Assign\ B_{Tree}[index] \leftarrow 2\ ((10)_2)$
13:               $//has\ one\ left\ child$
14:   else if $(B_{Tree}[L_{Ch}] \neq Nil)\ and\ (B_{Tree}[R_{Ch}] = Nil)$
15:     $Assign\ B_{Tree}[index] \leftarrow 1\ ((01)_2)$
16:               $//has\ one\ right\ child$
17:   else $Assign\ B_{Tree}[index] \leftarrow 0\ ((00)_2)$
18:               $//has\ no\ children$
19:   end if
20: end for
```

Algorithm 3.4 *$Depth - First\ traversal\ assignment$*

```
01: //Notation Description:
02: //$Nd$ holds a node content
03: $Nd \leftarrow Root\ Node\ of\ Tree$
04: repeat
05:     if $Nd\ a\ non-leaf\ element$ then
06:         $Nd \leftarrow Assigned\ Nd\ with$ "0"
07:     else if $GetNextChildNode(Nd) = False$ then
08:         $Nd \leftarrow Assigned\ Nd\ with$ "11"
09:     else
10:         $Nd \leftarrow Assigned\ Nd\ with$ "10"
11:     end if
12:     $Nd \leftarrow GetNextChildNode(Nd)$
13: until No More Elements
```

3.3.4 One-bit status tree based SOAP expression

The One-bit status tree is another proposed bit assignment process that uses only one bit code for each tag and data leaf item in the tree. The suggested bit codes ("0" and "1"') as assignment codes are completely based on the breadth-first traversal for the XML tree. In the breadth-first traversal, all nodes that are given at a specific level would be traversed completely before traversing any node at the next level. The basic strategy of this model is to assign the last traversed node of each level of the XML tree with "1" and assign all other nodes with "0". Therefore, when the decompression technique reads a node with status "1", that means the end of the current level nodes and all the nodes between the current status node and the considered parent node in addition to that status node are children to that parent node at the considered level.

Algorithm 3.5 is required to perform the breadth-first assignment process for the XML tree. The complexity of this assignment algorithm is $O(n^2)$, where n is the number of nodes and data values in the XML tree. Figure 3.15 shows the resultant status XML tree of the assignment process for this model. For example, the status textual form of the assigned XML tree would be expressed as {***0**StockQuoteResponse* ***1**ArrayOfStockQuote* ***0**StockQuote* ***1**StockQuote* ***0**Company* ***1**QuoteInfo* ***0**Company* ***1**QuoteInfo* ***1**WBC* ***0**Price* ***1**LastUpdated* ***1**ANZ* ***0**Price* ***1**LastUpdated* ***1**24.74* ***1**22/01/2010* ***1**21.1* ***1**22/01/2010*}

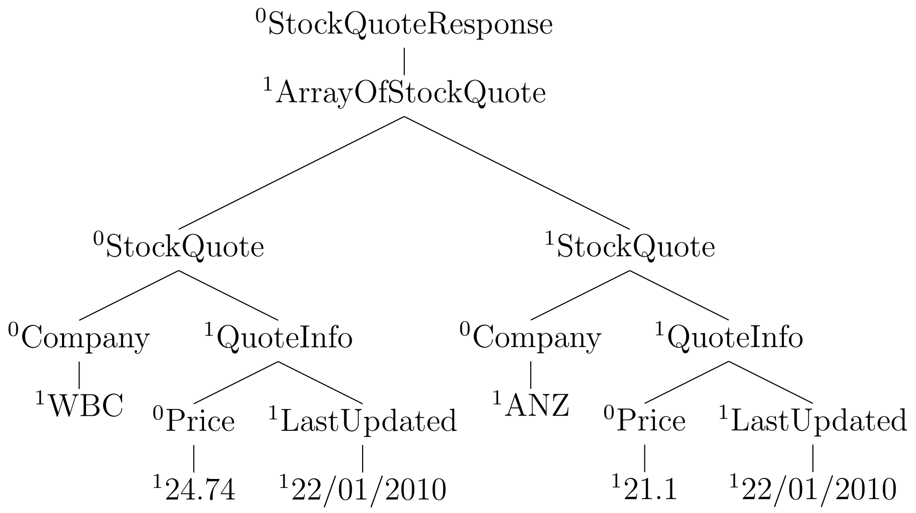


Figure 3.15: One-bit assigned SOAP message tree

3.3.5 Encoding of SOAP expressions

Both fixed and variable length encoding techniques are developed for the generated SOAP message expressions.

Huffman as variable-length encoding

As a variable-length encoding technique, Huffman binary tree encoding is one of the well-known methods that removes the redundancies of letters. Huffman is one of the lossless

Table 3.1: XML tree nodes redundancies of SOAP message

Node Content	**Node Redundancy**
StockQuoteResponse	1
ArrayOfStockQuote	1
StockQuote	2
Company	2
QuoteInfo	2
WBC	1
ANZ	1
Price	2
LastUpdated	2
24.74	1
22/01/2010	2
21.1	1

entropy encoding technique that could remove redundant information efficiently. The resultant compressed version of Huffman is completely determined by a set of probabilities. The input symbol with a very high probability of occurrences is encoded with very few bits. On the other hand, the input symbol with a low probability is encoded with a larger number of bits. This technique replaces each symbol in the input message with another symbol code (bit code). Basically, Huffman technique assigns variable length binary codes to characters such that determining the length of the binary codes is based on the relative frequencies (redundancies) of the corresponding characters.

Building Huffman binary tree is generally based on a number of iterative computing steps. First, order the characters of a list by ascending weight (weight is characters redundancy). Second, assign the first two characters that have the minimum weights (redundancies) as the children of an internal node such that its weight is the sum of the weight of the two children. Next, remove the two first characters and pick up the sum and put it on the same list in the correct position keeping the ascending order of the whole list. Table 3.1 shows the redundancies of XML tags in the given SOAP message. This process is repeated until all the items are assigned under one internal node. Once the Huffman tree is constructed, assigning the binary codes to individual characters is started. Technically, all the left edges in the tree are assigned by '0' and all the right edges are assigned by '1'.

Algorithm 3.5 *Breadth – First traversal assignment*

```
01: //Notation Description:
02: //$QN$ is a queue of the traversed nodes
03: //$Nd$ holds a node content
04: $Nd \leftarrow Root\ Node\ of\ Tree$
05: $Nd \leftarrow Assigned\ Nd\ with$ "0"
06: $QN \leftarrow Add\ Nd\ to\ the\ queue$
07: repeat
08:     $Nd \leftarrow Assigned\ Nd\ with$ "0"
09:     repeat
10:         $Ch \leftarrow GetNextChildNode(Nd)$
11:         if $Ch\ is\ the\ last\ child$ then
12:             $Ch \leftarrow Assigned\ Ch\ with$ "1"
13:             $QN \leftarrow Add\ Ch\ to\ the\ queue$
14:         else
15:             $Ch \leftarrow Assigned\ Ch\ with$ "0"
16:             $QN \leftarrow Add\ Ch\ to\ the\ queue$
17:         end if
18:     until No More children of Nd
19:     $Nd \leftarrow Next\ Node\ of\ the\ Queue(QN)$
20: until No More Elements
```

Traditionally, Huffman binary tree encoding is used to encode characters. Huffman method has been developed in this model to encode the whole content of XML complex and simple nodes as individual items of SOAP Message. In other words, contents of the XML nodes of the SOAP message are the basic items of Huffman encoding as alternative to characters. Table 3.2 shows the binary codes of the XML simple and complex nodes resulted from Huffman encoding technique.

Fixed-length encoding

Fixed length encoding is considered as a another encoding technique for encoding the assigned XML tags and data leaf. In fixed-length encoding, every input word would have the same code length. Therefore, the number of the required bits (B_{len}) for every single code is completely based on the total number of the considered XML tags and data leaf of the input XML text.

$$B_{len} = Round(\log(n) + 0.5) \tag{3.9}$$

Where n is the total number of the input XML tags and data leaf items. Table 3.2 shows the generated binary codes of the XML nodes using fixed-length encodings for the SOAP message given in Fig. 3.9. Figure 3.16 shows the structure for the compressed SOAP message.

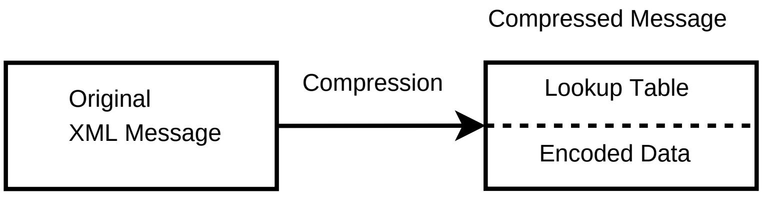


Figure 3.16: Compressed XML message structure

Table 3.2: Binary codes of XML nodes of SOAP message

Node Content	**Huffman Code**	**fixed-length Code**
StockQuoteResponse	0000	0000
ArrayOfStockQuote	0001	0001
StockQuote	001	0010
Company	010	0011
QuoteInfo	011	0100
WBC	11110	0101
Price	100	0110
LastUpdated	101	0111
24.74	11111	1000
22/01/2010	110	1001
21.1	1110	1010

3.4 Experiments and discussions

Generally, most experimental work of SOAP compression evaluation is for large and very large messages that vary from few kilobytes to tens of kilobytes. In this chapter, we consider message size varying from only 140 bytes to 53 Kbytes to show the efficiency of the proposed techniques for small as well as large message size. This also helps in investigating the performance of the proposed techniques on small messages against some of the well-known compression techniques such as XMILL, XbMILL, Gzip, and bzip2. In fact, the proposed compression techniques have shown significant achievements with promising compression ratio for all message sizes including very tiny samples as small as 140 bytes.

Technically, lossless compression techniques create lookup tables for mapping symbols to binary codes during the compression process. In the proposed models, once an XML message is compressed, it consists of two parts: lookup table and encoded data (i.e. tags and data leaf). For small messages, the lossless encoding would result in a big lookup table in comparison with the encoded part of the tags and data leaf. However, for large and very large SOAP messages, the resultant lookup table is quite small in comparison with the other

Table 3.3: Average compression ratio of both fixed and variable length encoding of Two-bit and One-bit status techniques in addition to the Binary-tree compression technique

Message Type	**Small**	**Medium**	**Large**	**Very large**
Range(byte)	140-800	800-3000	3000-20000	20000-55000
No.of Messages	40	40	40	40
Two-bit Fix.Length	2.16	3.2	7.38	11.57
Two-bit Var.Length	2.02	3.1	8.03	13.9
One-bit Fix.Length	2.2	3.24	7.6	12.11
One-bit Var.Length	2.04	3.11	8.31	14.7
Binary-tree Fix.Length	2.14	3.16	7.16	11.08
Binary-tree Var.Length	2.01	3.03	7.78	13.2

encoded part of the considered message. Consequently, the overall resultant compression ratio of large messages is higher than the small ones.

The evaluation of the proposed techniques is based on testing their performance in terms of the compression ratio and the processing time for both compression and decompression for messages in various sizes. A test bed has been set up with 160 SOAP messages that is equally divided into four groups based on message size. Each group of these messages has 40 samples. The considered groups are small (140-800 bytes), medium (800-3000 bytes), large (3000-20000 bytes), and Very large (20000-55000 bytes). The resultant average compression ratio for all the proposed techniques have been shown in Table 3.3. Both fixed-length and Huffman encodings of the proposed models showed very high compression ratios for large and very large documents. At the same time, all techniques have reduced the size of small and medium messages successfully. The results for small documents are interesting giving the fact that non of the existing techniques achieve such compression ratio. In fact, when these techniques were applied to small sized documents they may even add overhead to the compressed message size.

Both Two-bit and One-bit techniques using both fixed and variable length encodings

Table 3.4: Resultant compressed size of different SOAP messages using XMILL, XbMILL,gzip, and bzip2 compressors

Size(b)	**XMILL**	**XBMILL**	**Gzip**	**bzip2**
146	121	167	116	128
211	161	193	154	165
313	208	261	192	214
593	294	351	278	301
817	324	399	322	337
1392	475	556	473	492
2544	671	784	706	776
3110	739	829	833	804
9775	1458	1462	1861	1528
16997	2090	1900	2822	2019
22728	2500	2226	3531	2393
36114	3893	3447	5164	3193
47800	4764	4008	6462	3856
53346	5175	4236	7150	4105

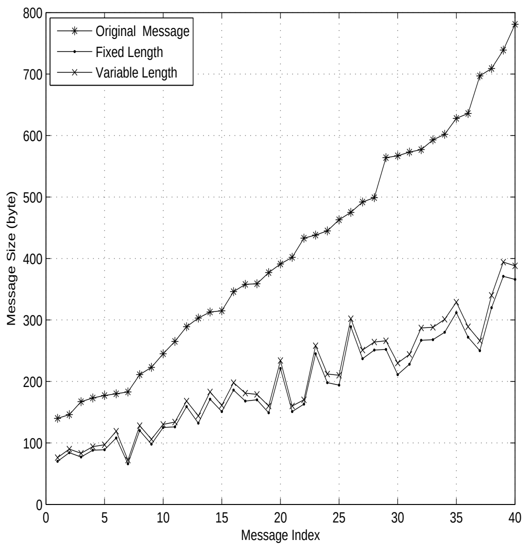
Table 3.5: Resultant compressed size of different SOAP messages using fixed and variable length encodings of One-bit, Two-bit status techniques, and Binary-tree based technique

Original	**Binary-tree**		**Two-bit Technique**		**One-bit Technique**	
Size(b)	**Fix.Len**	**Var.Len**	**Fix.Len**	**Var.Len**	**Fix.Len**	**Var.Len**
146	84	90	84	89	83	89
211	120	128	119	127	119	126
313	171	183	170	182	169	181
593	268	288	266	285	263	283
817	306	323	302	320	299	317
1392	483	512	477	507	471	501
2544	667	687	657	678	648	668
3110	772	795	760	783	748	771
9775	1441	1370	1403	1332	1366	1295
16997	1927	1723	1862	1658	1798	1594
22728	2318	2009	2232	1922	2146	1836
36114	3236	2702	3099	2565	2962	2427
47800	4033	3339	3851	3157	3670	2975
53346	4411	3635	4208	3432	4006	3230

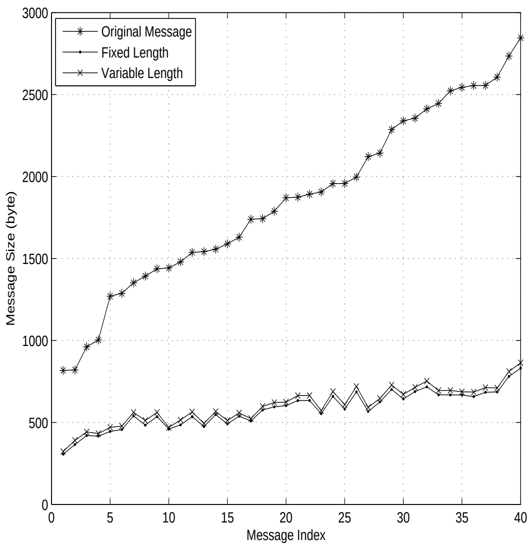
are compared to each other in addition to the Binary-tree based compression technique in Table 3.3 and to other XML-aware and generic compressors in Tables 3.4 and 3.5 showing their average compression ratios and the resultant reduced size for all the tested SOAP messages. There is a marginal difference between the results of fixed and variable length encodings of the Binary-tree based, Two-bit status tree based, and One-bit status tree based techniques for small and medium size messages as the fixed length encoding achieved better reduction than the variable length encoding. However, variable length encoding achieved significantly better results for large and very large size messages. On the other hand, One-bit technique shows better reduction than Two-bit technique when they use the same encoding for their status XML tree. All the proposed techniques in this chapter are compared against XML-aware compressors (e.g XMill and XbMill techniques) as well as generic compressors (e.g gzip and bzip2).

The SOAP messages size reduction that can be achieved by the Binary-tree based compression technique has been shown in Fig. 3.17 as results show the significant achievement of the Binary-tree technique on different size of messages. Figure 3.18 shows the original and compressed size of the four groups of XML documents (small, medium, large, and very large) using Huffman and fixed-length encodings for the Two-bit status tree compression technique. The results show that the size of all the tested documents have been compressed efficiently, even the smallest document (140 bytes) has been reduced to less than 90 bytes. Even better results were obtained by One-bit status tree compression technique. The results in Fig. 3.19 show the significant achievement of the One-bit status tree compression technique as the size of all the large and very large documents have been reduced significantly as well as reducing the size of small and medium documents successfully.

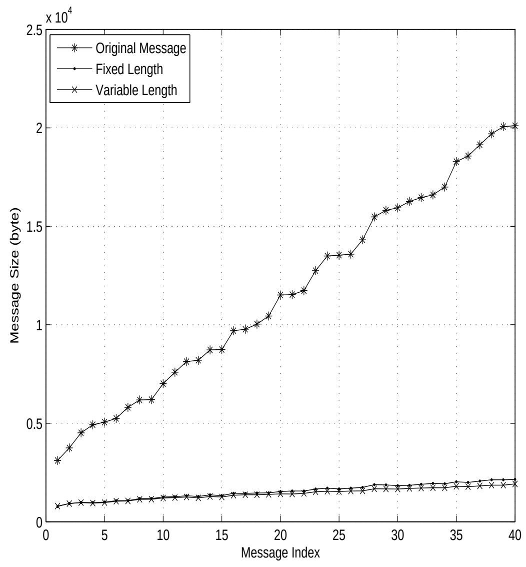
As shown in Table 3.4 and 3.5, all the proposed techniques outperformed both XML-aware (XMill and XbMill technique) and generic (gzip and bzip) compressors except one sample that is reduced slightly more by bzip2 technique than fixed length encoding of the



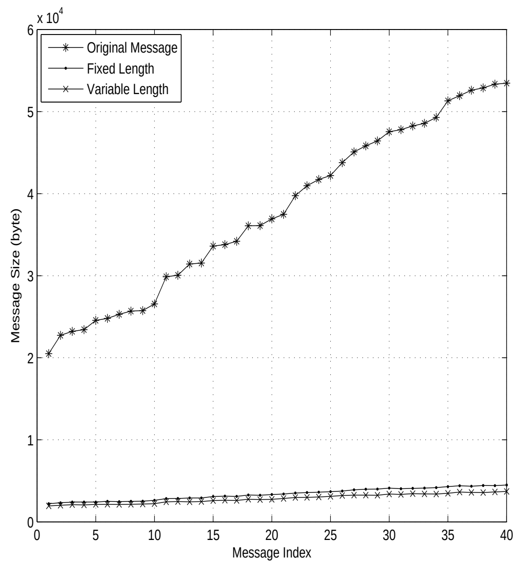
a. Small sized SOAP samples



b. Medium sized SOAP samples



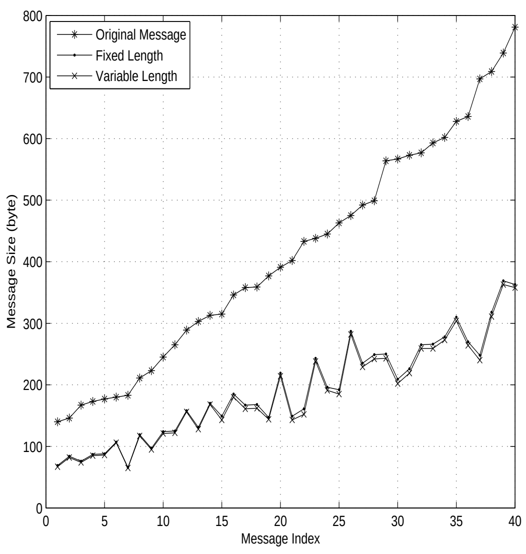
c. Large sized SOAP samples



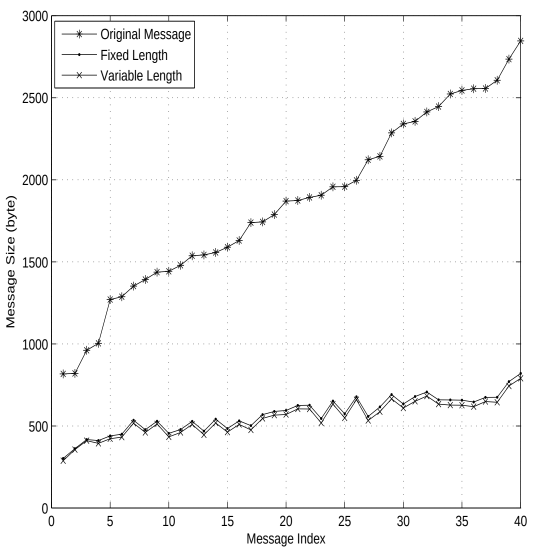
d. Very large sized SOAP samples

Figure 3.17: Binary-tree compression of different size of SOAP messages

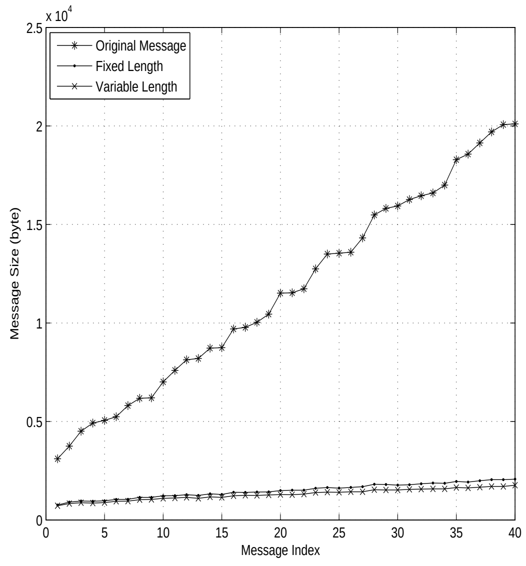
Binary-tree based technique and Two-bit status XML tree technique. Considering all the proposed techniques, fixed length encoding of all techniques has achieved slightly better



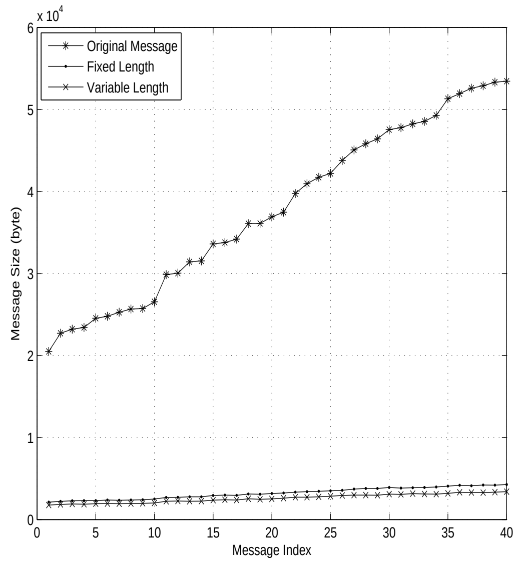
a. Small sized SOAP samples



b. Medium sized SOAP samples



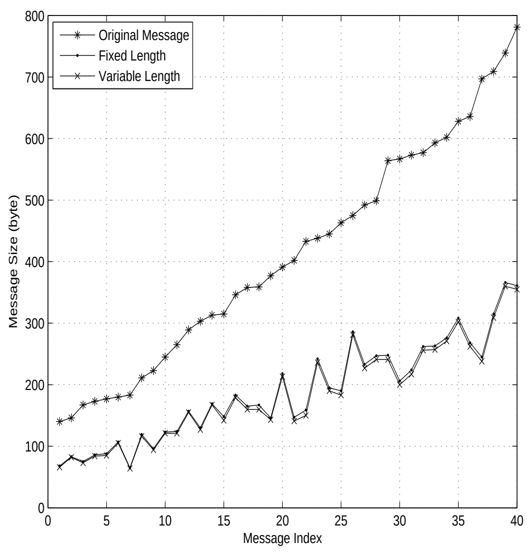
c. Large sized SOAP samples



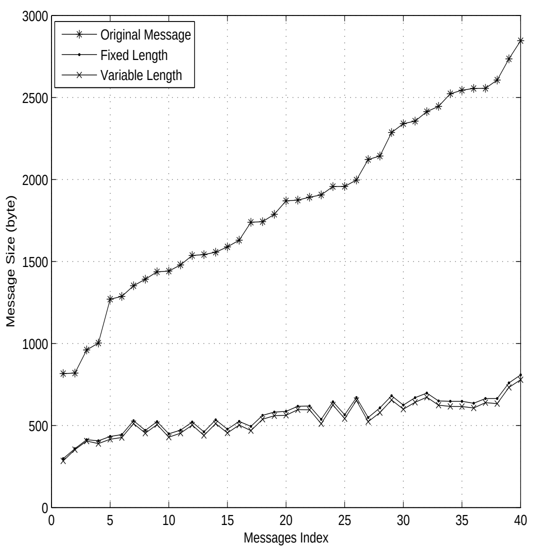
d. Very large sized SOAP samples

Figure 3.18: Two-bit variable-length and Two-bit fixed-length of different size of SOAP messages

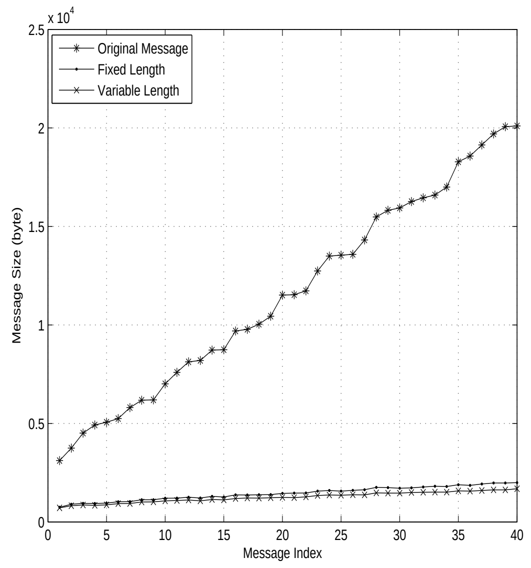
reduction in comparison with variable length encodings for small and medium size messages. On the other hand, variable length encoding (Huffman) has significantly outperformed the



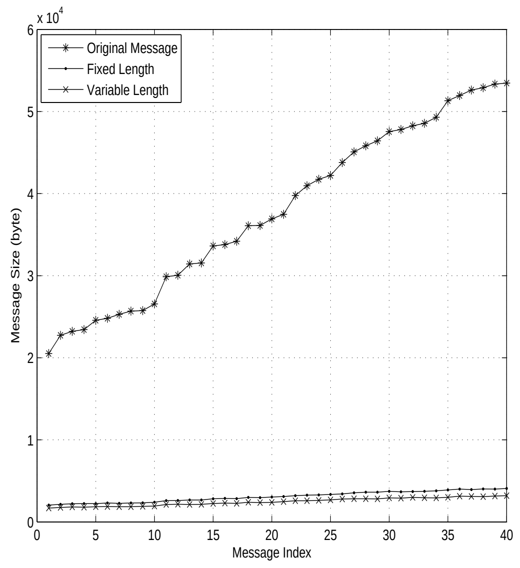
a. Small sized SOAP samples



b. Medium sized SOAP samples



c. Large sized SOAP samples



d. Very large sized SOAP samples

Figure 3.19: One-bit variable-length and One-bit fixed-length of different size of SOAP messages

fixed length encoding as well as all the other techniques for large and very large size messages. The achievement of bzip2 is very competitive with other XML-aware compressors for small, medium, and large XML messages and at the same time bzip2 outperformed both XMill and XbMill as well as gzip for the very large size messages. For small messages, gzip as a generic compressor has shown higher reduction than all the considered XML-aware encodings in addition to the bzip2. Furthermore, the results show better size reduction by XMill than XbMill for small messages while XbMill marginally outperformed XMill for all medium and large, and significantly for the very large messages.

Table 3.6: Experimental devices features

Device	**PC**	**Laptop**	**NetBook**	**PDA**
Manufacturer	HP	Dell	Acer	dopod
Model	HP Compaq dc7800	INSPIRON1525	AOD260	dopod 838
Processor	Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz	Intel(R) Pentium(R) Dual CPU T2370 @ 1.73GHz	Intel(R) Atom(TM) CPU N450 @1.66GHz	Samsung @ 400 MHz
RAM	3.48 GB	1.00 GB	1.00 GB	64 MB
System Type	32-bit Operating System	32-bit Operating System	32-bit Operating System	Microsoft Windows Mobile Version 5.0

Figure 3.20 shows the resultant average compression ratios for all the proposed techniques and other techniques that are involved in the designed testbed for the evaluation. As mentioned earlier, although fixed length encodings of the Binary-tree, Two-bit, and One-bit status tree techniques outperformed all the techniques (including variable length encoding of the proposed techniques) for small and medium messages. However, variable length encoding of all techniques again have achieved significantly better compression ratios than fixed length encoding as well as other techniques. At the same time, this confirms the fact that gzip has higher performance than other standards on small messages.

With the aim to evaluate the compression and decompression time on different platforms, four devices: personal computer (PC), laptop, netbook, and PDA are selected to run the

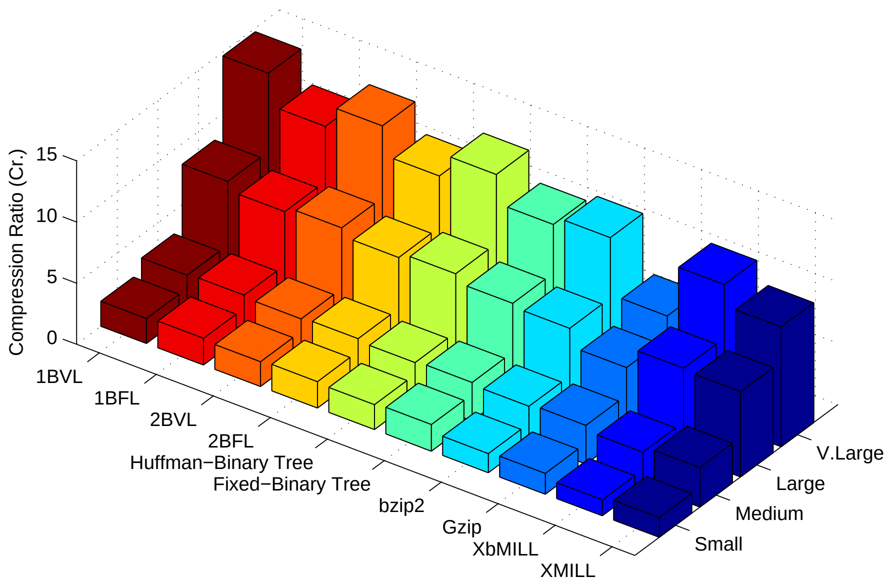


Figure 3.20: Average compression ratio of different implementations for all types of messages

proposed techniques. Table 3.6 shows the features of all the experimental devices (PC, Laptop, NetBook, and PDA). Figures 3.21, 3.22, and 3.23 show the compression time of small, medium, large, and very large sized messages using Binary-tree, Two-bit and One-bit Huffman based techniques respectively. Both One-bit and Two-bit compression techniques have shown significantly better compression time than the Binary-tree technique. It is clear that the performance of the experimental device has an impact on the consumed processing time as the PC device showed the best results for all techniques in comparison with other devices for the same technique. Although the PDA device has very limited computing power, it shows promising results for One-bit and Two-bit techniques. Table 3.7 shows the average compression time for small, medium, large, and very large sized messages in addition to the overall average compression time using all models. Both One-bit and Two-bit compression techniques require compression time significantly less than Binary-tree based technique of both fixed length and Huffman encodings. The results are potentially promising showing the fact that the proposed techniques require less than one millisecond for compressing small messages on PC and not more than 31.6 millisecond on PDA device.

Table 3.7: Average compression time (millisecond) of small, medium, large, and very large sized messages using fixed-length and Huffman based One-bit, Two-bit status techniques, and Binary-tree technique

		Binary-tree		**Two-bit Status Tree**		**One-bit Status Tree**	
Device	**Message Size**	**Fixed**	**Huffman**	**Fixed**	**Huffman**	**Fixed**	**Huffman**
PC	Small	4.05	4.025	0.1	0.18	0.1	0.53
	Medium	10.8	10.85	1.55	2.68	1.33	2.6
	Large	599.68	599.48	61.65	65.05	61.13	65.1
	Very large	12327.15	12330.85	609.33	611.7	607.98	611.65
	Overall	3235.42	3236.3	168.16	169.9	167.63	169.97
Laptop	Small	13.225	17	0.875	1.525	0.8	1.7
	Medium	33.1	30.98	7.28	6.95	5.58	8.75
	Large	1453.28	1458.08	198.55	197.28	179.48	188.38
	Very large	29515.48	29680.28	2026.7	2002.3	1978.63	1990.75
	Overall	7753.77	7796.58	558.35	552.01	541.12	547.39
NetBook	Small	14.73	17.45	1.65	4.48	4.63	5.1
	Medium	48.88	47.2	14.65	21.48	17.73	24.6
	Large	2350.68	2335.58	573.65	728.95	776.75	770.93
	Very large	43486.95	43057.73	4906.05	5321.78	5198.55	4899.85
	Overall	11475.31	11364.49	1374	1519.17	1499.41	1425.12
PDA	Small	305.9	308.85	28.78	36.93	24.68	31.6
	Medium	1085.13	1049.58	187.68	235.55	180.05	239.85
	Large	53187.73	52055.05	5582.2	5788.55	5567.13	5746.35
	Very large	1020907	1034418	59075.35	59602.85	59193.35	59617.93
	Overall	268871.4	271958	16218.5	16415.97	16241.3	16408.93

Furthermore, Figures 3.24, 3.25, and 3.26 show the decompression time for small, medium, large, and very large sized messages using Binary-tree, Two-bit and One-bit techniques respectively. Again, both Two-bit and One-bit techniques showed potentially higher performance in terms of the required decompression time on the devices in comparison with the Binary-tree technique. Table 3.8 shows the average decompression time for small, medium,

Table 3.8: Average decompression time (millisecond) of small, medium, large, and very large sized messages using One-bit, Two-bit status techniques, and Binary-tree technique

Device	**Message Size**	**Binary-tree**	**Two-bit Status Tree**	**One-bit Status Tree**
PC	Small	0.06	0.02	0.02
	Medium	0.53	0.03	0.06
	Large	223.85	0.23	0.185
	Very large	5894.6	1.58	1.55
	Overall	1529.76	0.46	0.43
Laptop	Small	0.16	0.15	0.03
	Medium	2.55	0.18	0.13
	Large	536.6	1.58	1.33
	Very large	13753.5	4.48	4.23
	Overall	3573.16	1.59	1.43
NetBook	Small	0.18	0.16	0.18
	Medium	3.78	1.13	1.2
	Large	852.23	5.45	5.95
	Very large	21992.18	12.65	21.25
	Overall	5712.04	4.83	7.14
PDA	Small	7.68	0.23	0.18
	Medium	71.75	1.68	1.7
	Large	8133.68	12.55	12.85
	Very large	194035.5	40.35	40.45
	Overall	50562.14	13.7	13.79

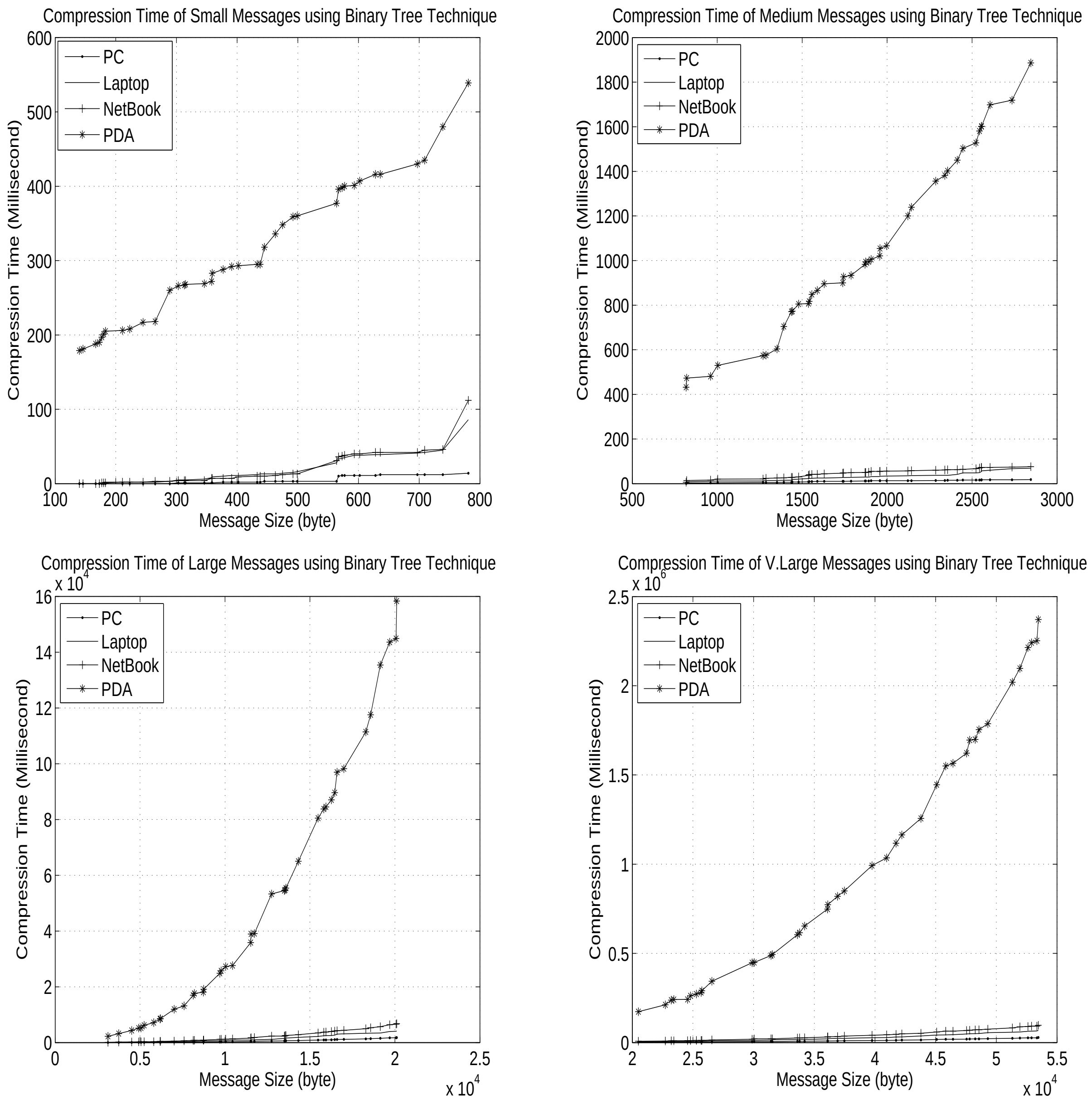


Figure 3.21: Compression time of small, medium, large, and very large sized messages using Huffman based Binary-tree technique

large, and very large sized messages in addition to the overall average decompression time using all models. Both techniques require even less than one millisecond to decompress small sized messages on all devices. Moreover, they require about only 14 milliseconds to decom-

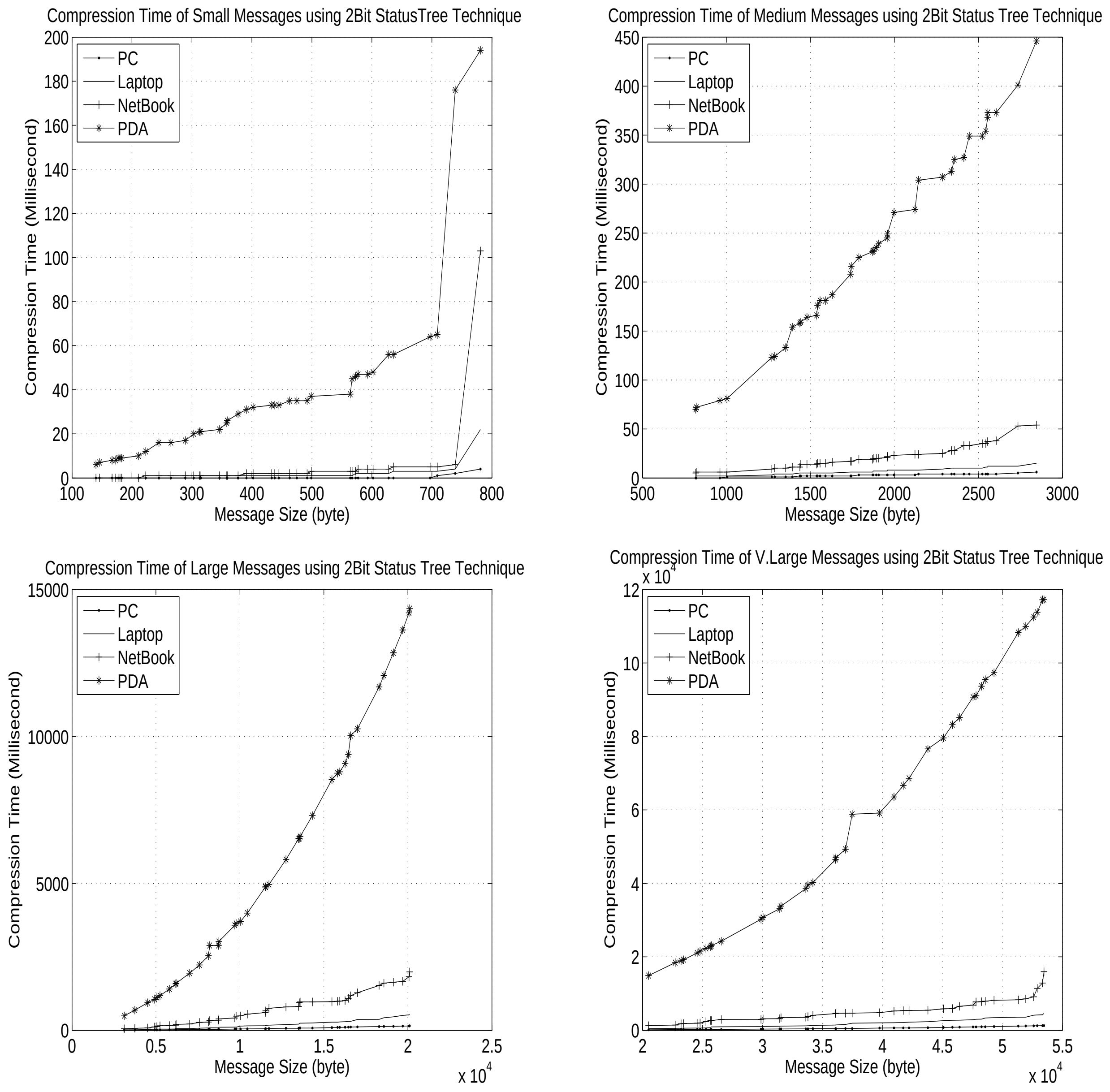


Figure 3.22: Compression time of small, medium, large, and very large sized messages using Huffman based Two-bit status technique

press the very large sized messages on the PDA device and less than 2 milliseconds on the PC.

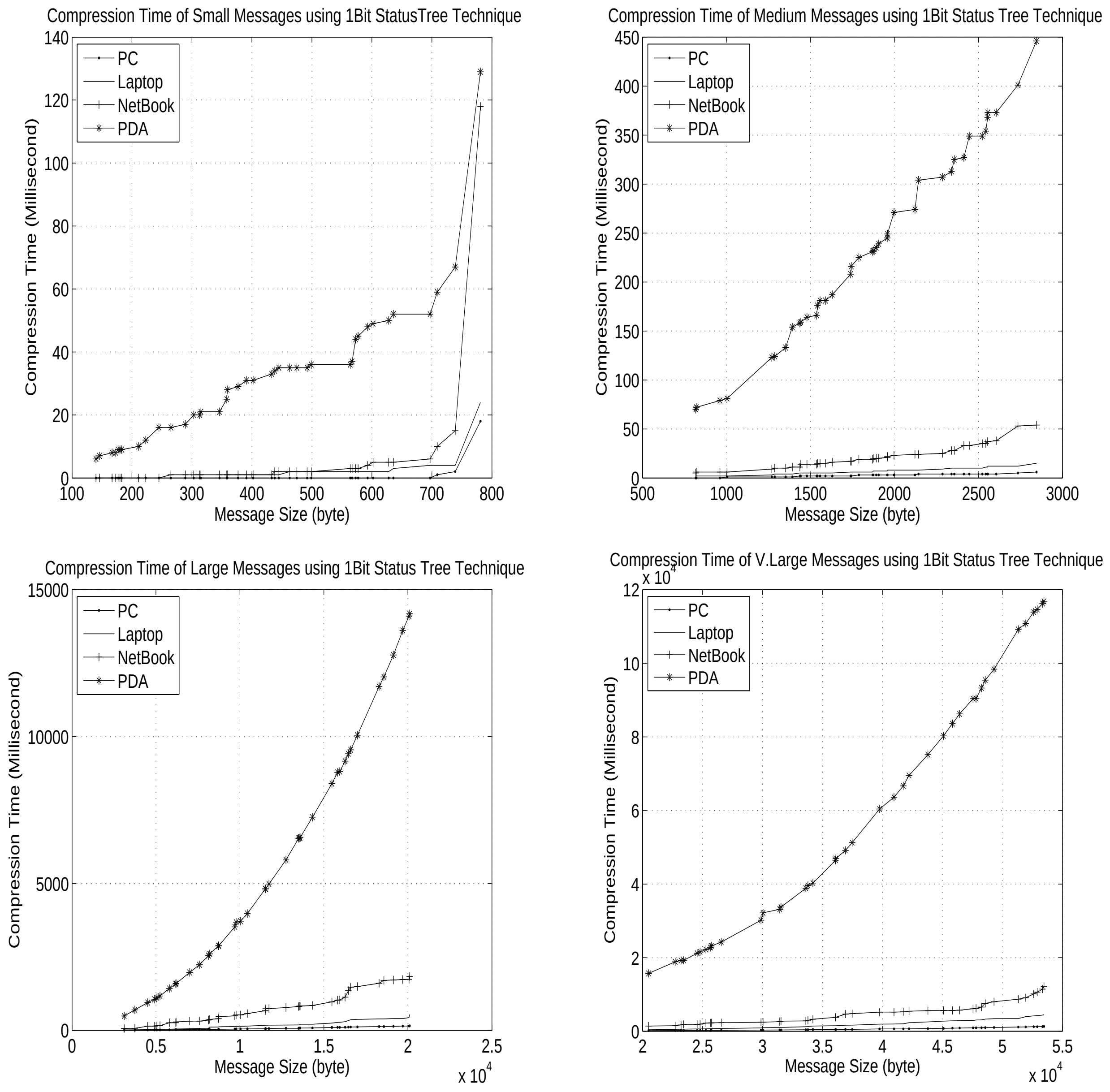


Figure 3.23: Compression time of small, medium, large, and very large sized messages using Huffman based One-bit status technique

3.5 Conclusion

Developing Web services compression techniques and getting higher average compression ratio would improve the performance of Web services by reducing the network traffic. More-

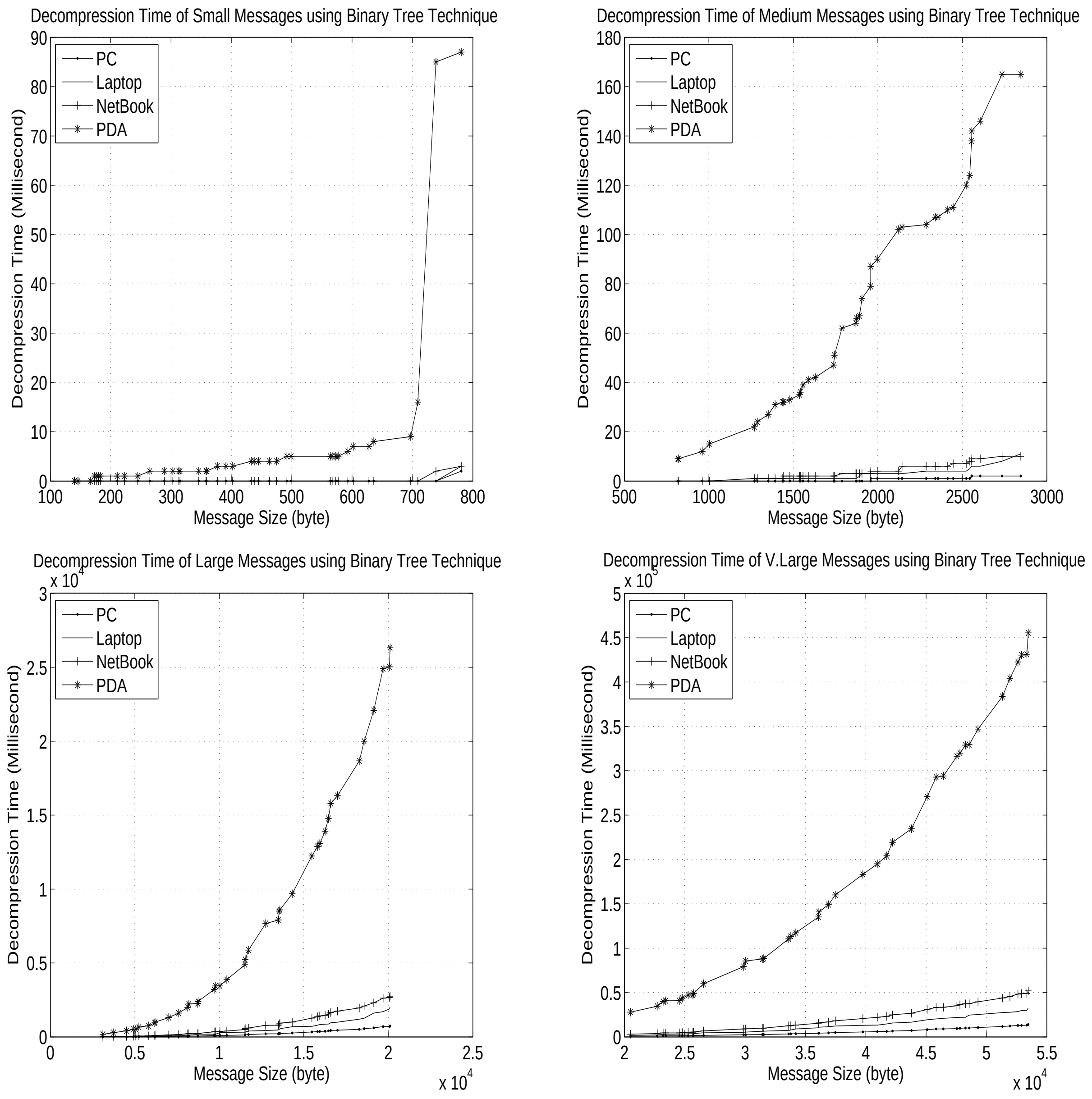


Figure 3.24: Decompression time of small, medium, large, and very large sized messages using Binary-tree technique

over, this development would support low bandwidth environment, and especially, the low connectivity clients such as PDAs connected to an enterprise server. In this chapter, SOAP message tree structure has been developed in an innovative way to reorder the XML nodes

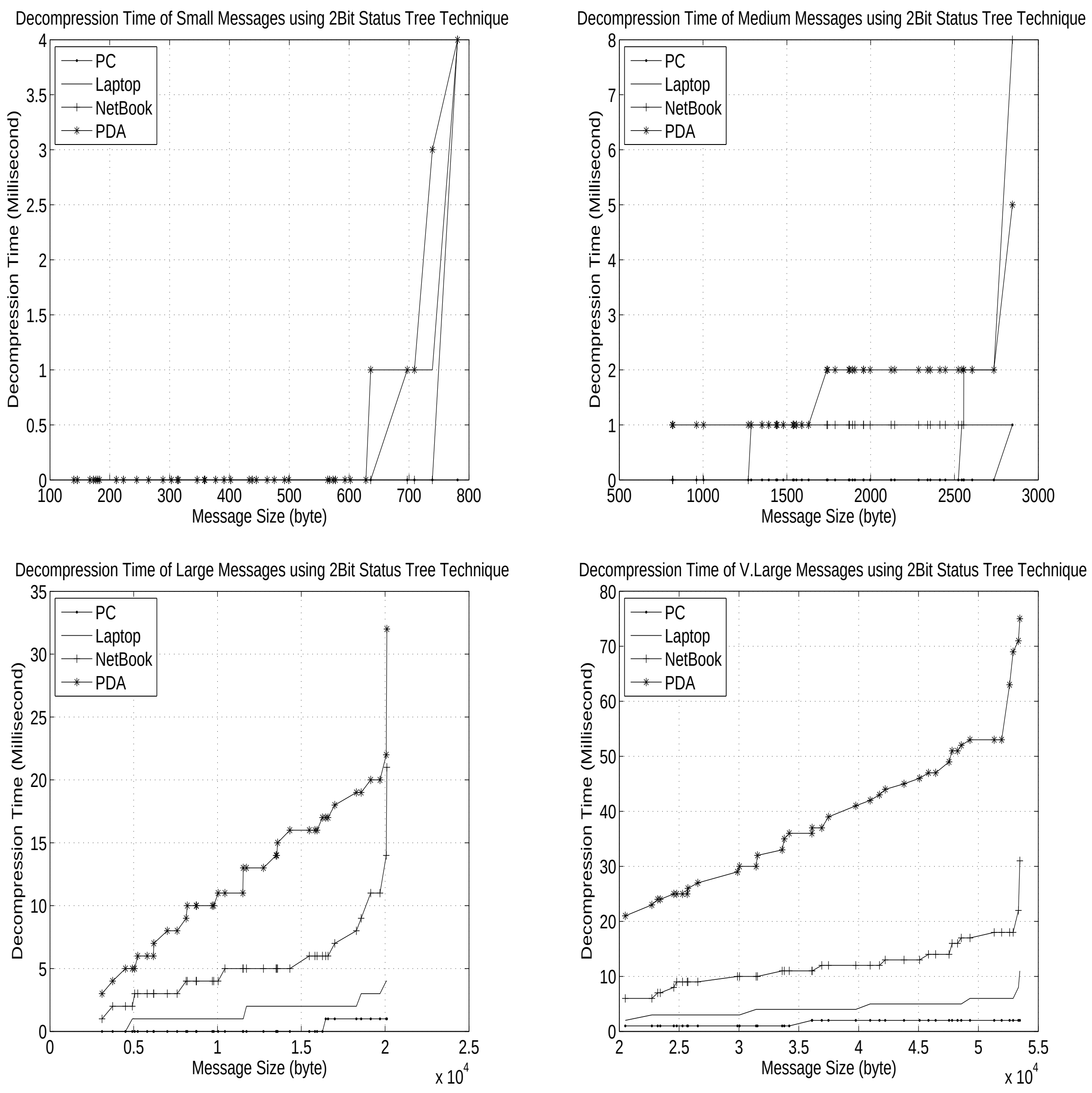


Figure 3.25: Decompression time of small, medium, large, and very large sized messages using Two-bit status technique

in such a way to be re-constructable. At the same time, the entire XML nodes have been chosen as individual input for both fixed-length and variable-length encodings.

Three assignment techniques for SOAP message tree are developed using several bit codes

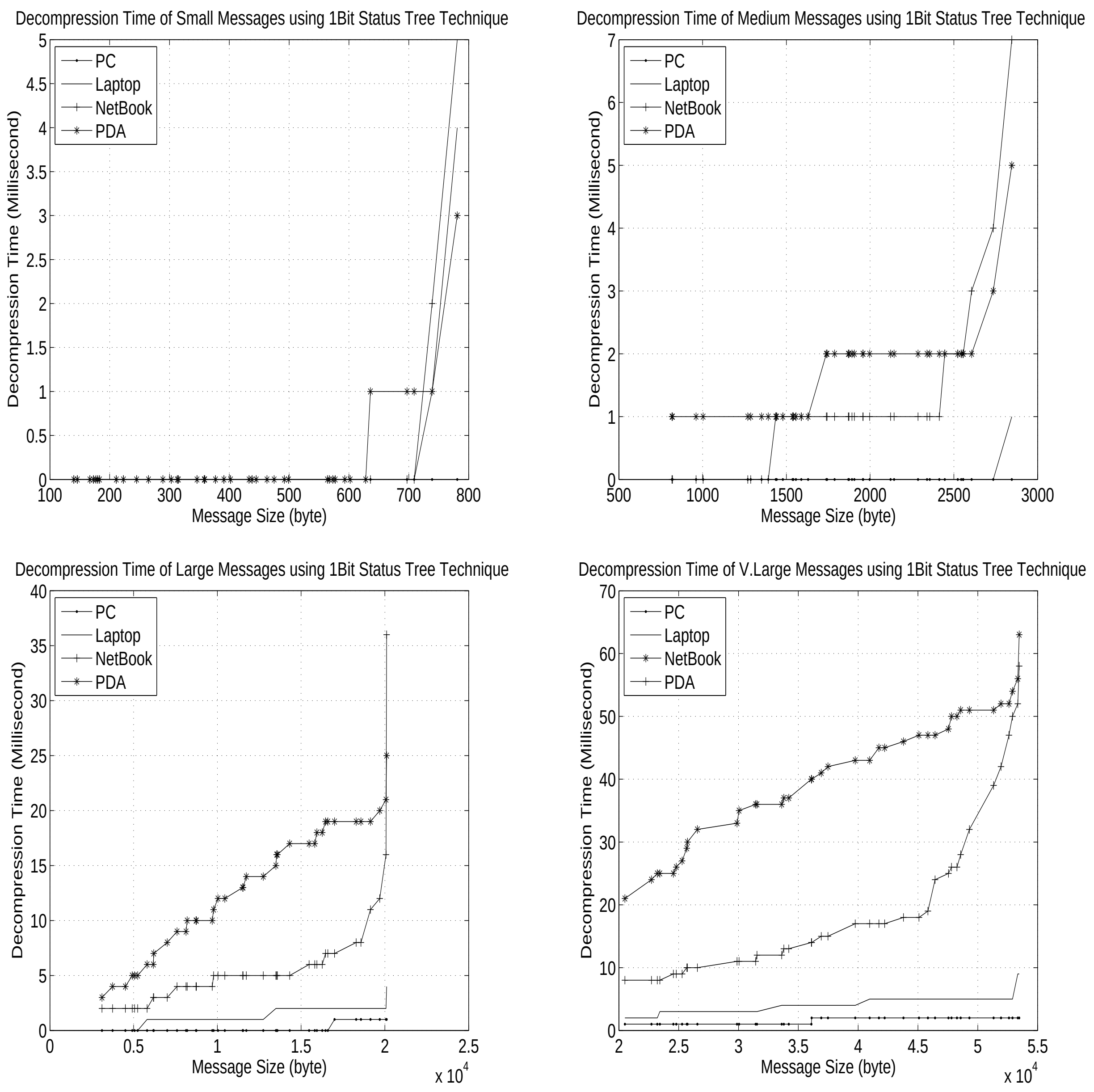


Figure 3.26: Decompression time of small, medium, large, and very large sized messages using One-bit status technique

strategies to create a reduced size as well as re-constructable XML expression by removing the duplicated tags. First technique is mainly based on developing new Binary-tree based transformation to the XML tree and then assigning the generated binary tree with four bit code combination in order to generate a re-constructable SOAP messages expression. Second technique (Two-bit technique) is based on assigning all the XML tags with one bit code "0" if it is not a data leaf item and two bit code with two values "10" and "11" if it is a data leaf item. This technique is utilizing the features of the depth-first traversal for XML tree. The third technique (One-bit technique) assigns all the tags and data leaf items with one bit code "0" or "1", and it is mainly based on the breadth-first traversal for XML tree.

Fixed length and Huffman as a variable length encodings are proposed to encode the resultant XML expression of the assignment techniques. Experiments show that the proposed techniques outperform all the existing methods by achieving higher compression ratios. Furthermore, The required compression and decompression time for the proposed techniques are found to be potentially supportive for both low connectivity devices over the Internet and bandwidth constrained communication environment.

While Huffman showed a significant average of compression ratio in comparison with other considered techniques, fixed-length encoding dose not perform as well for very large samples. Fixed length encoding is found to be efficient for small and medium document size outperforming all other techniques including variable length encoding. However, Huffman encoding is the most efficient for large and very large documents.

Chapter 4

Redundancy-aware SOAP Messages Similarity & Aggregation

This chapter tries to answer the second research question posted in section 1.2. Simple cost similarity measurements such as Jaccard coefficient and cosine similarity measurements have been developed in this chapter to provide the capability of grouping more than two similar messages in one cluster. Then, new redundancy-aware aggregation models are proposed with the aim to potentially reduce the total SOAP network traffic.

The chapter is organized as follows: Section 4.1 discusses the main drawbacks of SOAP, motivation for this research, and briefly describes the proposed solution. Section 4.2 introduces a predictor for compressibility measurements for SOAP messages. Then, states the development of Jaccard based clustering technique, and finally explains the Vector space model and its clustering model for grouping SOAP messages based on their cosine similarity degrees. Section 4.3 shows the technical strategy for utilizing the compression concepts in the aggregation process with the proposed Binary-tree, Two-bit, and One-bit aggregation models. The evaluation of the proposed techniques is depicted in section 4.4. Finally, section 4.5 concludes the chapter.

4.1 Introduction

SOAP has been developed to improve interoperability of Web services in Cloud environments [Xiong and Perros, 2009; Miyamoto et al., 2009]. However, Cloud Web services inherit the disadvantages of SOAP as messages are bigger than the real payload of requested services [Stantchev, 2009], which can cause high network traffic. As a result, Cloud Web services often suffer from congestion and bottlenecks due to the high number of client Web requests and the large size of Web messages [Vecchiola et al., 2009]. This could result in slowing down the performance of the Cloud Web applications or halting them completely [Miyamoto et al., 2009; Mancini et al., 2009].

4.1.1 Motivation

Several approaches have been proposed to develop compression techniques and only few for textual aggregation models as an efficient solution that could improve the performance of Web services significantly by minimizing the network traffic. Despite the fact they are capable of enhancing the performance of web services to some extent, they still suffer technical drawbacks that could affect their overall performance like high storage consumption. Although both compression and aggregation models have similar objectives and exploit similarity within the message itself (redundancy in compression) or with other messages (similarity in aggregation), they have failed to take advantage of each other to achieve higher performance.

Furthermore, aggregation technically needs the support of SOAP similarity measurement schemes to enable aggregating messages with potential size reduction through utilizing the high similarity degree inside the group of the considered messages. Generally, the existing simple cost SOAP similarity measurements such as Jaccard coefficient work on pairs of messages which more advance measurements are required to strengthen the performance of the aggregation models.

4.1.2 Proposed solution

In this chapter, three aggregation models are introduced that are based on utilizing the compression concepts to exploit the redundancies of SOAP messages. The basic objective of the proposed model is to provide an efficient aggregation that can reduce the size of the messages significantly. The compression techniques proposed in chapter 3 are developed in this chapter as redundancy-aware aggregation models. Binary-tree, Two-bit and One-bit status XML tree aggregation techniques aim to enable the Cloud application servers to aggregate a group of messages that have a certain degree of similarity and send them as one compact message in order to minimize the network traffic (see Fig. 4.1). The resultant aggregated messages of SOAP responses are extractable at the closest routers to the receivers (clients) to deliver only the required response to these clients.

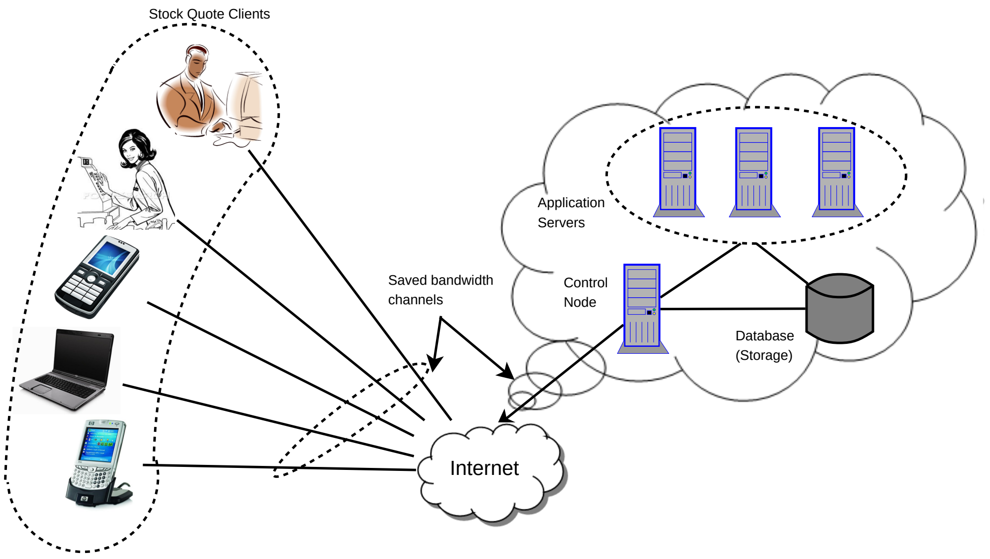


Figure 4.1: Cloud Web services scenario for a Stock Quote

Three similarity measurements of SOAP messages are introduced in order to investigate the similarity based clustering model that can group messages with a significant similarity degree to enable the aggregation techniques to achieve potential message size reduction. Compressibility measurement, Jaccard coefficient, and Vector Space Model are proposed in order to cluster SOAP messages based on their similarity. Compressibility is developed to predict the possibility of size reduction that can be achieved on SOAP message pairs. Jaccard coefficient and Vector Space Model are developed to group SOAP messages in larger predefined size clusters (not only pairs).

4.2 Similarity measurements and clustering of SOAP messages

Similarity-based clustering of SOAP messages represent a defacto operation for aggregation approaches by clustering messages with a high level of similarity to strengthen aggregation resulting in high size reduction. In this chapter, we first introduce compressibility for pairs of SOAP messages as a simple and effective similarity measurement tool to support the proposed compression based aggregation technique by computing the compressibility of messages. Jaccard coefficients are well-known for computing similarity of pairs of messages [Phan et al., 2008], and next, we exploit this feature to build a new clustering algorithm with n-message (where $n \geq 2$) sized clusters. Furthermore, another new clustering technique based on Vector Space Model is proposed as a fixed cluster size technique in order to exploit the highest similarity that can be achieved in group of messages.

4.2.1 Compressibility measurements

Compressibility measurement is proposed in this chapter as an alternative to the traditional similarities of SOAP messages by considering the redundancy within messages. In fact, clusters of SOAP message measurements determine the compressible SOAP Web messages that have common redundancy and can be combined efficiently with the aim of achieving high size reduction. As the proposed aggregation technique is a redundancy based model, the size of the Web messages is an effective criteria in predicting the potential reduction of the aggregated message size. Hence, the compressibility measurements consider the Web message size as a basic parameter as well as computing the overlapped ratio of the XML

$< StockQuoteResponse >$

$< ArrayOfStockQuote >$

$< StockQuote >$

$< Company > IBM < /Company >$

$< QuoteInfo >$

$< Price >$ 22.36 $< /Price >$

$< LastUpdated >$ 06/05/2010 $< /LastUpdated >$

$< /QuoteInfo >$

$< /StockQuote >$

$< StockQuote >$

$< Company > HP < /Company >$

$< QuoteInfo >$

$< Price >$ 21.54 $< /Price >$

$< LastUpdated >$ 06/05/2010 $< /LastUpdated >< /QuoteInfo >$

$< /StockQuote >$

$< /ArrayOfStockQuote >$

$< /StockQuoteResponse >$

a. S_1

$< StockQuoteResponse >$

$< ArrayOfStockQuote >$

$< StockQuote >$

$< Company > NAB < /Company >$

$< QuoteInfo >$

$< Price >$ 26.47 $< /Price >$

$< LastUpdated >$ 06/05/2010 $< /LastUpdated >$

$< /QuoteInfo >$

$< /StockQuote >$

$< StockQuote >$

$< Company > CommonWealth < /Company >$

$< QuoteInfo >$

$< Price >$ 27.51 $< /Price >$

$< LastUpdated >$ 06/05/2010 $< /LastUpdated >< /QuoteInfo >$

$< /StockQuote >$

$< /ArrayOfStockQuote >$

$< /StockQuoteResponse >$

b. S_2

Figure 4.2: a. (S_1):SOAP response to the getStockQuote(IBM, HP) request, b.(S_2):SOAP response to the getStockQuote(NAB, CommonWealth) request

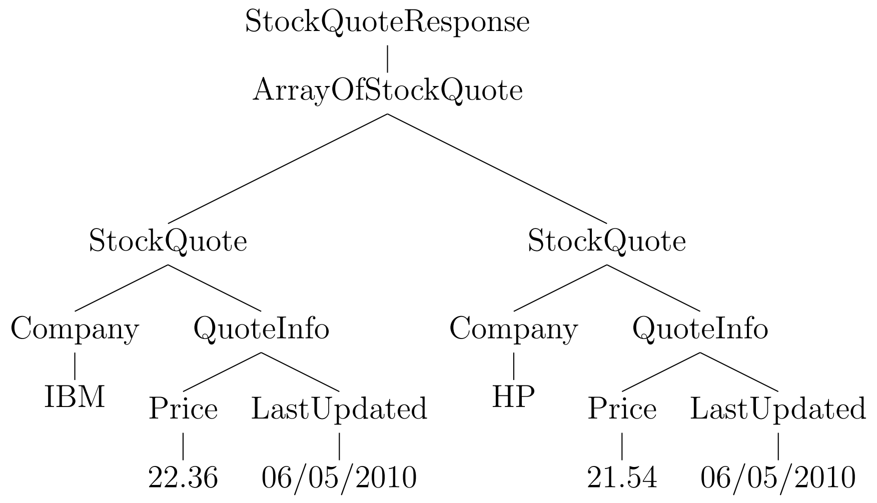


Figure 4.3: Generated XML message tree for S_1 SOAP messages

tags between messages. Equation 4.1 is required for computing the overlapping ratio for a set of SOAP messages $(S_1, S_2, ...S_N)$.

$$Ov(S_1, S_2, ...S_N) = \frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} Sh(S_i, S_j)}{\sum_{i=1}^{N} Tot(S_i)} \tag{4.1}$$

Where

- $Sh(S_i, S_j)$ is the number of common XML tags and data items in both messages S_i and S_j.

- $Tot(S_i)$ is the total number of XML tags and data items in message S_i.

Equation 4.2 computes the overlapping $Ov(S_1, S_2)$ ratio of only two SOAP messages S_1 and S_2.

$$Ov(S_1, S_2) = \frac{Sh(S_1, S_2)}{Tot(S_1) + Tot(S_2)} \tag{4.2}$$

Equation 4.3 is required to compute the overall compressibility measurement $Cm(S_1, S_2)$ of messages S_1 and S_2.

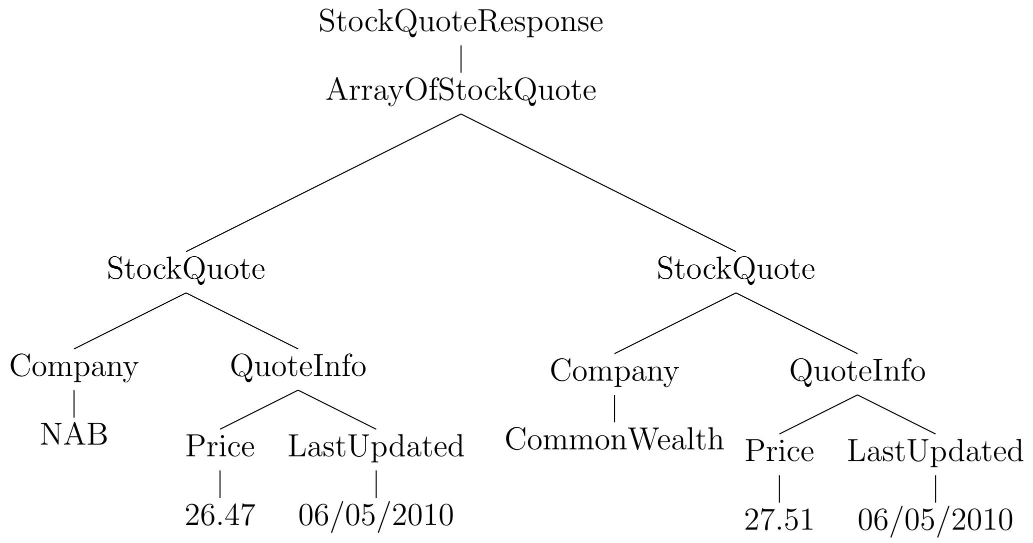


Figure 4.4: Generated XML message tree for S_2 SOAP messages

$$Cm(S1, S2) = Ov(S_1, S_2) \times Log(Tot(S_1, S_2)) \tag{4.3}$$

Where

- $Ov(S_1, S_2)$ is the overlapping ratio between two messages S_1 and S_2.
- $Tot(S_1, S_2)$ is the total number of XML items in both S_1 and S_2 messages.

For the given SOAP messages S_1 and S_2 in Fig. 4.2, the shared common nodes ($Sh(S_1, S_2)$) is 14 as it can be computed from generated matrix form of SOAP messages (see Fig. 4.5). The total nodes ($Tot(S_1)+Tot(S_2)$) in both messages is 36. Therefore, the overlapping ratio can be computed as:

$$Ov(S_1, S_2) = \frac{14}{36} = 0.388 \tag{4.4}$$

Then, the overall compressibility of messages S_1 and S_2 can be computed as:

$$Cm(S_1, S_2) = 0.388 \times Log(36) \tag{4.5}$$

$$Cm(S_1, S_2) = 0.388 \times 1.556 = 0.61 \tag{4.6}$$

4.2.2 Jaccard messages grouping

The Jaccard similarity coefficient is a statistical factor that is usually used for comparing the similarity and diversity of XML messages [Wang and hua Li, 2009]. The Jaccard coefficient is defined as the size of the intersection of two XML messages divided by the size of the union of the same messages [Chung et al., 2010]. Similar messages are determined by computing their similarities of both non-leaf and leaf nodes of all the XML trees. For non-leaf nodes, the Jaccard similarity is computed as the ratio of common nodes between two messages.

$$Jc_{temp}(S1, S2) = \frac{|N_{nd}(S1) \cap N_{nd}(S2)|}{|N_{nd}(S1) \cup N_{nd}(S2)|} \tag{4.7}$$

Where

- $N_{nd}(XMLtree)$ is a set of distinctive non-leaf XML nodes; and
- $|X|$ is the cardinality of the set X

In this chapter, the same Jaccard coefficient equation is modified to compute the similarity of the XML messages for leaf nodes only.

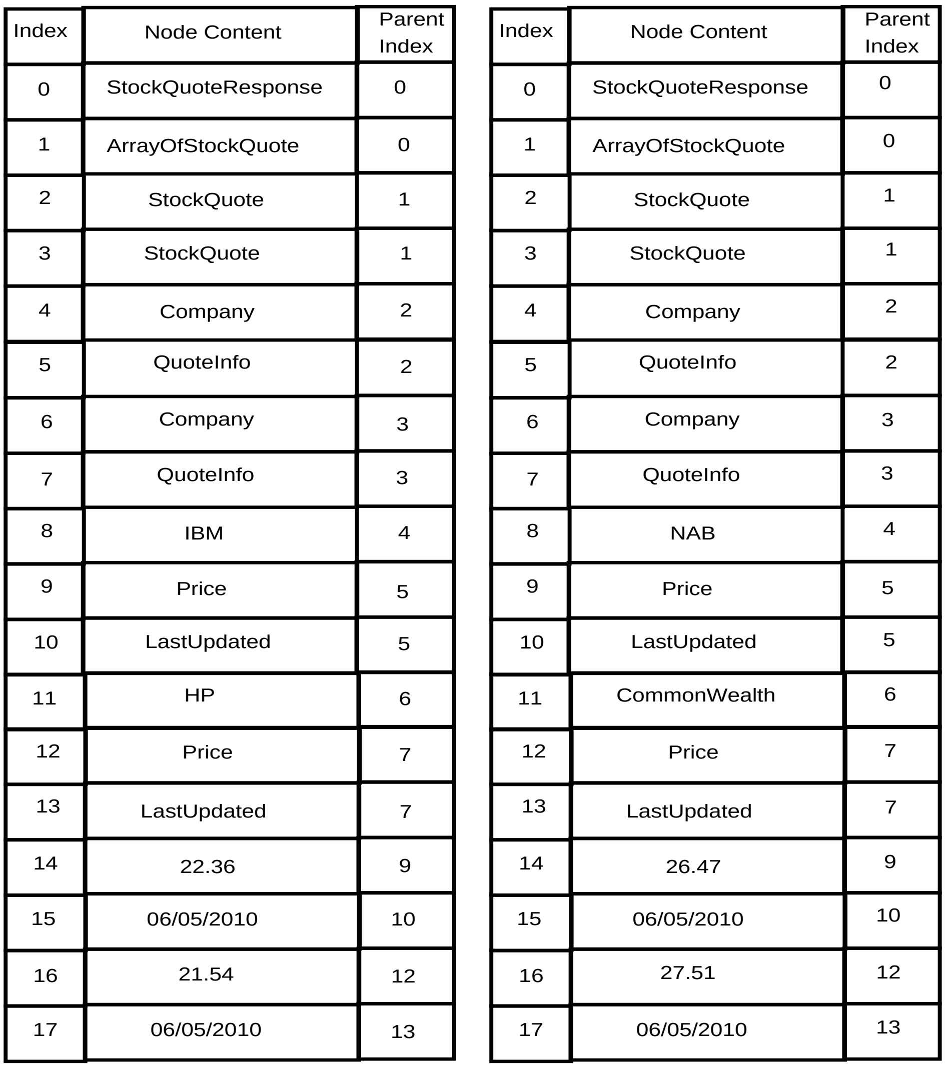
$$Jc_{leaf}(S_1, S_2) = \frac{|N_{ch}(S_1) \cap N_{ch}(S_2)|}{|N_{ch}(S_1) \cup N_{ch}(S_2)|} \tag{4.8}$$

Where

- $N_{ch}(XMLtree)$ is a set of the distinctive characters of the leaf XML nodes; and

The similarity between two service messages S_1 and S_2 is computed using the following equation:

$$Sim_{Jc}(S_1, S_2) = Jc_{temp}(S_1, S_2) \times N_x + Jc_{leaf}(S_1, S_2) \times N_l \tag{4.9}$$



Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	ArrayOfStockQuote	0
2	StockQuote	1
3	StockQuote	1
4	Company	2
5	QuoteInfo	2
6	Company	3
7	QuoteInfo	3
8	IBM	4
9	Price	5
10	LastUpdated	5
11	HP	6
12	Price	7
13	LastUpdated	7
14	22.36	9
15	06/05/2010	10
16	21.54	12
17	06/05/2010	13

a.

Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	ArrayOfStockQuote	0
2	StockQuote	1
3	StockQuote	1
4	Company	2
5	QuoteInfo	2
6	Company	3
7	QuoteInfo	3
8	NAB	4
9	Price	5
10	LastUpdated	5
11	CommonWealth	6
12	Price	7
13	LastUpdated	7
14	26.47	9
15	06/05/2010	10
16	27.51	12
17	06/05/2010	13

b.

Figure 4.5: Generated matrix form for two SOAP messages (a.) S_1 and (b.) S_2

Where

- N_x is the total number of non-leaf XML nodes; and
- N_l is the total number of leaf XML nodes.

Furthermore, Jaccard similarity measurement is proposed in this chapter as a simple grouping technique of XML messages that are clustered into equally sized groups (more than two messages) based on their Jaccard similarities. Algorithm 4.1 is required to create the XML message groups in order to enable the proposed models to aggregate messages according to the resultant Jaccard clustered groups. In this algorithm, the centroids are selected based on the first available point (i.e. unclustered message). Firstly, the messages are flagged with a Boolean value (initially 'true') to assign all points as either still available and waiting to be clustered or already clustered to one of the generated groups based on the similarity to the nominated centroids. After selecting the first available centroid, the Jaccard similarity is computed with the remaining available points. Then, the clustered points are selected according to their high degree of similarity with the considered centroid. The complexity of algorithm 4.1 is $O(n^2 \times m)$, where n is the number of the considered documents (Sn) and m is the number of groups of items (Gn) in each message.

4.2.3 Vector space messages grouping

The vector space model is one of the well-known techniques in information retrieval and textual documents clustering [Liu et al., 2010]. It is mainly based on computing the cosine similarities of documents in order to investigate their similarity degree [Chen and Song, 2009]. The basic cosine similarity of the Vector space model involves computing the items weight of the documents which reflect their descriptiveness in a statistical way [Liu et al., 2010]. In this chapter, the Vector space model is proposed as similarity measurement for SOAP messages as it is developed to cluster them into equal sized groups. Equation 4.10 measures the similarity between two SOAP messages S_1 and S_2:

Algorithm 4.1 *Jaccard Clustering*

```
01://**Notation Description:**
02://$Sn$ holds the number of documents
03://$Gst$ holds the resultant groups
04://$Gn$ holds the number of groups
05:
06:**For** $i = 1$ **To** $Sn//Initializing\ flags$
07:  $VFlag(i) \leftarrow True$
08:**Next** $i$
09:$Gst \leftarrow ""//Initializing\ the\ resultant\ groups$
10:$Gn \leftarrow \frac{Sn}{Gs}$ $//Number\ of\ groups$
11:**For** $G_{index} = 1$ **To** $Gn$
12:  **For** $i = 1$ **To** $Sn$ $//Determine\ the\ next\ center\ document$
13:    **if** $VFlag(i) = True$ **then**
14:      $C \leftarrow i$
15:      $Gst \leftarrow concatenate(Gst, C)$
16:      *exit loop*
17:    **end if**
18:  **Next** $i$
19:  **For** $i = C + 1$ **To** $Sn$ $//Compute\ the\ Jaccard\ similarities$
20:    **if** $VFlag(i) = True$ **then**
21:      $Sim_{Jc}(i) = Jc_{temp}(S_C, S_i) \times N_x + Jc_{leaf}(S_C, S_i) \times N_l$
22:    **end if**
23:  **Next** $i$
24:  **For** $i = 1$ **To** $Gs//Find\ the\ closest\ documents$
25:    $Max_{index} \leftarrow 0$
```

....**Continue**

```
26:    $//Initializing\ the\ closest\ document\ index$
27:    For $j = C + 1$ To $Sn$
28:      if $(Max_{index} = 0)$ then
29:        $Max_{index} \leftarrow j$
30:      else
31:        if $(VFlag(j) = True)$ and $(Sim_{Jc}(j) > Sim_{Jc}(Max_{index}))$ then
32:          $Max_{index} \leftarrow j$
33:        end if
34:      end if
35:    Next $j$
36:    $Gst \leftarrow concatenate(Gst, Max_{index})$
37:    $VFlag(Max_{index}) = False$
38:  Next $i$
39:  $Gst \leftarrow concatenate(Gst, "\&")//End\ of\ Group$
40:Next $G_{index}$
```

$$Sim_{VS}(S_1, S_2) = \frac{W_{S_1}.W_{S_2}}{\|W_{S_1}\| \, \|W_{S_2}\|} \tag{4.10}$$

Where

- W_{S_1} and W_{S_2} are vectors that include the XML document items weight of messages S_1 and S_2 respectively.
- $\|W_{S_1}\|$ and $\|W_{S_2}\|$ represent the resultant norm values of the weight vectors of both messages S_1 and S_2 respectively.

Therefore, the cosine similarity equation can be described in details as shown in equation 4.11:

$$Sim_{VS}(S_1, S_2) = \frac{\sum_{i=1}^{N} W_{S_1}(i) \times W_{S_2}(i)}{\sqrt{\sum_{i=1}^{N} W_{S_1}(i)} \times \sqrt{\sum_{i=1}^{N} W_{S_2}(i)}} \tag{4.11}$$

The Vector space model is developed to cluster SOAP messages according to their cosine similarities. Algorithm 4.2 is required to generate the Vector space based SOAP clusters. First, the algorithm determines the centers of documents by computing the summations of the frequencies of the weighted XML items for each vector and sorting them in descending order and then cluster them based on the required group size using the values of summations as the key of their distribution. The first XML document of each cluster is assigned as a center point. Then, the rest of the XML documents are grouped into their clusters based on the similarity degree with the document that is being compared. Again, the points (messages) are initially flagged as available using Boolean (initially 'true') and then the centroids are excluded as they are flagged 'false'. For every single centroid, its cosine similarities with the rest available points are investigated to be clustered with the considered centroid. Algorithm 4.1 complexity is $O(n^2) + O(m)$, where n is the number of SOAP messages (Sn) and m is the number of groups of XML items (Gn) in each message.

4.3 Compression based aggregation of SOAP messages

Generally, compression techniques have been used to enhance the performance of Web services by reducing the overall size of SOAP messages over the net to minimize the network traffic. Technically, compression techniques are based on exploiting the redundant items inside the same object. In this chapter, the targeted redundancy of compression techniques is extended to be exploited with other objects (SOAP messages) to be the key factor of a new aggregation strategy powered by the compression concepts. The new proposed aggregation models would enable Cloud Web servers to aggregate SOAP responses using compression targets to reduce the aggregated messages size efficiently.

Algorithm 4.2 *Vector Space Clustering*

```
01://**Notation Description:**
02://$Sn$ holds the number of documents
03://$Gst$ holds the resultant groups
04://$Gn$ holds the number of groups
05:
06:$Gst \leftarrow$ ""//$Initializing\ the\ resultant\ groups$
07:$Gn \leftarrow \frac{Sn}{Gs}$//$Number\ of\ groups$
08:$Cn \leftarrow Gn$//$Number\ of\ centers$
09:**For** $i = 1$ **To** $Sn$//$Sum\ Weights\ of\ Samples$
10:  $SumW(i) \leftarrow \sum_{j=1}^{N} W_{S_i}(j)$
11:**Next** $i$
12:$TCnt(1..N) \leftarrow Ascending\ Sort(SumW(1..N))$
13:    //$Sorting\ weight\ vector$
14:$K \leftarrow 1$//$Initializing\ the\ centers\ index$
15:**For** $i = 1$ **To** $Cn$ //$Determine\ centers\ index$
16:  $Cnt(i) \leftarrow K$
17:  $K \leftarrow K + Gs$
18:**Next** $i$
19:**For** $i = 1$ **To** $Sn$//$Initializing\ flags$
20:  $VFlag(i) \leftarrow True$
21:**Next** $i$
22:**For** $i = 1$ **To** $Cn$//$Exclude\ centers$
23:  $VFlag(Cnt(i)) \leftarrow False$
24:**Next** $i$
25:**For** $i = 1$ **To** $Gn$//$Clustering\ documents$
```

....**Continue**

```
26: $C \leftarrow Cnt(i)//Get\ center\ index$
27: $Gst \leftarrow concatenate(Gst, C)$
28: **For** $j = 1$ **To** $Sn//Compute\ similarities$
29:   **if** $VFlag(j) = True$ **then**
30:     $Dp \leftarrow \sum_{k=1}^{N}[W_{S_C}(k) \times W_{S_j}(k)]$
31:     $Np \leftarrow SumW(C) \times SumW(j)$
32:     $Sim_{VS}(j) \leftarrow \frac{Dp}{Np}$
33:   **end if**
34: **Next** $j$
35: $T_{Sim}(1..N) \leftarrow Descending\ Sort(Sim_{VS}(1..N))$
36: $//Sorting\ similarities$
37: **For** $j = 1$ **To** $Gs - 1//Include\ closest\ documents$
38:   $Gst \leftarrow concatenate(Gst, T_{Sim}(j))$
39: **Next** $j$
40: $Gst \leftarrow concatenate(Gst, "\&")//End\ of\ cluster$
41:**Next** $i$
```

In this chapter, the compression models proposed in chapter three are developed to generate the compact aggregated messages by first building XML trees, then investigating their compressibility or their similarities based on Jaccard measurements or Vector Space Model (VSM) and finally encoding them using the Binary-tree, Two-bit, and One-bit compression based aggregation process which start by computing the textual SOAP expression and then make use of fixed-length or Huffman (as a variable-length) encoding.

4.3.1 Generation of SOAP message expression

It is required to generate the minimized XML textual expression (using the XML tree) in such a way that guarantees rebuilding the XML tree again in order to regenerate the original SOAP message. In the aggregation models, the same traversals (binary-tree, depth-first, and breadth-first) of the proposed compression techniques are used to generate the minimized XML textual expression by assigning all the tags with some binary codes to enable rebuilding the XML tree by recognizing the correct position of each tag.

Equation 4.12 represents the general formula of the proposed traversals in assigning the XML items to generate the textual expression for SOAP messages.

$$T_{EXP} = \bigcup_{i=1}^{Nd} B_i Tag_i \tag{4.12}$$

Where

- B_i is the binary code value of the considered XML textual items.
- Tag_i is the assigned XML textual item.
- Nd is the total number of both complex and simple XML items ($N_c + N_s$).

Binary-tree traversal

The generated SOAP expression using the Binary-tree traversal of S_1 is: {***10****StockQuoteResponse* ***10****ArrayOfStockQuote* ***11****StockQuote* ***11****Company* ***00****IBM* ***10****QuoteInfo* ***11****Price* ***00****22.36* ***10****LastUpdated* ***00****06/05/2010* ***10****StockQuote* ***11****Company* ***00****HP* ***10****QuoteInfo* ***00****Price* ***00****21.54* ***10****LastUpdated* ***00****06/05/2010*} and for S_2 is {***10****StockQuoteResponse* ***10****ArrayOfStockQuote* ***11****StockQuote* ***11****Company* ***00****NAB* ***10****QuoteInfo* ***11****Price* ***00****26.47* ***10****LastUpdated* ***00****06/05/2010* ***10****StockQuote* ***11****Company* ***00****CommonWealth* ***10****QuoteInfo* ***00****Price* ***00****27.51* ***10****LastUpdated* ***00****06/05/2010*}

Depth-first traversal

The resultant textual expression using the depth-first traversal of S_1 can be represented as follow. {***0****StockQuoteResponse* ***0****ArrayOfStockQuote* ***0****Stock Quote* ***0****Company* ***10****IBM* ***0****Quote-Info* ***0****Price* ***10****22.36* ***0****LastUpdated* ***11****06/05/2010* ***0****StockQuote* ***0****Company****10****HP* ***0****Quote-Info****0****Price****10****21.54****0****LastUpdated* ***11****06/05/2010*}. Similarly, message S_2 can be expressed as: {***0****StockQuoteResponse* ***0****ArrayOfStockQuote* ***0****StockQuote* ***0****Company* ***10****NAB* ***0****QuoteInfo* ***0****Price* ***10****26.47* ***0****LastUpdated* ***11****06/05/2010* ***0****StockQuote* ***0****Company* ***10****CommonWealth* ***0****QuoteInfo* ***0****Price* ***10****27.51* ***0****LastUpdated* ***11****06/05/2010*}

Breadth-first traversal

The textual expression using breadth-first traversal of S_1 would be as follow. {***0****StockQuoteR-esponse* ***1****ArrayOfStockQuote* ***0****StockQuote* ***1****StockQuote* ***0****Company* ***1****QuoteInfo* ***0****Company* ***1****QuoteInfo* ***1****IBM* ***0****Price* ***1****LastUpdated* ***1****HP* ***0****Price* ***1****LastUpdated* ***1****22.36* ***1****06/05/2010* ***1****21.54* ***1****06/05/2010*} and for S_2 would be expressed as: {***0****StockQuoteResponse* ***1****ArrayOfS-tockQuote* ***0****StockQuote* ***1****StockQuote* ***0****Company* ***1****QuoteInfo* ***0****Company* ***1****QuoteInfo* ***1****NAB* ***0****Price* ***1****LastUpdated* ***1****CommonWealth* ***0****Price* ***1****LastUpdated* ***1****26.47* ***1****06/05/2010* ***1****27.51* ***1****06/05/2010*}.

4.3.2 Aggregation process of SOAP expressions

Encoding of XML textual expressions is the core component of the proposed aggregation models that would generate the final compact version of the considered messages. Fixed and variable length encoding techniques are proposed to generate the aggregated compact message of the combined textual SOAP expressions. Both encodings are well-known as lossless compression techniques that can remove the redundancies of letters by assigning binary codes for these letters. The resultant encoded message structure has two parts: the lookup table which includes unique content for every single item in the XML textual expression while the second part includes the binary codes of the encoded messages. SOAP messages are aggregated during the encoding process by generating one common lookup table for all the considered SOAP expressions.

Figure 4.6 shows the structure of both the individually compressed messages. In comparison with the new structure of the aggregated messages, Fig. 4.7 shows the structure of the common look up table that is shared with all the aggregated SOAP expressions which reduce the required size by removing the duplicated occurrences and keeping only one occurrence in the look up table. Figure 4.8 shows the technical strategy of the aggregation process of the textual expressions for two SOAP messages. The size of the aggregated message lookup table is smaller than the accumulated size of lookup tables of the individually compressed messages as a result of the fact of sharing the common XML items of different messages in generating one common lookup table as it increases the probability of occurrences of items. Referring to Fig. 4.7 and 4.6, Common Lookup Table (CLT) is smaller than the accumulated size of both lookup table 1 and lookup table 2 (LT1+ LT2). On the other hand, the size of the binary encoded parts of the aggregated message is smaller than the encoded parts of the compressed messages. It is mainly based on the lengths of the binary codes of the generated mappings for the XML items in the lookup table as they are encoded with fewer bits in the common lookup table.

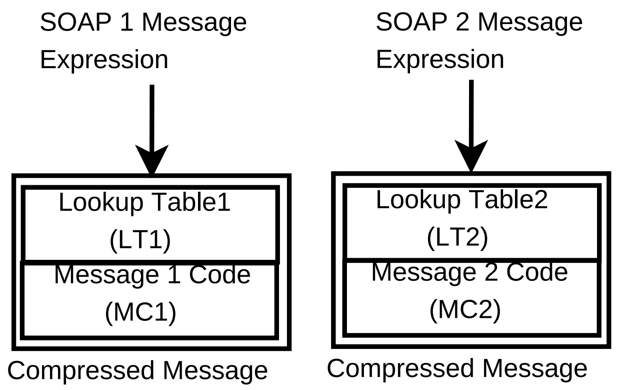


Figure 4.6: Compressed message structures

Fixed-length and Huffman are extended to exploit the redundancy inside the single message itself in addition to the redundancy with other involved messages. Both encoding replace XML items with unique binary codes based on their total frequencies in all the considered messages. Table 4.1 shows the redundancies and binary codes for the XML items in both expressions of S_1 and S_2 messages.

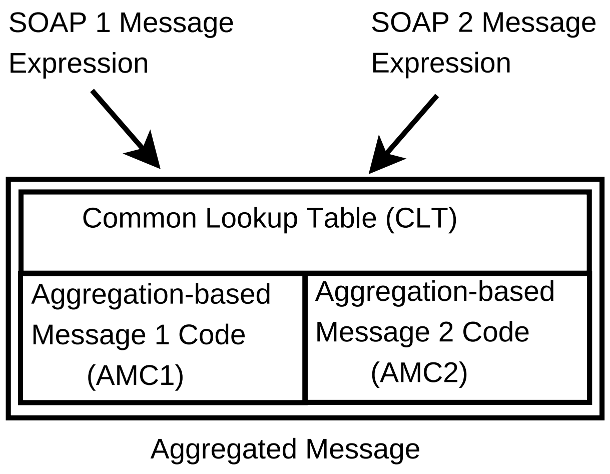

SOAP 1 Message Expression
SOAP 2 Message Expression
Common Lookup Table (CLT)
Aggregation-based Message 1 Code (AMC1)
Aggregation-based Message 2 Code (AMC2)
Aggregated Message

Figure 4.7: Aggregated message structures

4.4 Experiments and discussion

In the evaluation of the proposed aggregation models, we have considered the same variety of SOAP message sizes used in chapter three that range from only 140 bytes to 53 Kbytes in order to show the efficiency of the models on small messages as well as large ones. The objective of considering small messages is to investigate the fact that lossless encodings usually create large lookup table in comparison with the encoded part of the input message. It could cause in many cases an even larger encoded message than the uncompressed one. At the same time, this evaluation shows an accurate investigation for both fixed-length and Huffman encodings of Binary-tree, Two-bit, and One-bit techniques against other standard compression techniques (i.e gzip, bzip2, XMill, and XBMill).

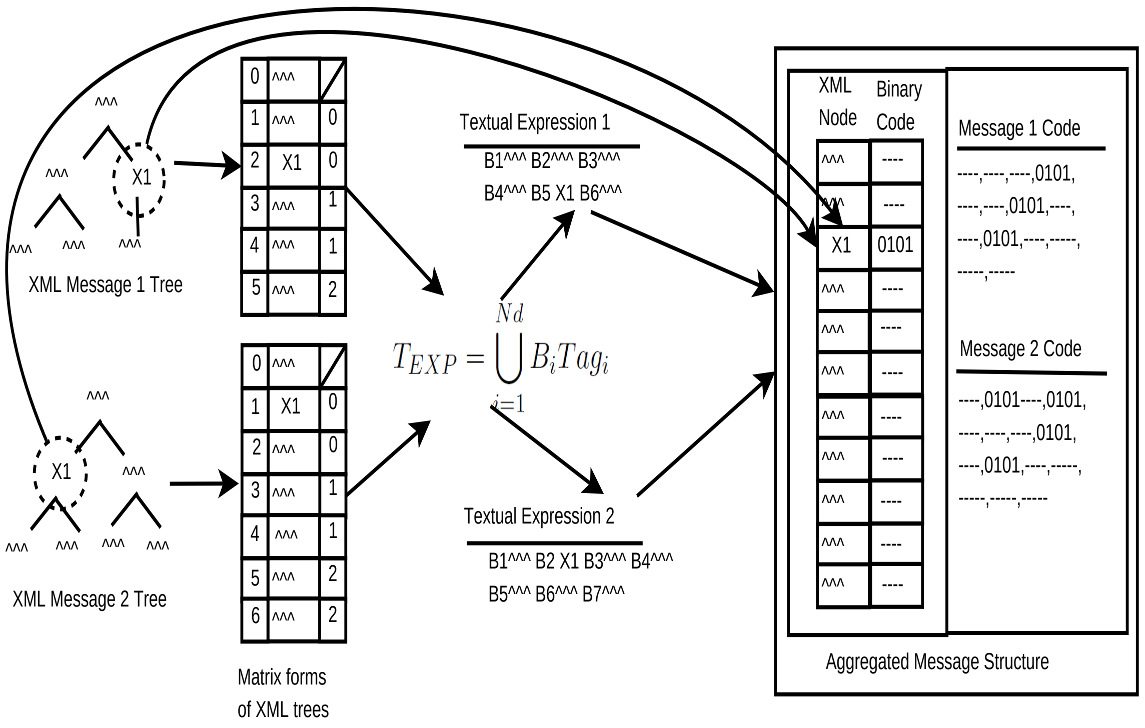


Figure 4.8: SOAP Web message aggregation strategy and compact message structure

The same testbed used in chapter 3 is used again in this chapter that consists of 160 real SOAP Web messages that are distributed equally into four groups based on the message size: small (140-800 bytes), medium (800-3000 bytes), large (3000-20000 bytes), and Very large (20000-55000 bytes). Since single compression concepts are used as basis for the proposed aggregation model, the compression schemes are first applied as standalone techniques and then compared against their developed aggregation models in order to show the ability of the compression schemes in achieving higher reduction in their aggregation process. With the aim to investigate the performance difference of the aggregation techniques from the compression, all of the SOAP messages in the proposed testbed are first divided into two lists ('A' and 'B') and compressed separately using the Two-bit status tree compression technique. Then, the Two-bit status tree aggregation technique is applied on every pair of messages for all groups (small, medium, large, and very large).

Table 4.1: Binary codes of XML nodes for SOAP messages S_1 and S_2

Node Content	**Node Redundancy**	**Huffman Code**	**Fixed-Length Code**
StockQuote	4	100	0000
Company	4	1011	0001
QuoteInfo	4	1010	0010
LastUpdated	4	001	0011
06/05/2010	4	000	0100
Price	4	0011	0101
StockQuoteResponse	2	1101	0110
ArrayOfStockQuote	2	1100	0111
IBM	1	001001	1000
HP	1	001000	1001
NAP	1	001011	1010
CommonWealth	1	001010	1011
22.36	1	11101	1100
21.54	1	11100	1101
26.47	1	11111	1110
27.51	1	11110	1111

Figures 4.9 and 4.10 show the ability of the Two-bit status tree standalone compression technique in compressing SOAP messages using both fixed-length and Huffman encodings. Both encodings show promising results as fixed-length encoding has achieved compression ratios that are up to 2.81, 4, 9.8, and 12.74 for small, medium, large, and very large messages respectively. On the other hand, Huffman encoding has shown similar results for small and medium messages while it has achieved significantly higher compression ratios on large and very large messages, up to 2.9, 4.14, 11.78, and 16.1 for small, medium, large, and very large messages respectively. From these results, we can see that fixed-length encoding performs better on small and medium sized messages.

Furthermore, The developed aggregation versions of the proposed compression techniques are applied using the same set of messages by aggregating SOAP message pairs that belong to the same group (see Fig. 4.11). The results have shown higher compression ratios that are up to 2.93, 5.17, 11.69, and 13.68 using Two-bit status tree based on fixed-length encoding

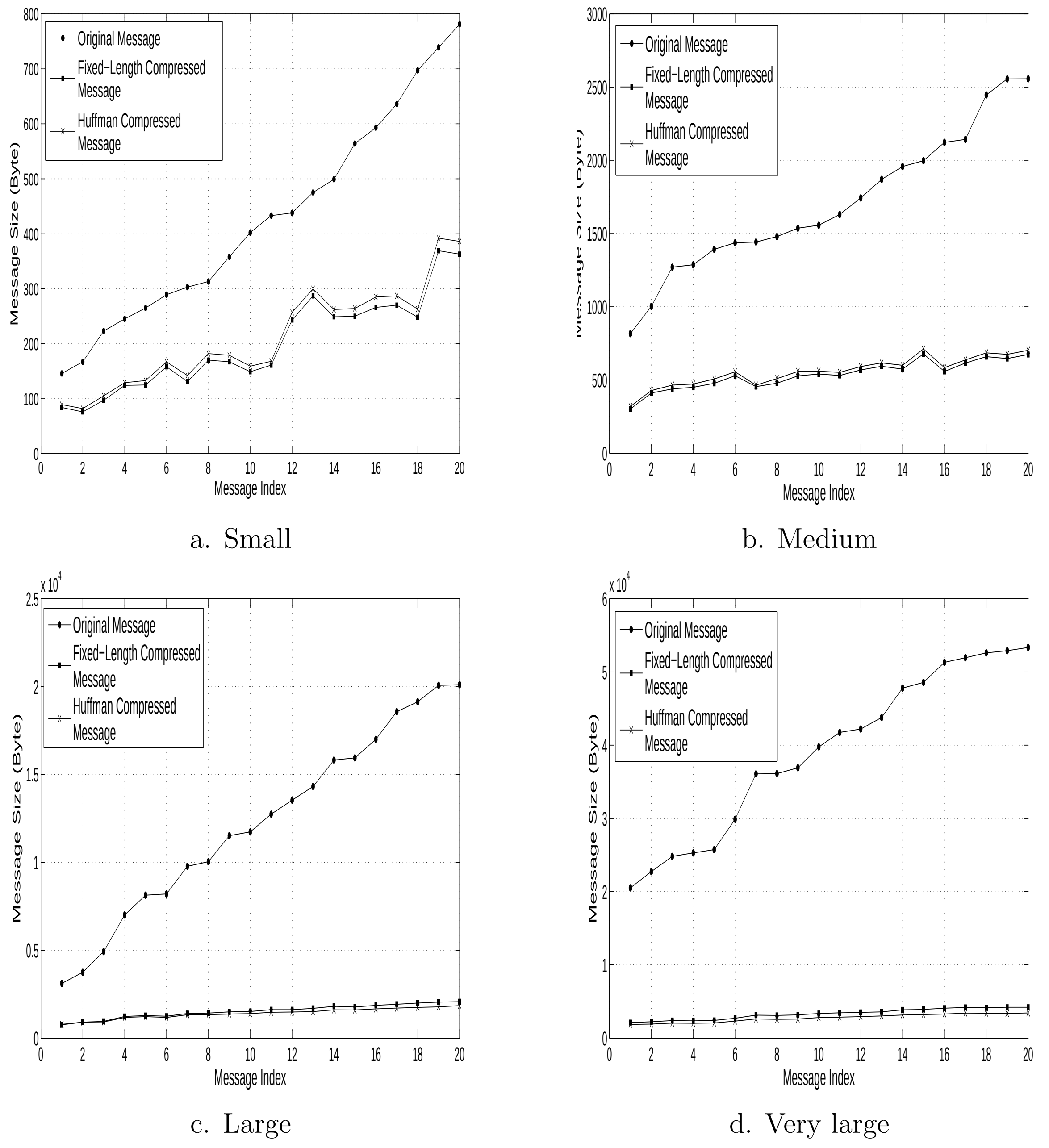
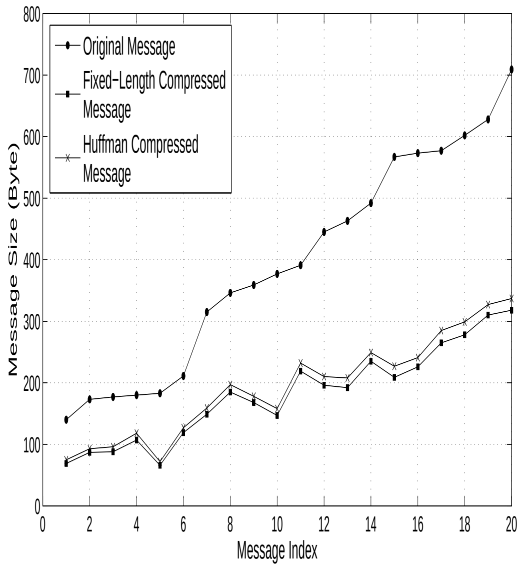
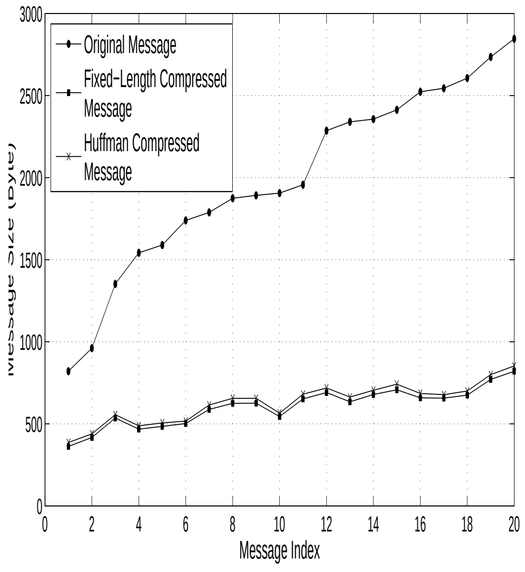


Figure 4.9: Original and compressed size for small, medium, large, and very large sized SOAP messages of List 'A' using Two-bit status tree based fixed-length and Huffman encodings

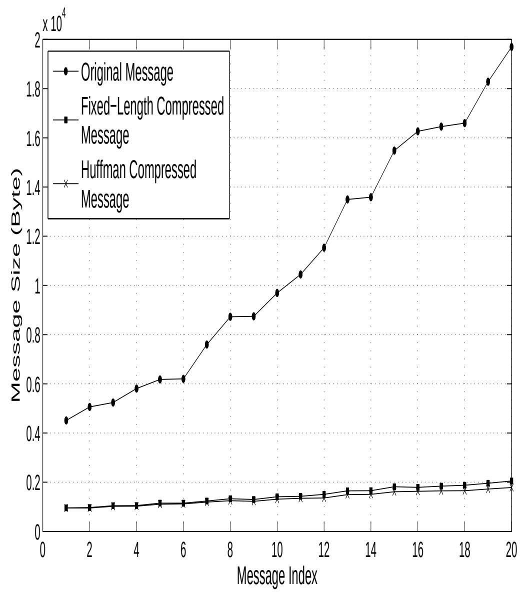
for small, medium, large, and very large messages respectively. Aggregation with Two-bit status tree based on Huffman encoding achieved even higher compression ratios for all types,



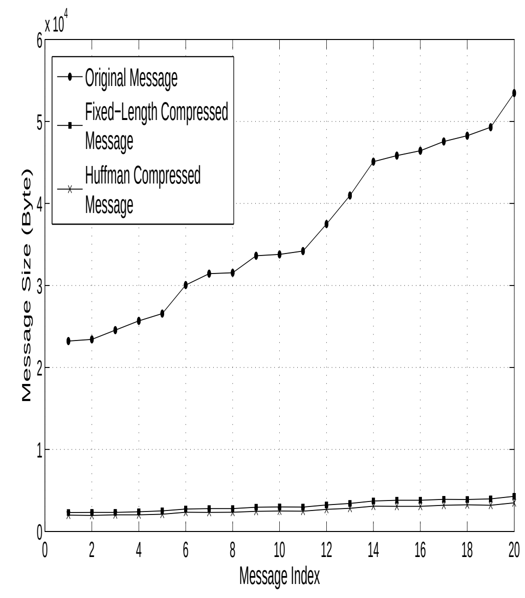
a. Small



b. Medium



c. Large



d. Very large

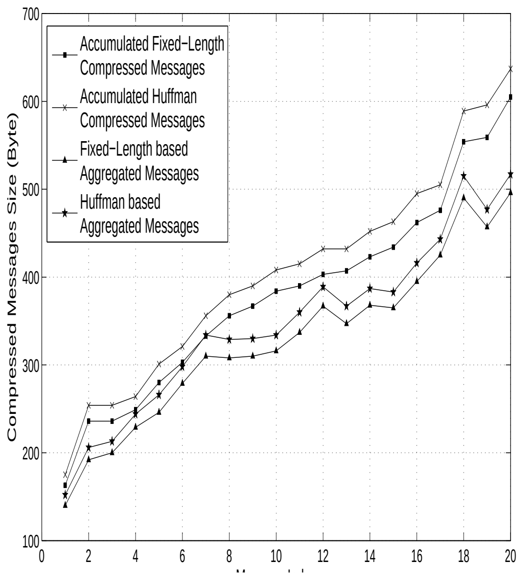
Figure 4.10: Original and compressed size for small, medium, large, and very large sized SOAP messages of List 'B' using Two-bit status tree based fixed-length and Huffman encodings

Table 4.2: Minimum, maximum, and average compression ratios for stand alone compression techniques of fixed and variable length encodings

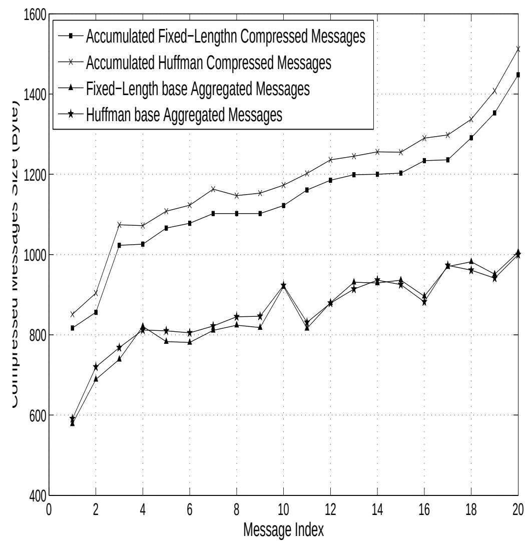
		Standalone					
	Message	**Fixed-Length**			**Huffman**		
	Size	**Min.**	**Max.**	**Average**	**Min.**	**Max.**	**Average**
Binary-tree	**Small**	1.64	2.79	2.14	1.51	2.62	2.00
	Medium	2.25	3.89	3.16	2.10	3.73	3.03
	Large	4.03	9.44	7.16	3.91	10.81	7.78
	V.Large	9.29	12.16	11.08	10.62	14.83	13.2
Two-bit Tree	**Small**	1.66	2.81	2.16	1.68	2.90	2.22
	Medium	2.27	3.96	3.20	2.30	4.14	3.34
	Large	4.09	9.79	7.38	4.3	11.78	8.5
	V.Large	9.64	12.74	11.57	11.57	16.1	14.33
One-bit Tree	**Small**	1.66	2.85	2.18	1.69	2.93	2.24
	Medium	2.28	4.02	3.24	2.32	4.21	3.38
	Large	4.16	10.17	7.60	4.37	12.33	8.80
	V.Large	10.01	13.38	12.12	12.11	17.16	15.17

except for the small group (2.75, 5.22, 13.8, and 17.52 for small, medium, large, and very large messages respectively). Moreover, detailed results are shown in Tables 4.2 and 4.3 on the achievements of both the compression and aggregation models respectively. Clearly, aggregation models significantly outperformed their compression techniques showing the fact of the powerful aggregation process with the utilization of the compression techniques. Furthermore, results have clarified that high compression ratios have been achieved for large and very large documents and aggregation with both fixed-length and Huffman encodings using the proposed techniques have reduced the size of small and medium messages successfully. The results for small SOAP messages are interesting as very few existing standard techniques are capable of reducing small messages. The results also show that the One-bit status tree based aggregation technique has outperformed both Two-bit status tree and Binary-tree based models.

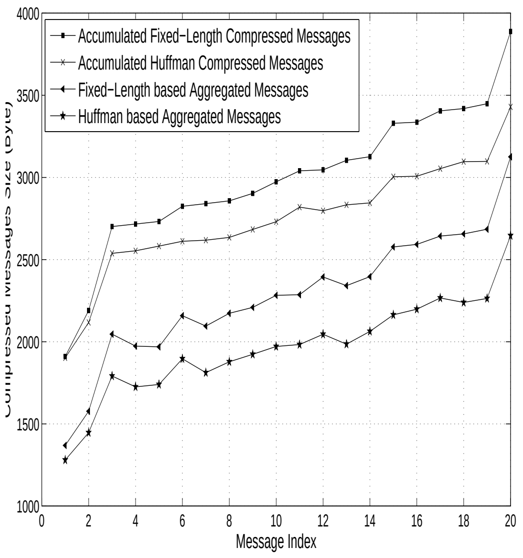
With the aim of showing a precise evaluation for the aggregation techniques, another two lists ('C' and 'D') of SOAP messages are created with 12 samples each and used to



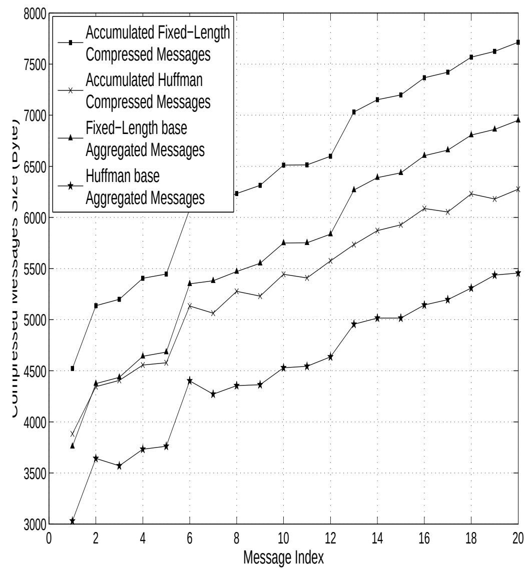
a. Small messages



b. Medium messages



a. Large messages



b. Very large messages

Figure 4.11: Resultant aggregation compact message and accumulated size for compressed messages (i.e pairs) using Two-bit status technique deploying fixed-length and Huffman encodings

Table 4.3: Minimum, maximum, and average compression ratios for aggregation techniques of fixed and variable length encodings

		Aggregation (Pairs of Messages)					
	Message	**Fixed-Length**			**Huffman**		
	Size	**Min.**	**Max.**	**Average**	**Min.**	**Max.**	**Average**
Binary-tree	**Small**	2.67	4.52	3.63	2.53	4.50	3.55
	Medium	4.39	9.02	6.99	4.34	10.16	7.54
	Large	9.64	13.07	11.93	11.07	16.43	14.62
	V.Large	12.62	14.1	13.64	15.70	18.45	17.53
Two-bit Tree	**Small**	2.70	4.60	3.69	2.56	4.59	3.60
	Medium	4.46	9.35	7.19	4.41	10.57	7.79
	Large	10.02	13.76	12.51	11.57	17.55	15.51
	V.Large	13.26	14.90	14.39	16.71	19.85	18.8
One-bit Tree	**Small**	2.73	4.69	3.75	2.59	4.67	3.65
	Medium	4.54	9.69	7.40	4.49	11.01	8.04
	Large	10.42	14.53	13.15	12.12	18.81	16.51
	V.Large	13.97	15.80	15.23	17.85	21.49	20.26

investigate the aggregation models performance against such well known standard compression techniques such as XMILL and XbMILL as XML-aware techniques and gzip in addition to bzip2 as generic techniques. The resultant aggregated size for every pair of messages is compared to the accumulated compressed size of its messages using standard compression techniques in addition to their compression versions. The results shown in Tables 4.4, 4.5, and 4.6 confirm the high performance of the proposed aggregation models in achieving the highest compression ratios in comparison with all other standalone compression techniques. Binary-tree, Two-bit, and One-bit aggregation based on fixed length and Huffman encodings have shown promising achievements for small and medium message pairs. However, Huffman based aggregation technique is shown to be the best performer for large and very large message pairs.

Compressibility measurement is investigated and it has proven that aggregated messages with higher compressibility can be reduced more (see Fig. 4.12) which justifies the need for clustering SOAP messages for the purpose of aggregation. For the aggregation, we cre-

Table 4.4: Accumulated size for compressed SOAP messages using XMILL, XbMILL, gzip, bzip2 techniques

Message Size(Byte)		**Accumulated Compressed Size**			
List C	**List D**	**Gzip**	**bzip2**	**XMILL**	**XBMILL**
636	463	517	563	542	650
358	140	304	316	320	417
265	602	433	470	458	556
1743	2340	1250	1272	1184	1362
2122	1542	1122	1132	1054	1233
1442	820	834	931	854	1024
8129	19697	4845	3584	3659	3666
11516	18285	5069	3744	3783	3758
16997	4510	3936	3052	3045	2951
47800	45085	9653	6063	9377	8621
29876	48257	7448	4972	5247	6574
52899	40961	8332	7559	9470	8603

Table 4.5: Accumulated size for compressed SOAP messages using Fixed and Variable length based Binary-tree, Two-bit, One-bit compression techniques

Message Size(Byte)		**Accumulated Compressed Size**					
		Binary-tree		**Two-bit**		**One-bit**	
List C	**List D**	**Fixed Length**	**Huffman**	**Fixed Length**	**Huffman**	**Fixed Length**	**Huffman**
636	463	466	499	462	495	458	491
358	140	238	257	236	254	233	252
265	602	406	435	403	432	400	429
1743	2340	1219	1270	1203	1255	1187	1239
2122	1542	1042	1086	1026	1072	1011	1057
1442	820	825	860	817	851	809	844
8129	19697	3444	3115	3336	3007	3230	2899
11516	18285	3561	3210	3448	3097	3336	2984
16997	4510	2906	2694	2824	2611	2743	2530
47800	45085	7924	6586	7568	6230	7214	5875
29876	48257	6898	5873	6599	5575	6302	5277
52899	40961	7982	6538	7624	6180	7266	5821

Table 4.6: Aggregated size for SOAP messages using Fixed and Variable length based Binary-tree, Two-bit, One-bit compression techniques

Message Size(Byte)		**Aggregated Messages Size**					
		Binary-tree		**Two-bit**		**One-bit**	
List C	**List D**	**Fixed Length**	**Huffman**	**Fixed Length**	**Huffman**	**Fixed Length**	**Huffman**
636	463	400	420	395	416	391	412
358	140	194	209	192	206	190	204
265	602	371	393	367	389	364	386
1743	2340	952	941	936	925	921	910
2122	1542	836	827	821	812	807	798
1442	820	586	599	578	591	570	583
8129	19697	2699	2305	2592	2198	2485	2090
11516	18285	2798	2376	2685	2263	2573	2150
16997	4510	2240	1978	2158	1896	2077	1815
47800	45085	7161	5665	6806	5309	6451	4954
29876	48257	6135	4934	5837	4636	5539	4338
52899	40961	7220	5794	6861	5436	6503	5077

ate groups of messages which may contain more than two messages for higher compression. Both Jaccard and Vector Space Model based clustering are evaluated in terms of the possible achieving compression ratio by the proposed aggregation techniques in addition to the processing time they require to cluster SOAP messages according to their similarity degrees. Figure 4.13 shows the average clustering time of both Jaccard and Vector Space Models for small, medium, large, and very large messages. Vector Space Model requires less processing time than Jaccard based clustering technique for all messages. The resultant SOAP messages clusters of the proposed clustering models are aggregated by Binary-tree, Two-bit, and One-bit techniques.

Figures 4.14, 4.15, and 4.16 show aggregated message size reduction using One-bit and Two-bit status tree techniques and Binary-tree aggregation respectively with Jaccard based clustered groups of 5 messages per group. Furthermore, Fig. 4.17, 4.18, and 4.19 show aggregated message size reduction using One-bit and Two-bit status tree techniques and

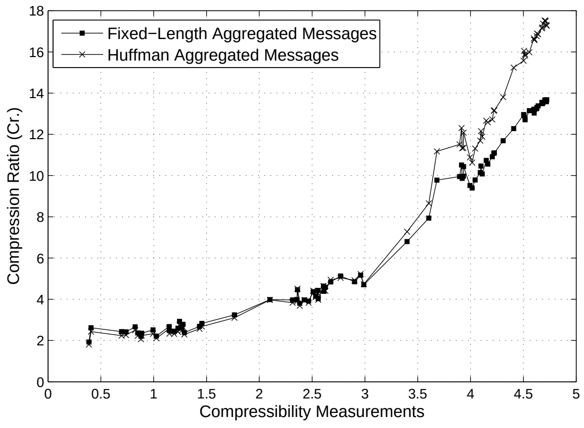


Figure 4.12: Compressibility measurements and compression ratios of aggregated SOAP Web message pairs

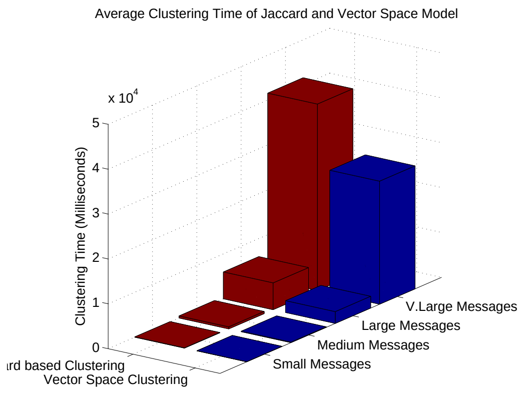


Figure 4.13: Average clustering time in (milliseconds) for Jaccard and Vector Space Model for small, medium, large, and very large messages with 40 messages each

Binary-tree aggregation respectively with Vector Space Model based clustered groups of 5 messages per group. The aggregation models have shown significant messages size reduction. However, the One-bit status tree aggregation technique has achieved the best size reduction in comparison with Two-bit status tree and Binary-tree aggregations.

Figures 4.20 and 4.21 show the average compression ratios that have been achieved using Two-bit and One-bit status tree based aggregation models in addition to the Binary-tree aggregation model for Jaccard and Vector Space Models based clustered messages. SOAP

Table 4.7: The aggregation time in (milliseconds) for Binary-tree, Two-bit, and One-bit techniques for computed SOAP message clusters using Jaccard clustering model

Message Size	**Number of Messages per Cluster**	**Binary-tree**		**Two-bit**		**One-bit**	
		Fixed Length	**Huffman**	**Fixed Length**	**Huffman**	**Fixed Length**	**Huffman**
Small	2	9.2	9.6	0.25	0.65	0.35	0.65
	3	10.43	10.93	0.36	0.86	0.36	0.86
	4	13.3	14.1	0.4	1.3	0.5	1.2
	5	16	17.25	0.63	1.63	0.75	1.63
	6	19.57	20.86	1	2.29	1.29	2.57
	7	22.67	24	1.33	2.83	1.5	3.17
	8	30.8	32.8	1.2	3.4	1.6	3.4
	9	34	36.4	2	3.8	1.8	3.6
	10	37.25	40	2.5	4.5	2.5	4.75
Medium	2	22.65	24.8	3.4	5.5	3.45	5.65
	3	43.57	32.14	5.29	7.86	5.36	8
	4	40.6	42.8	7.7	11.1	7.7	11.1
	5	49.13	52.75	9.63	13.63	9.63	13.25
	6	58.14	63	11.43	15.43	11.14	15
	7	69.33	73.5	14.33	17.83	14	17.5
	8	89.4	93.4	16.4	20.2	16.2	20.4
	9	89.8	93.6	16.8	20.4	16	20.6
	10	111.25	115.75	21	26.5	20.5	24.75
Large	2	1217.6	1223.5	122.2	126.55	122.45	126.4
	3	1745.29	1755.79	173.43	178.43	173.79	177.86
	4	2440.2	2440.4	248.7	253.4	247.7	252.3
	5	3017.63	3049.75	317.88	322.38	316.13	318.88
	6	3490.57	3485.14	366.14	373.14	364.29	370.29
	7	4037.17	4051.83	428	435.83	426.5	432.17
	8	4848.6	4849.8	507.8	513.4	508.8	513.8
	9	4851.2	4893.4	513.8	519.4	507	512.8
	10	6053.75	6065.75	627.5	643.5	633.5	644.75
V.Large	2	24832.7	24878.7	1254.85	1262.5	1256.75	1257
	3	35539.36	35602.5	1802.29	1806.36	1807.64	1807.93
	4	49741	49826.9	2545.7	2549	2545.8	2545.9
	5	62331.63	62449	3186.13	3190.5	3197.25	3196.25
	6	71238.71	71364	3659.57	3663.71	3674	3667
	7	83158	83290.33	4277.83	4288	4288.17	4283.5
	8	99798.4	99960.4	5133.8	5139.6	5248.8	5149.8
	9	99946	100151.2	5197.2	5175.4	5241.4	5186.2
	10	125038.8	125278.3	6472	6484	6528	6503.75

Table 4.8: The aggregation time in (milliseconds) for Binary-tree, Two-bit, and One-bit techniques for computed SOAP message clusters using Vector Space clustering model

Message Size	Number of Messages per Cluster	Binary-tree Fixed Length	Binary-tree Huffman	Two-bit Fixed Length	Two-bit Huffman	One-bit Fixed Length	One-bit Huffman
Small	2	9.25	9.5	0.3	0.65	0.75	0.7
	3	9.71	10.5	0.29	0.79	0.5	1.07
	4	12.5	13.5	0.6	1.7	0.5	1.7
	5	17	18.25	0.75	2	0.88	1.88
	6	22.71	24.14	0.71	2.57	0.71	2.57
	7	22.5	24.33	1.33	3.17	1.33	3.17
	8	30.4	33	1.6	4.4	2	4
	9	33.6	36	2.2	3.8	1.8	4.2
	10	30.25	32.75	2.5	5.25	2.5	5
Medium	2	22.1	24.55	3.55	5.75	3.55	6
	3	29.43	32.86	5.21	8.36	5.36	8.5
	4	42.3	46.3	8	11.8	8	12.3
	5	48.63	53	9.75	13.63	9.63	13.63
	6	62.43	67.43	12.14	16.29	12.29	15.89
	7	71	75.33	14.17	18.67	14.17	18
	8	89.2	93.4	17	21.2	17.2	21.2
	9	92.4	97.2	17.2	21.6	17.2	22.2
	10	89.5	95	20.75	25.25	21	25.25
Large	2	1208.5	1219	121.7	126	122.7	126.6
	3	1643.36	1653.36	168.93	173.71	169.36	173.93
	4	2610.7	2631.9	268.4	272.1	268.6	272.5
	5	2936.63	2967.88	310.63	317.5	309.5	316.13
	6	3624.86	3644	378.57	384	377.43	383.43
	7	3841.33	3870.33	413.33	410.83	406.33	413.17
	8	4975	5030.4	533	528.8	528.6	527.4
	9	5028.2	5061	528.4	536.6	529.6	533
	10	5117.75	5152.25	560.25	573.75	566.25	575.5
V.Large	2	24833.45	24831.1	1260.4	1260.8	1261.3	1259.9
	3	36132.79	36154.79	1839.64	1847.21	1834.79	1845.93
	4	47977.8	48014.1	2477.2	2487.8	2481.8	2475
	5	65797	65929.88	3351.5	3360.63	3358.5	3353.38
	6	74040.14	74067.71	3791.86	3798.86	3794.86	3791
	7	84062.67	84113	4296.5	4306.5	4304.83	4307.67
	8	103559	103653.6	5322.2	5357.2	5325.8	5324.8
	9	99212.8	99157	5111.2	5126	5134.8	5182
	10	104450.5	104586.3	6589	6589.25	6598.25	6616.25

Table 4.9: The De-aggregation time in (milliseconds) for Binary-tree, Two-bit, and One-bit techniques for the aggregated SOAP message based on Jaccard clustering resultant groups

	Number of Messages	**Jaccard based Clustering Model**		
Message Size	**per Cluster**	**Binary-tree**	**Two-bit**	**One-bit**
Small	2	0.05	0.05	0.05
	3	0.05	0.05	0.05
	4	0.2	0.1	0.05
	5	0.125	0.05	0.05
	6	0.857	0.143	0.05
	7	1.333	0.333	0.05
	8	2	0.6	0.4
	9	3.4	1.2	1.4
	10	6	1.75	1.75
Medium	2	0.05	0.05	0.05
	3	0.05	0.05	0.05
	4	0.2	0.05	0.05
	5	1	0.05	0.125
	6	1.714	0.143	0.429
	7	2.333	0.833	0.833
	8	4	1.2	1.8
	9	4.6	1.8	2.2
	10	8.5	3	3.5
Large	2	0.05	0.05	0.05
	3	0.571	0.05	0.05
	4	1.5	0.5	0.4
	5	2.625	1	1.375
	6	4	1.857	1.571
	7	5.833	2.5	2.667
	8	8.8	4	4
	9	10.6	4.4	4.8
	10	16	7.25	7.25
V.Large	2	0.1	0.05	0.05
	3	1.714	0.643	0.786
	4	4.3	1.8	1.9
	5	7.5	3.5	3.5
	6	10.426	5	5.143
	7	16.167	7.167	7.333
	8	22.4	10.4	11.4
	9	26.4	12	13
	10	39.5	18.5	18.75

Table 4.10: The De-aggregation time in (milliseconds) for Binary-tree, Two-bit, and One-bit techniques for the aggregated SOAP message based on Vector Space Clustering resultant groups

	Number of Messages	**Vector Space Model based Clustering**		
Message Size	**per Cluster**	**Binary-tree**	**Two-bit**	**One-bit**
Small	2	0.05	0.05	0.05
	3	0.05	0.05	0.05
	4	0.1	0.05	0.1
	5	0.625	0.05	0.05
	6	1	0.286	0.286
	7	1.333	0.05	0.5
	8	2.8	1	0.8
	9	3.4	1.4	1.2
	10	5.5	2	1.75
Medium	2	0.05	0.05	0.05
	3	0.143	0.05	0.05
	4	0.6	0.05	0.05
	5	1.125	0.125	0.125
	6	1.857	0.429	0.143
	7	2.667	0.833	0.833
	8	4.2	1.4	1.6
	9	5.2	2	2.2
	10	8	3	3.25
Large	2	0.05	0.05	0.05
	3	0.214	0.071	0.071
	4	1.8	0.8	0.4
	5	2.875	1.375	1.5
	6	4.286	1.571	1.857
	7	6.333	2.833	2.333
	8	10	4.2	4.2
	9	10.6	4.8	5.4
	10	14	6.75	7
V.Large	2	0.15	0.05	0.05
	3	2	0.929	0.857
	4	4.5	2.2	2.2
	5	8.125	4	4.125
	6	11.143	5.429	5.571
	7	16.333	7.667	7.667
	8	24.2	12.2	12.8
	9	28	12.6	12.8
	10	36.25	19.75	19.75

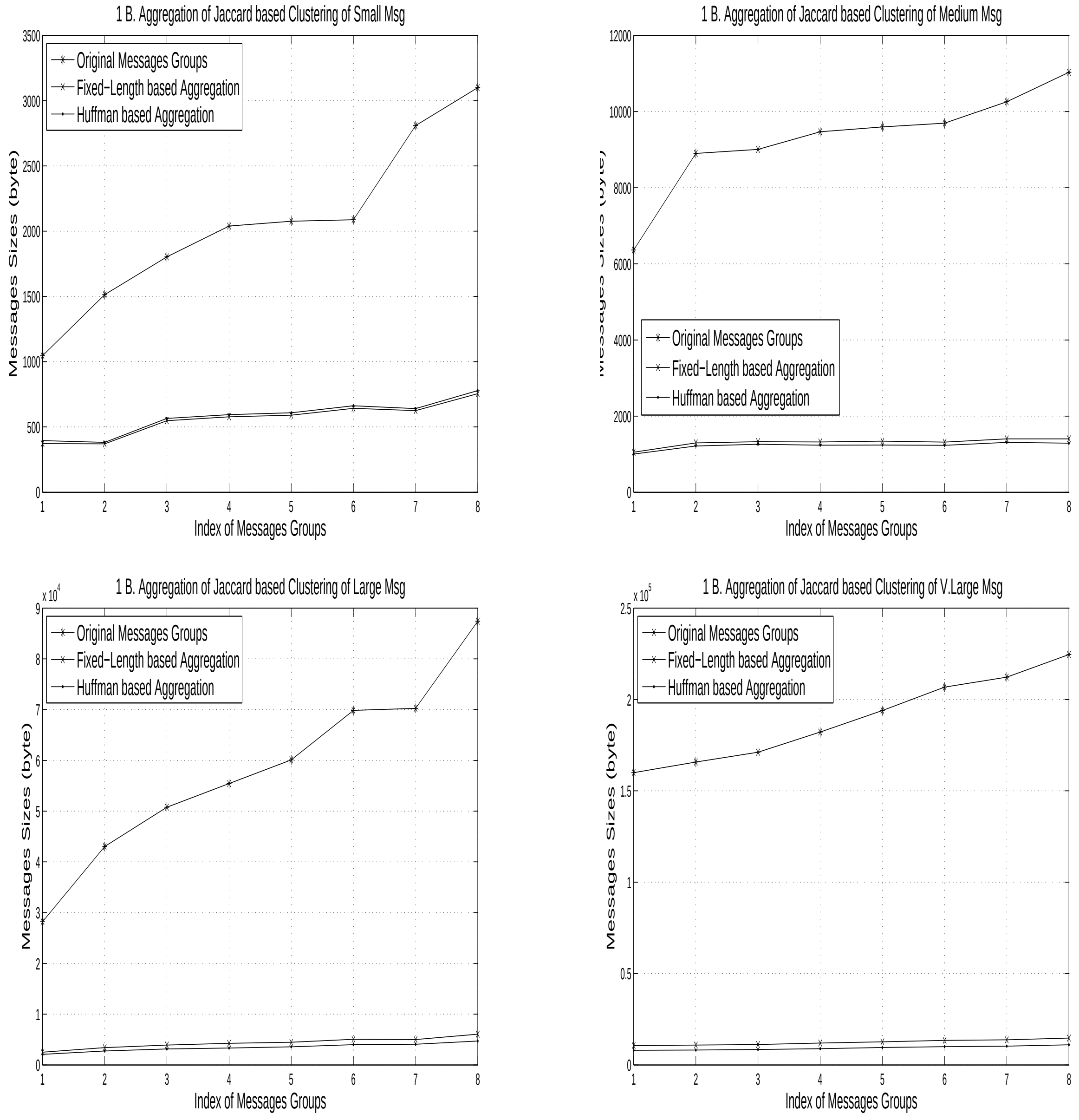


Figure 4.14: One-bit resultant aggregated compact size of small, medium, large, and very large SOAP messages using Jaccard similarity grouping with 5 messages per group as the group size

messages are aggregated with 9 different group sizes starting from 2 messages per cluster and up to 10 messages per cluster. The results show that as cluster size increases, a higher

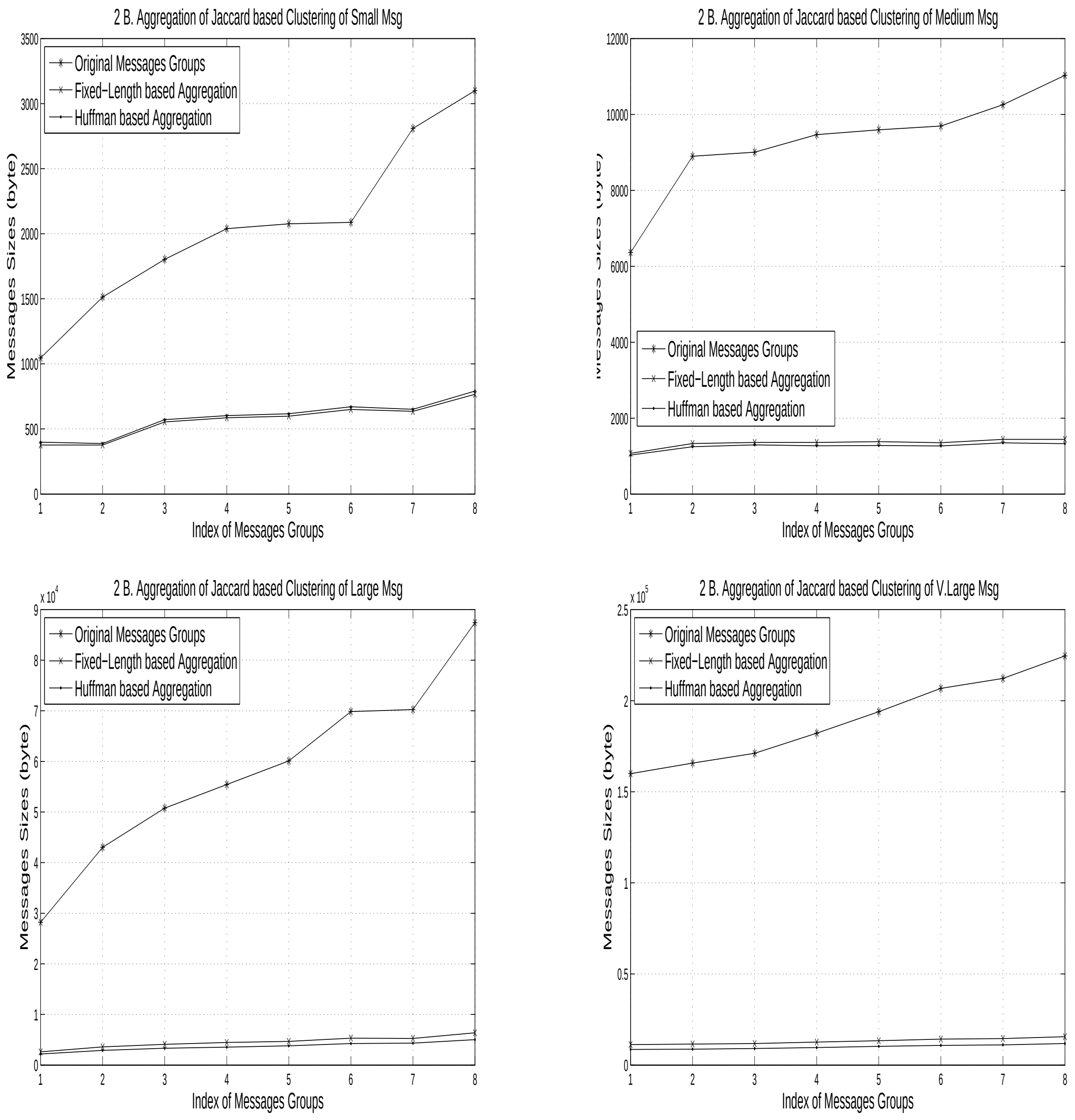


Figure 4.15: Two-bit resultant aggregated compact size for small, medium, large, and very large SOAP messages using Jaccard similarity grouping with 5 messages per group as the group size

compression ratio can be achieved. The compression ratios of the aggregated groups based on the Vector Space Model clustering are slightly higher than the aggregated groups based

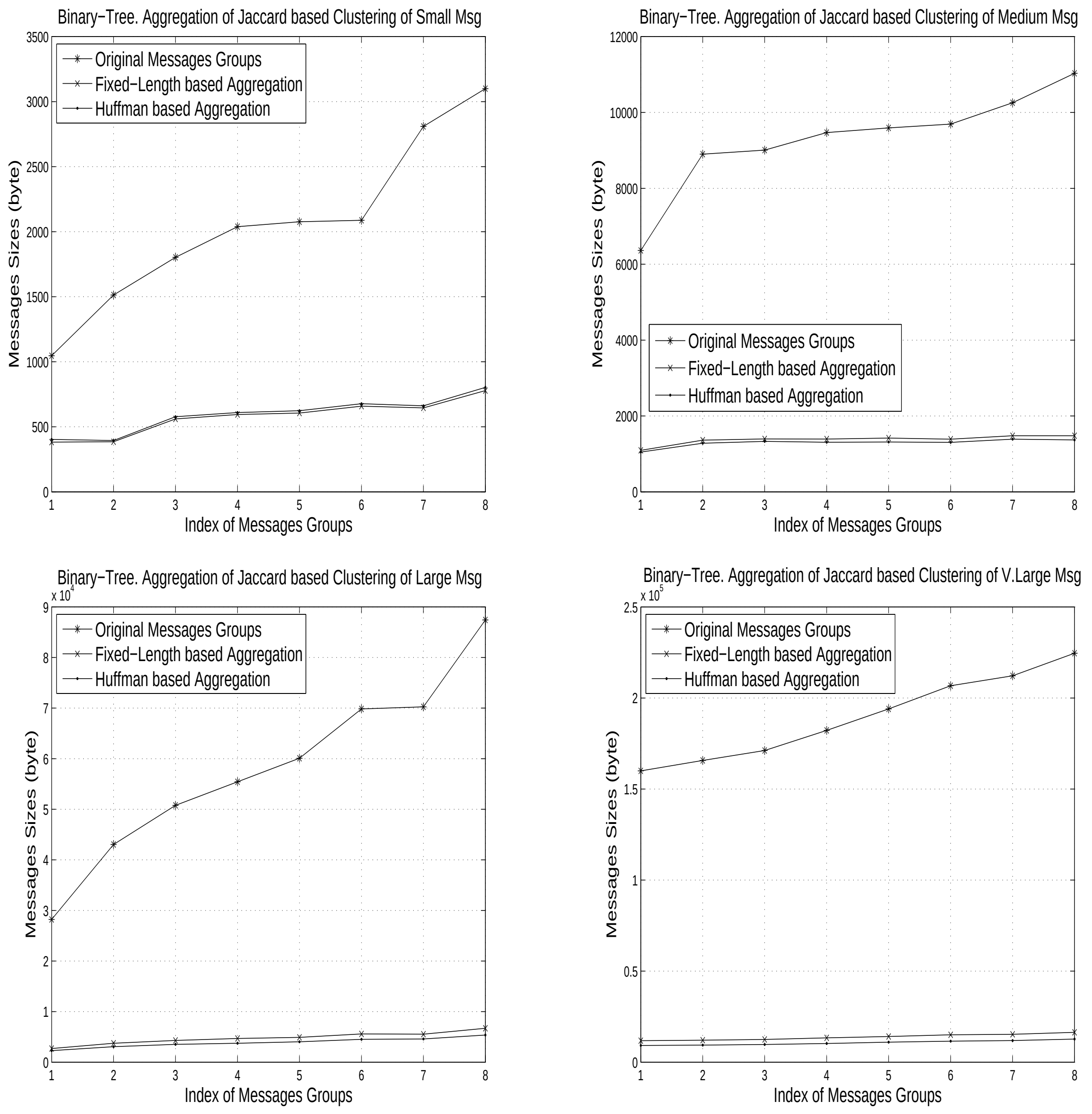


Figure 4.16: Binary-tree resultant aggregated compact size for small, medium, large, and very large SOAP messages using Jaccard similarity grouping with 5 messages per group as the group size

on Jaccard clustering.

The aggregation and de-aggregation time for Two-bit and One-bit status tree techniques

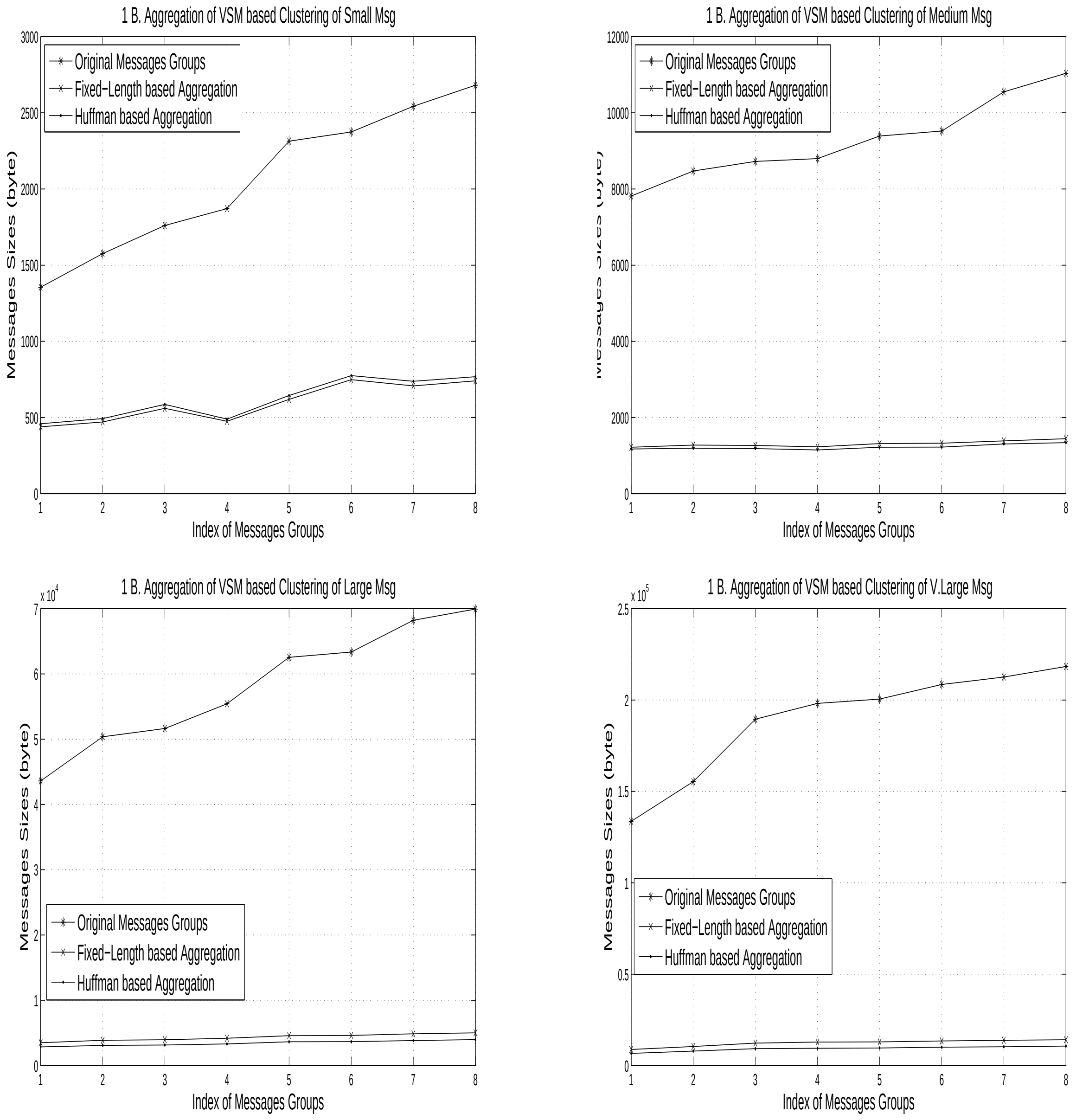


Figure 4.17: One-bit resultant aggregated compact size for small, medium, large, and very large SOAP messages using Vector Space Model similarity grouping with 5 messages per group as the group size

in addition to the Binary-tree technique are investigated for five SOAP messages, one for each group size: small, medium, large, and very large. Figures 4.22 and 4.23 show the aggre-

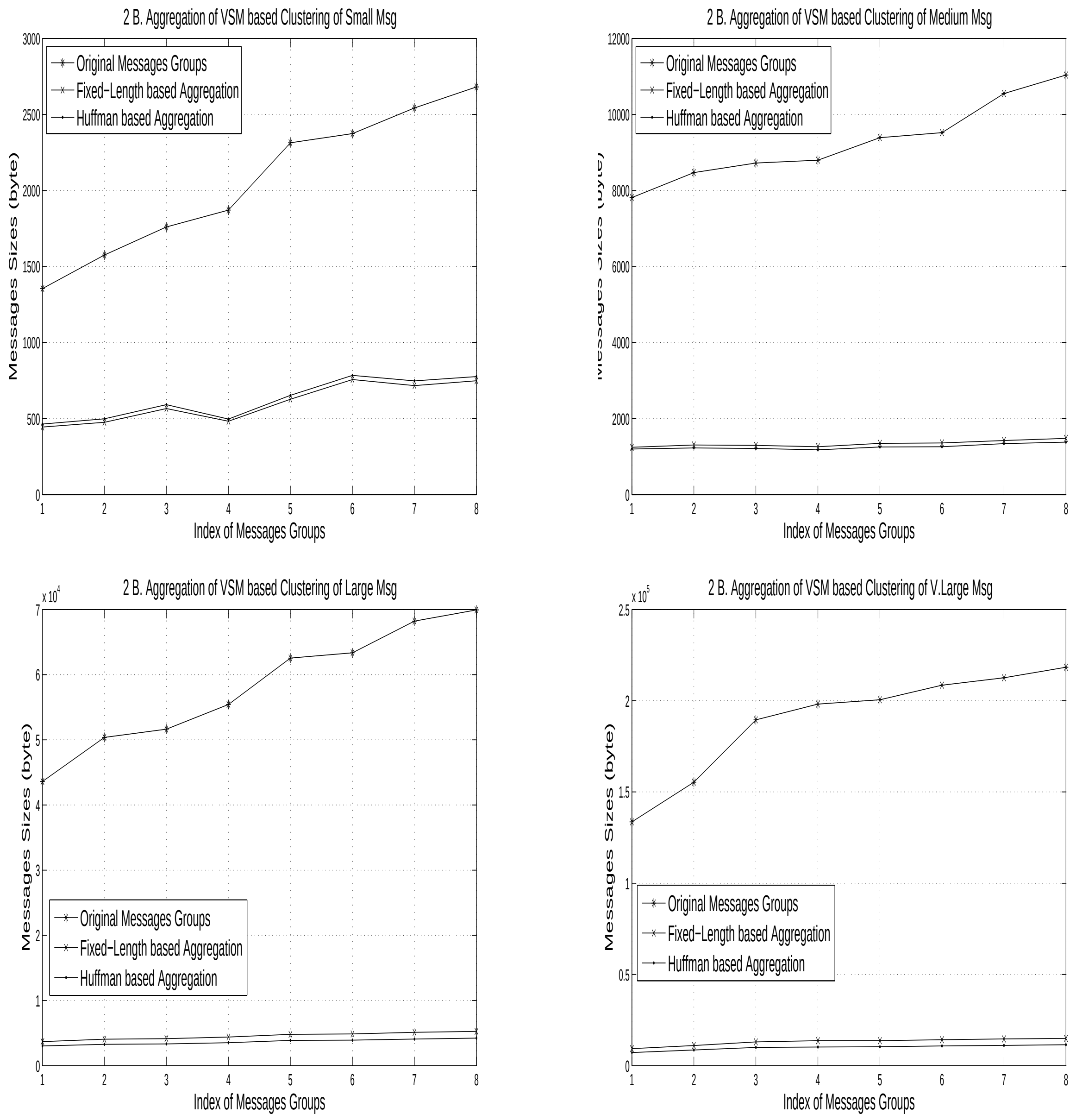


Figure 4.18: Two-bit resultant aggregated compact size for small, medium, large, and very large SOAP messages using Vector Space Model similarity grouping with 5 messages per group as the group size

gation time for all proposed aggregation models with both Jaccard and Vector Space Model based clustering respectively. Furthermore, Figures 4.24 and 4.25 show the de-aggregation

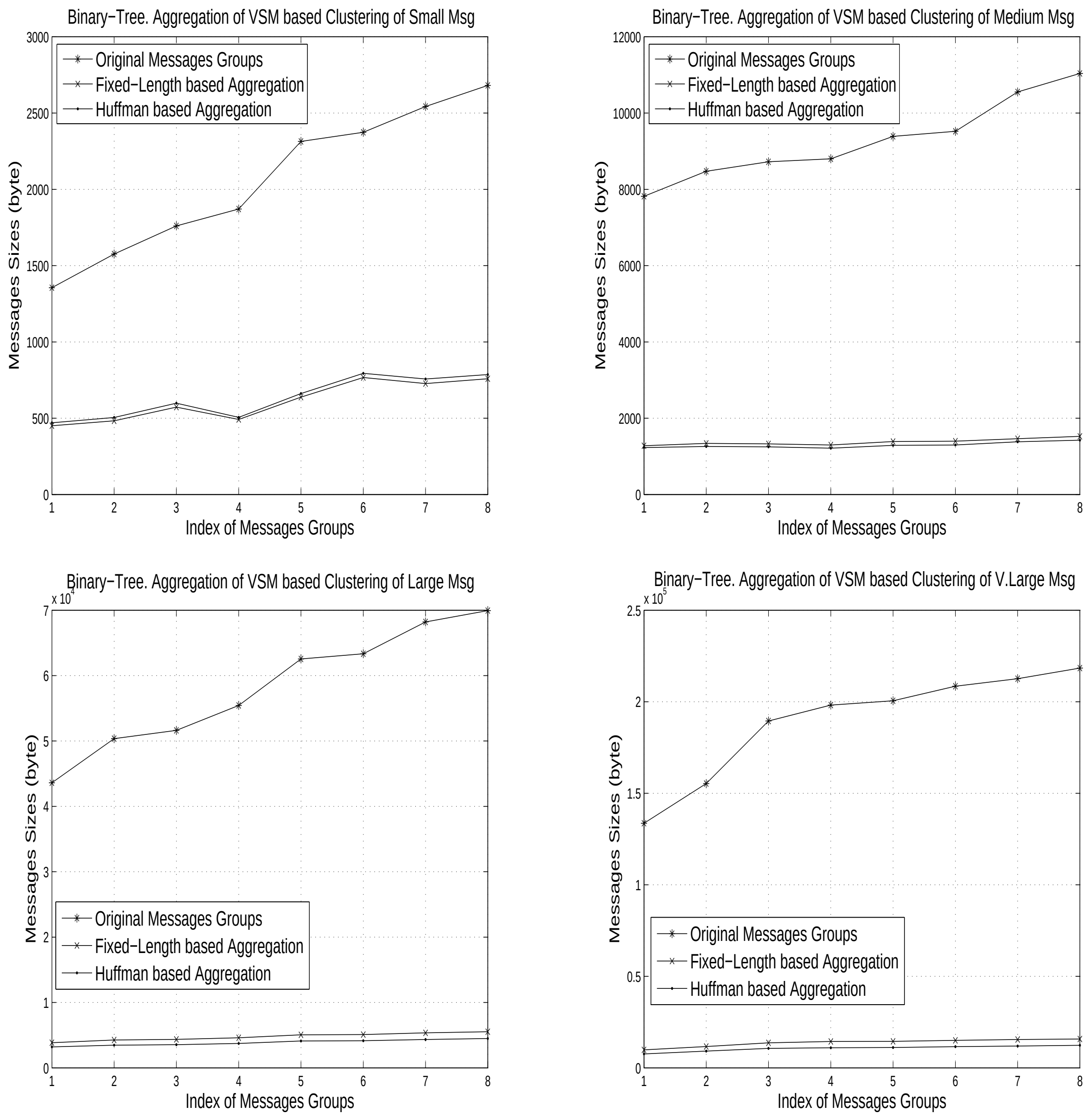
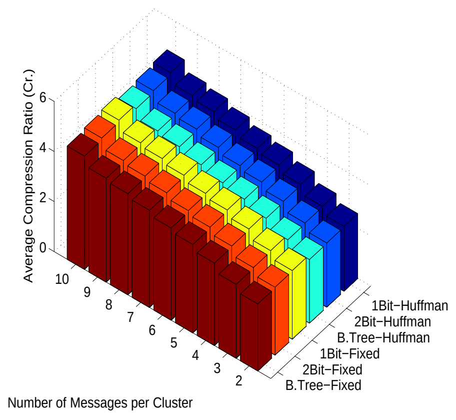
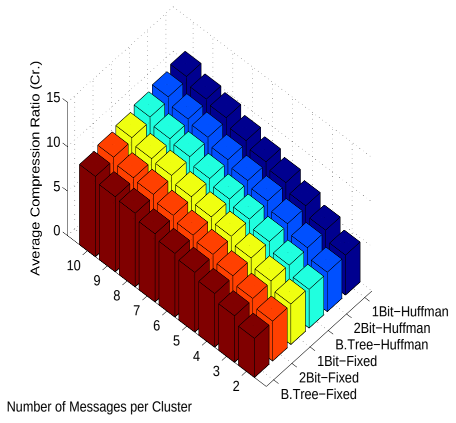


Figure 4.19: Binary-tree resultant aggregated compact size for small, medium, large, and very large SOAP messages using Vector Space Model similarity grouping with 5 messages per group as the group size

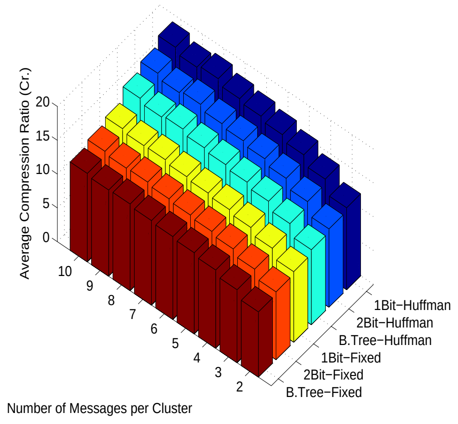
time for all proposed aggregation models with both Jaccard and Vector Space Model based clustering respectively. The One-bit and Two-bit aggregation techniques have shown sig-



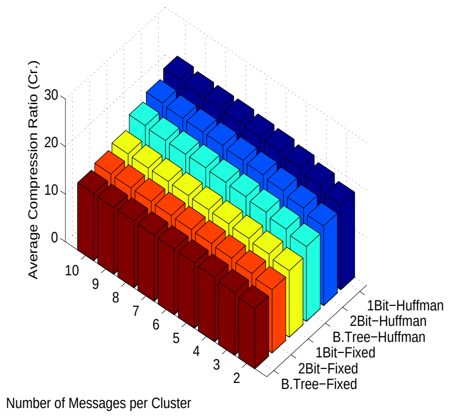
a. Small sized messages



b. Medium sized messages



c. Large sized messages

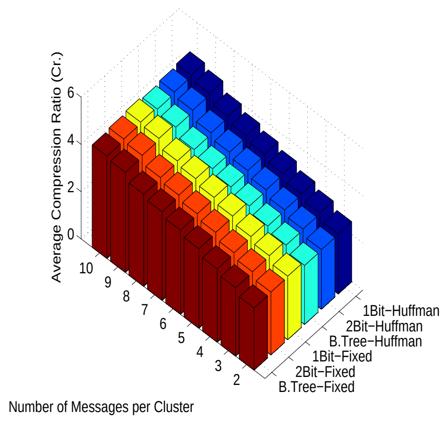


d. V.Large sized messages

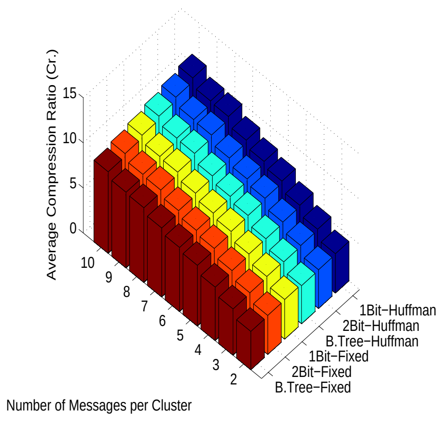
Figure 4.20: Resultant average compression ratios for small, medium, large, and very large groups of aggregated SOAP messages using Jaccard similarity grouping

nificantly better processing time than the Binary-tree technique. The same applies to the de-aggregation time as both One-bit and Two-bit techniques outperformed the Binary-tree technique by consuming potentially less processing time for de-aggregating SOAP messages.

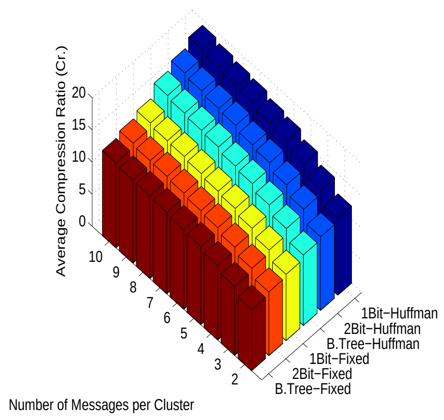
In order to investigate the required processing time for both the aggregation and de-



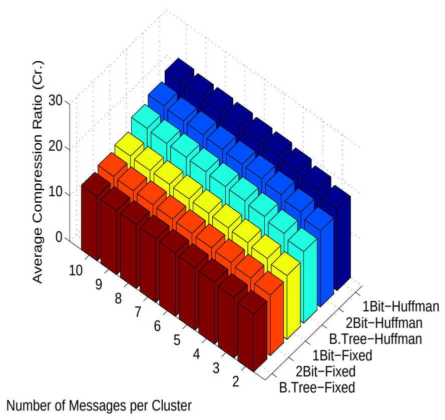
a. Small sized messages



b. Medium sized messages



c. Large sized messages



d. V.Large sized messages

Figure 4.21: Resultant average compression ratios for small, medium, large, and very large groups of aggregated SOAP messages using Vector Space Model similarity grouping

aggregation techniques in more detail, both approaches are investigated using different cluster sizes of SOAP messages that range from only 2 and up to 10 messages in each cluster. Tables 4.7 and 4.8 show the processing time for aggregating small, medium, large, and very large sized messages using Binary-tree, Two-bit, and One-bit aggregation models based on both Jaccard and Vector Space clustering models. Moreover, Tables 4.9 and 4.10 show the

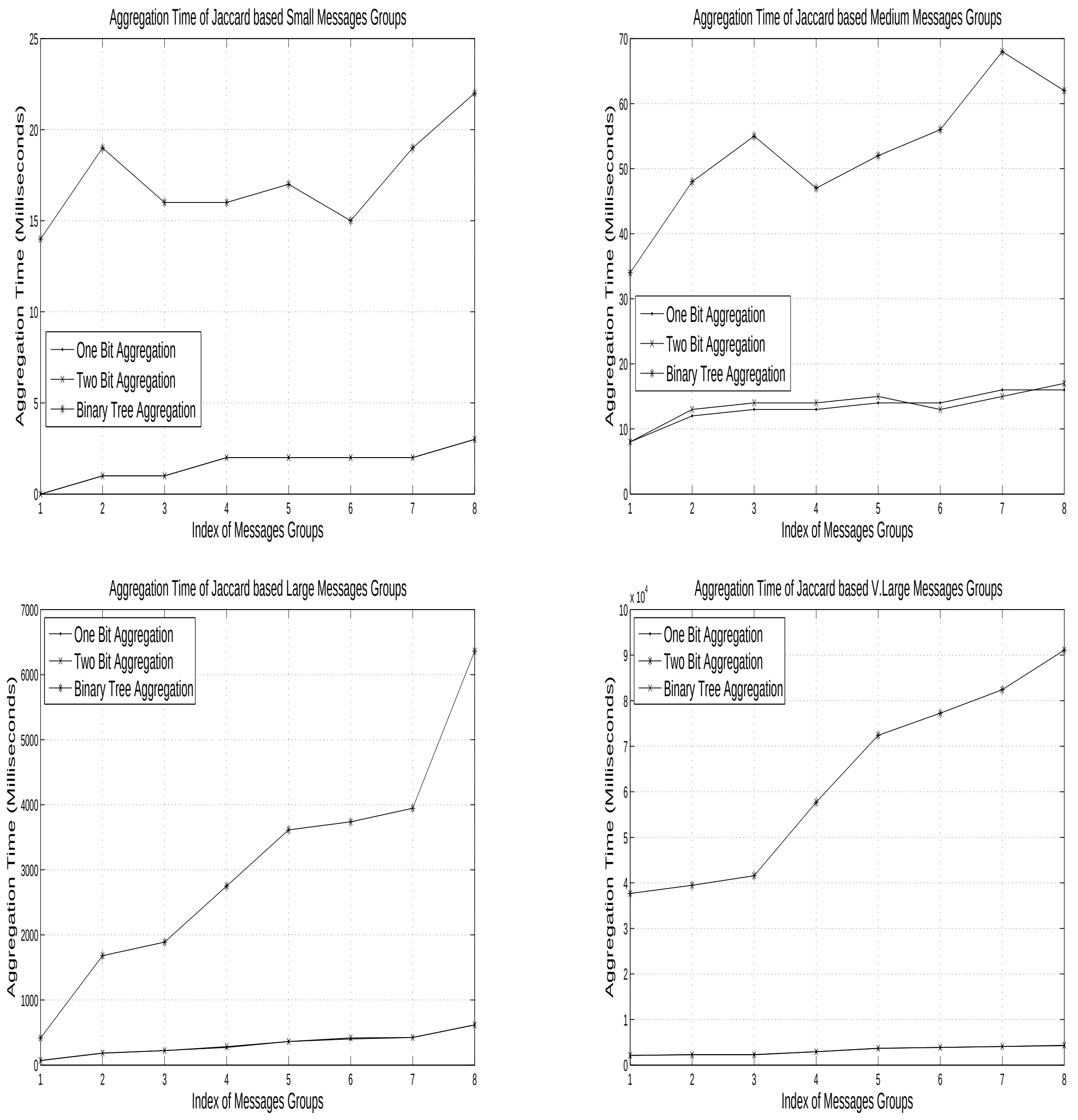


Figure 4.22: One-bit, Two-bit, and Binary-tree aggregation time for Jaccard based clustered small, medium, large, and very large SOAP messages with 5 messages per group as the group size

de-aggregation time for the aggregated clusters of SOAP messages that are grouped using both Jaccard and Vector Space models. The results show that both Two-bit and One-bit

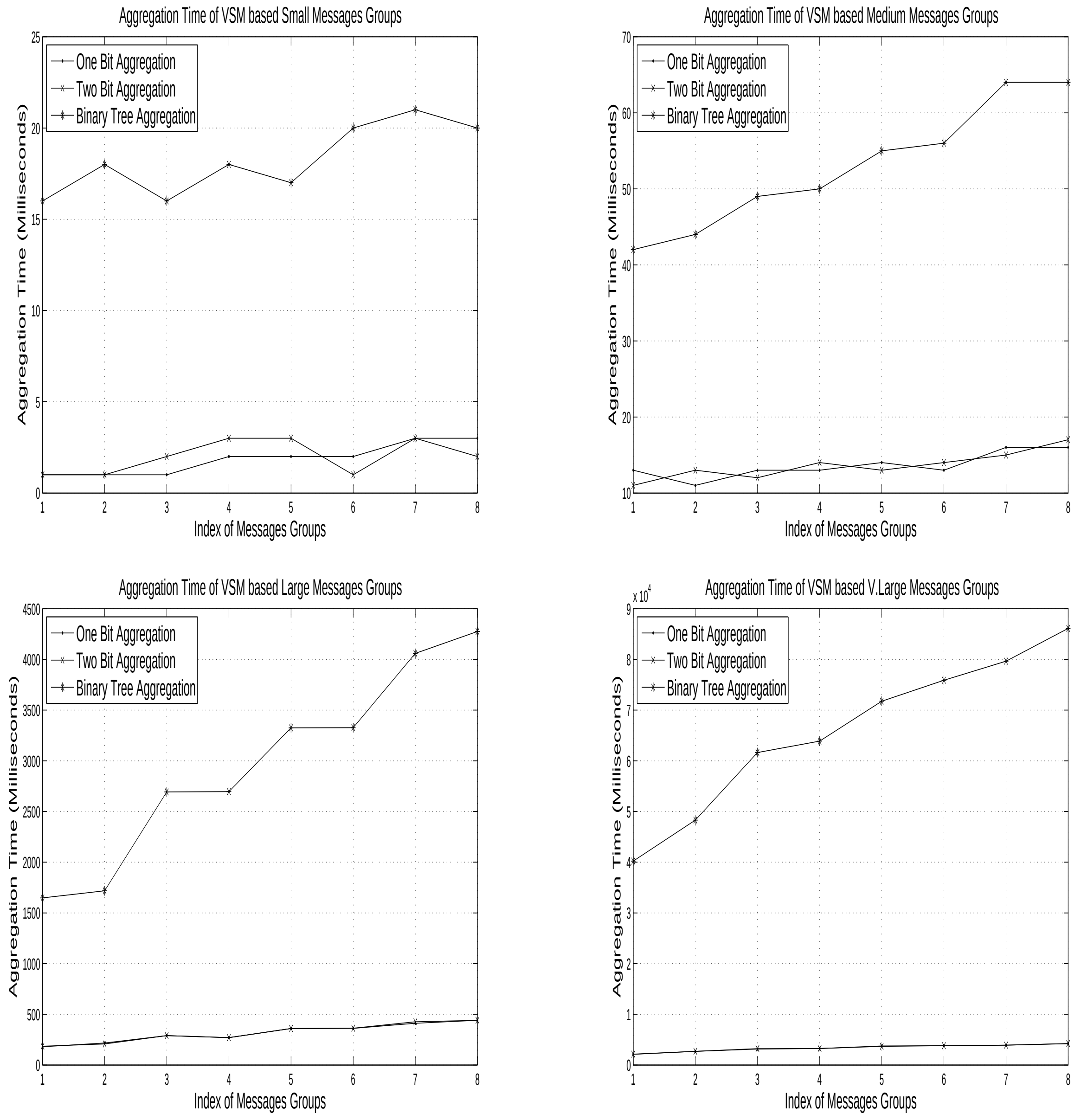


Figure 4.23: One-bit, Two-bit, and Binary-tree aggregation time for Vector Space Model based clustered small, medium, large, and very large SOAP messages with 5 messages per group as the group size

techniques have achieved tremendously better performance in terms of the processing time for both aggregation and de-aggregation versions. Furthermore, the aggregation and de-

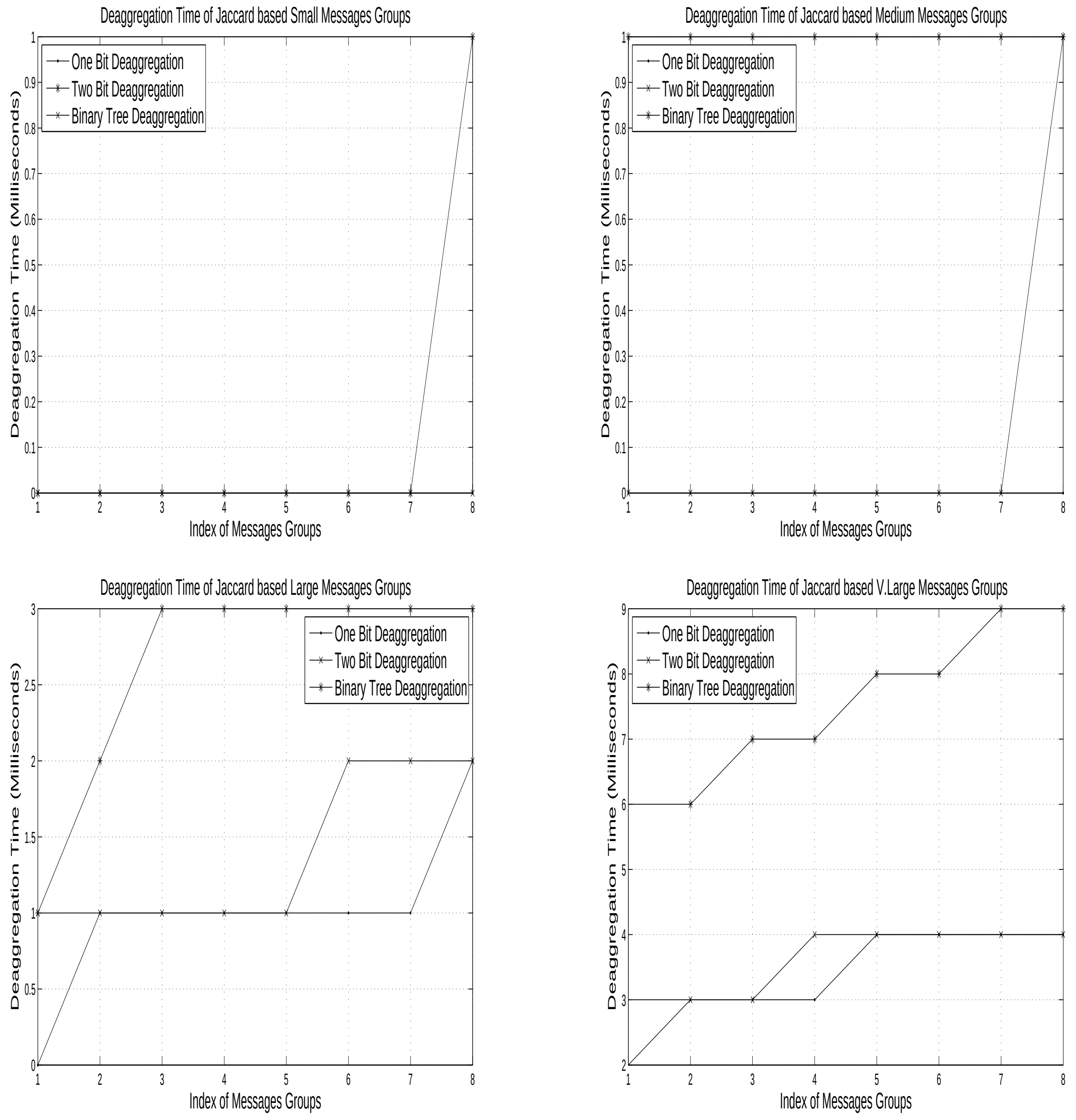


Figure 4.24: One-bit, Two-bit, and Binary-tree de-aggregation time for Jaccard based clustered small, medium, large, and very large SOAP messages with 5 messages per group as the group size

aggregation time for Two-bit and One-bit techniques are very close to each other and the processing time increases as the size of SOAP messages (small, medium, large, and V.large)

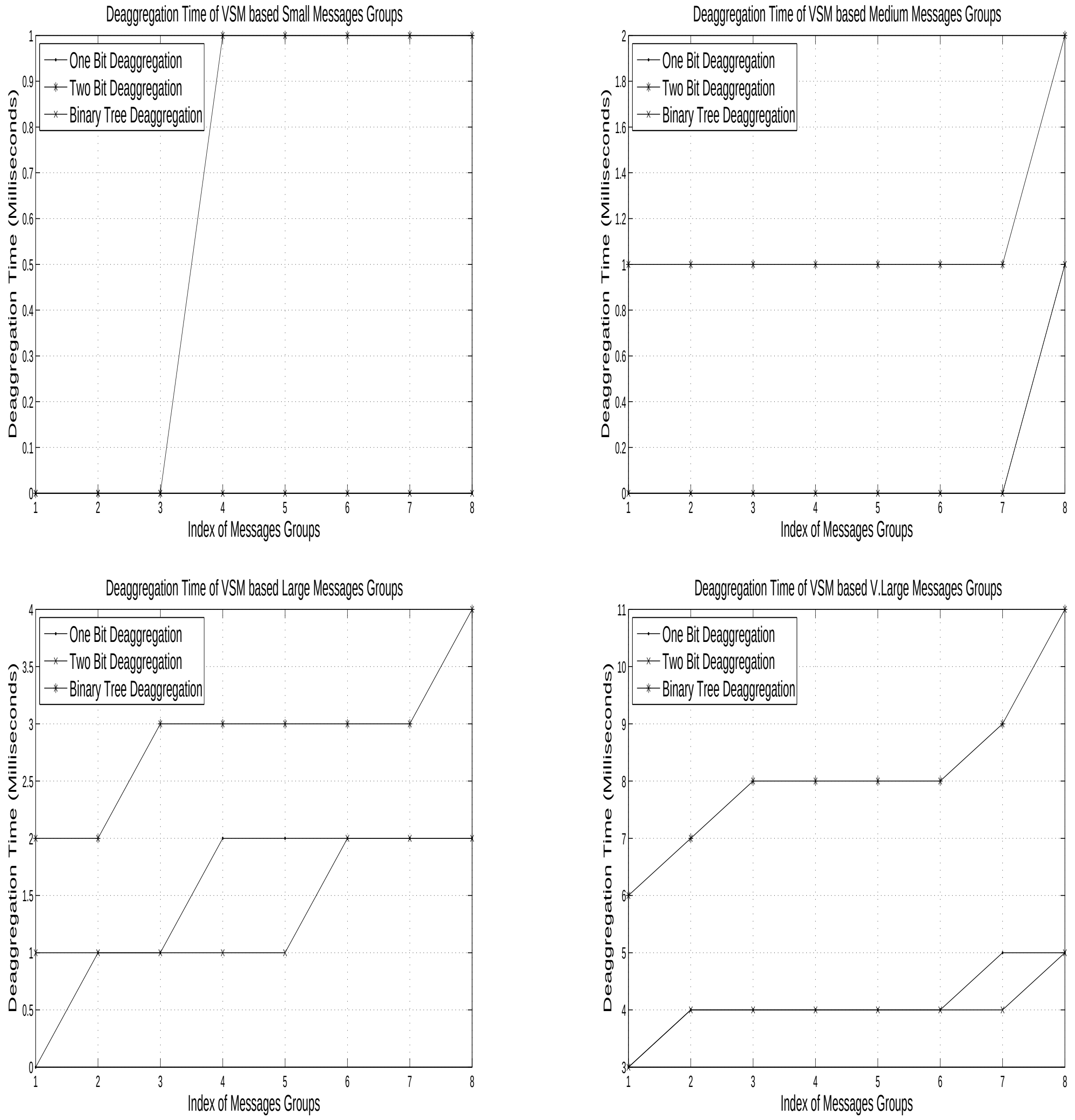


Figure 4.25: One-bit, Two-bit, and Binary-tree de-aggregation time of Vector Space Model based clustered small, medium, large, and very large SOAP messages with 5 messages per group as the group size

increase.

4.5 Conclusion

XML-aware compression techniques can be developed into efficient SOAP aggregation models that can exploit redundancies in several SOAP messages. In this chapter, we have shown that redundancy-based aggregation techniques can outperform all standalone compression techniques by achieving higher compression ratios for small, medium, large and very large sized messages. The performance of the Web services can be improved by applying the redundancy-aware aggregation models enabling Cloud Web servers to generate a compact message that can be used by receivers and extract the original messages. A new compressibility measurement technique in our work shows that we can predict the ability of aggregation models when we group similar messages appropriately. The One-bit XML status tree aggregation technique outperformed all other standard compression techniques in addition to the Two-bit XML status tree and Binary-tree based aggregation models. Both Jaccard and Vector Space based clustering models have shown significant performance, enabling aggregation models to achieve high compression ratios ≥ 20. However, Vector Space outperformed Jaccard clustering in terms of supporting the aggregation models to reduce the size of SOAP messages efficiently and the required processing time to cluster messages.

Chapter 5

Fractal Self-Similarity for Dynamic SOAP Clustering Model

This chapter answers the third research question posted in section 1.2 by modulating and developing the self-similarity principle of fractal mathematical model to compute the similarity of SOAP messages. Fractal is proposed as an unsupervised clustering technique that is dynamically grouping SOAP messages.

The chapter is organized as follows. First, subsections 5.1.1, 5.1.2 and 5.1.3 show the motivation, main contributions, and evaluation strategy respectively. Section 5.2 explains the process of computing XML documents dataset and how to represent XML messages as numeric vectors in the generated dataset. Section 5.3 discusses the fractal mathematical model and how it can be utilized in clustering SOAP messages. Next, section 5.4 explains the computations of fractal coefficients of SOAP messages in a separate algorithm and then the fractal root mean square error criteria. Section 5.5 describes the experimental evaluation of the proposed clustering technique. Finally, the conclusion is presented in section 5.6.

5.1 Introduction

Aggregation of SOAP messages is an effective solution that could potentially reduce the required bandwidth. Although SOAP aggregation has resulted in significant improvements, it still needs to be supported with advance similarity measurements for SOAP messages with the aim to cluster SOAP messages with high similarity degree as group-wise and not only pair-wise.

5.1.1 Motivation

Aggregation of SOAP messages is one of the modern and effective models to reduce network traffic. The aggregation schemes can be used in a number of network applications and scenarios to be enabled to minimize the required bandwidth over the Internet like multicasting of aggregated SOAP messages to the Web clients and split them at the closest routers (see Fig. 5.1). In chapter 4, we have introduced a new SOAP messages aggregation strategy by utilizing compression concepts. The proposed aggregation models are basically strengthened by the redundancy awareness features of compression as alternative similarity measurements to aggregate messages in one compact structure. Technically, these aggregation models consist of two main activities: transforming the XML tree of SOAP messages into minimized SOAP textual expression and then encoding them with either fixed-length or Huffman encoding techniques. Although these aggregation techniques can aggregate as many messages as requested by the Web server, advance cluster-based similarity measurements are still required to find out which group of SOAP messages is optimum to be aggregated as alternative to the traditional pair-based SOAP messages similarity measurements.

Generally, standard clustering techniques such as K-Means and Vector Space Model [Liu et al., 2004; Yongming et al., 2008] could be alternatives to the SOAP similarity measurements. However, they do not represent an efficient similarity measurement because of the following drawbacks:

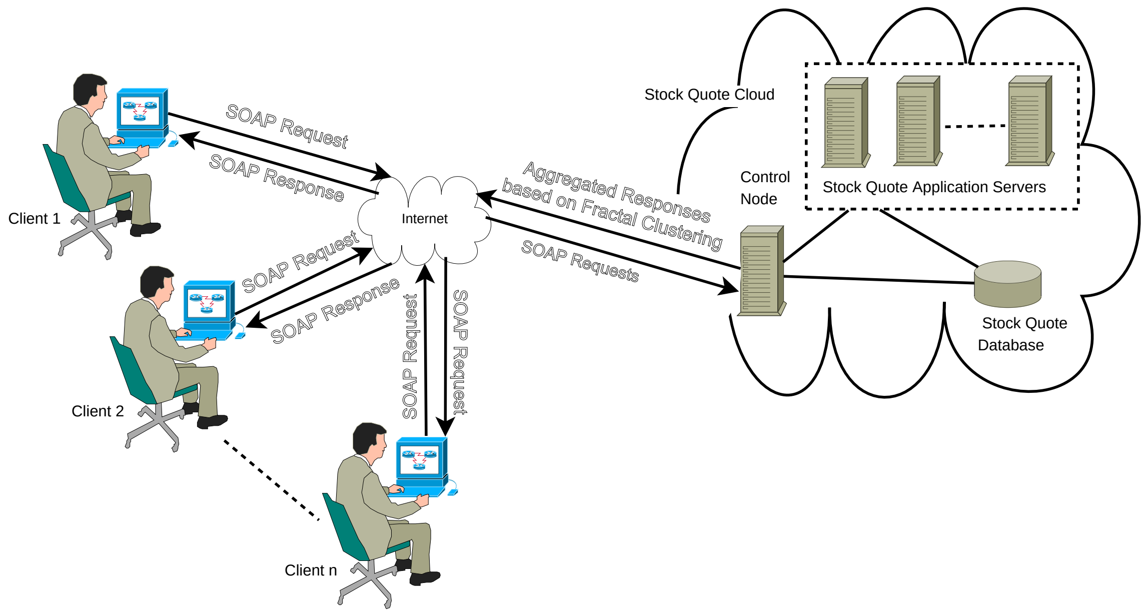


Figure 5.1: Clustering and aggregation support for stock quote cloud Web services over the Internet

- *High complexity*: They have high complexity that could result in long clustering time. This is caused by their iterative computations for finalizing clusters.
- *Inefficient prediction of clusters*: A fixed number of clusters usually results in inefficient prediction of clusters. Messages are likely to be clustered without having high similarity degree with other messages in the same cluster. Moreover, fixed number of cluster based models do not work efficiently with non-globular clusters such as SOAP clusters with high redundancy.
- *Inaccurate and inefficient centroids selection*:Most clustering models start with initial partitions that might be selected randomly usually result in inaccurate and inefficient clustering of messages.

Unsupervised and dynamic clustering models are effective techniques to solve the problem of SOAP clusters with high common redundancy.

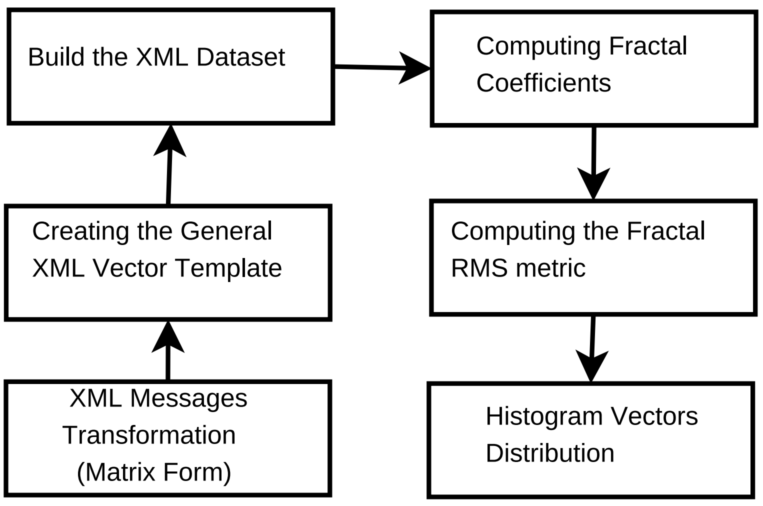

Build the XML Dataset
Computing Fractal Coefficients
Creating the General XML Vector Template
Computing the Fractal RMS metric
XML Messages Transformation (Matrix Form)
Histogram Vectors Distribution

Figure 5.2: Main model components

5.1.2 Contribution

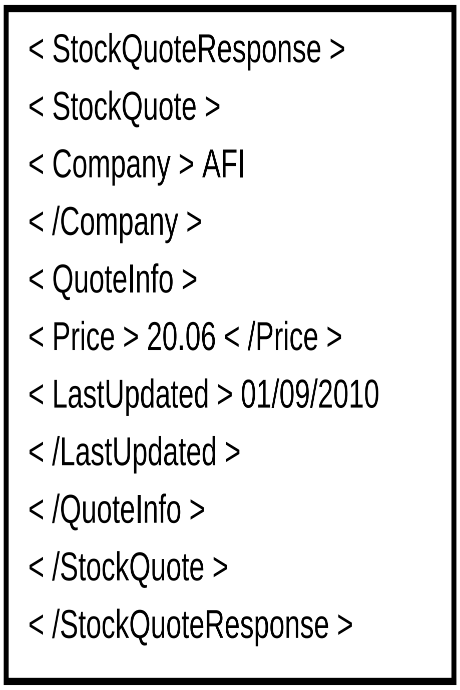
Fractal, as a mathematical model, provides powerful self-similarity measurements for the fragments of regular and irregular geometric objects in their numeric representations [t, 1994]. Partitioning Iterated Function System (PIFS) represents the power of fractal in depicting the similarity of smaller parts in the same numeric object [Baharav, 1999]. PIFS explains the dynamics of creating fractals by uniting several copies of the same object with different scales which every copy is made up of smaller scaled copies of itself. In comparison with other traditional XML similarity measurements, fractal can provide similarity measurement to complete parts (set of features) of the objects at once and not only investigating XML features separately.

With the aim to provide efficient clustering predictions, this chapter investigates fractal fragments in SOAP messages as their XML tree could be segmented into several fractal objects. Figure 5.6 shows SOAP fractal segments in comparison with Mandelbrot fractal set. Figure 5.2 states the main components of the proposed clustering technique. The main contributions made in this chapter are:

- *Efficient prediction:* Fractal mathematical parameters are introduced to compute SOAP message similarities that are applied based on the numeric representation of SOAP messages. The proposed technique aims to create clusters with a very high degree of similarity by dynamically grouping them together where the proposed technique does not require a predefined number of clusters. Experimental results show significant predictions for similar SOAP messages by the proposed technique in comparison with K-Means and PCA combined with K-Means. The accurate predictions of the proposed technique is capable of achieving better compression ratios than other clustering models [Liu et al., 2004; Yongming et al., 2008].

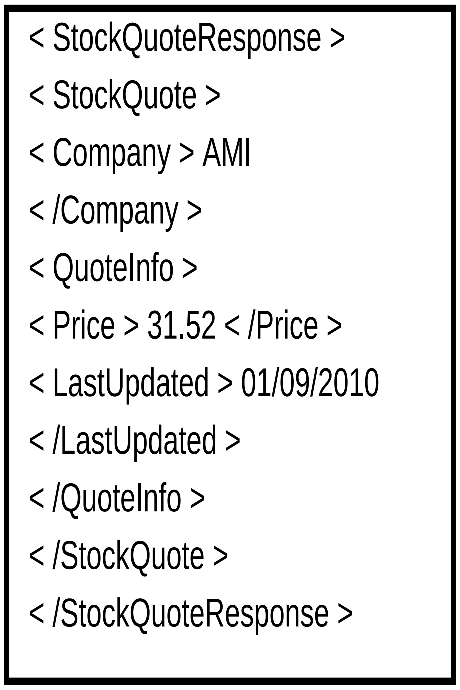
- *Low complexity clustering:* SOAP fractal similarities are developed to devise a new unsupervised auto clustering technique. These similarity measurements are based on computing fractal coefficients of numeric fragments that construct a single numeric object [Baharav, 1999]. The proposed technique provides a low complexity clustering in comparison with iterative models. The clustering time required by the proposed technique is potentially less when compared with other iterative clustering models [Liu et al., 2004; Yongming et al., 2008]. Fractal clustering requires only 20% and 16% of the required time by K-Means and PCA combined with K-Means respectively.

- *Efficient dataset for accurate clustering:* The proposed dataset of SOAP messages is a set of numeric vectors showing the local and global loads of XML items. These vectors are broken up into equally sized blocks. Fractal coefficients of the vector blocks represent the similarity parameters that are compared with blocks of other vectors to be the key metric for clustering SOAP messages. The proposed structure for the dataset has accurately reflected the features of SOAP messages and enabled the clustering technique to efficiently measure their similarities.



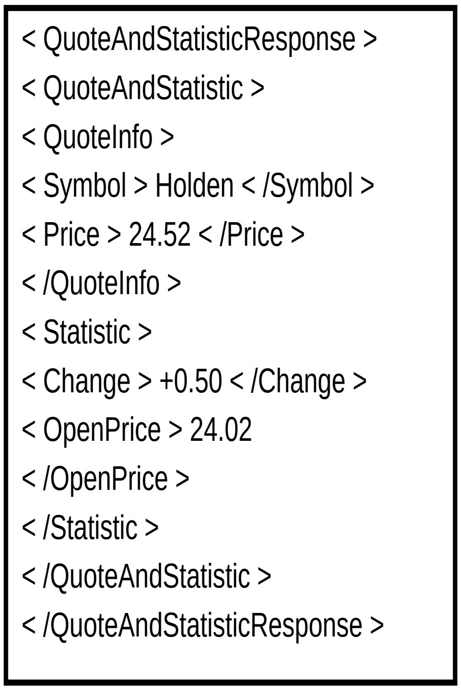
```
< StockQuoteResponse >
< StockQuote >
< Company > AFI
< /Company >
< QuoteInfo >
< Price > 20.06 < /Price >
< LastUpdated > 01/09/2010
< /LastUpdated >
< /QuoteInfo >
< /StockQuote >
< /StockQuoteResponse >
```

a. S_1



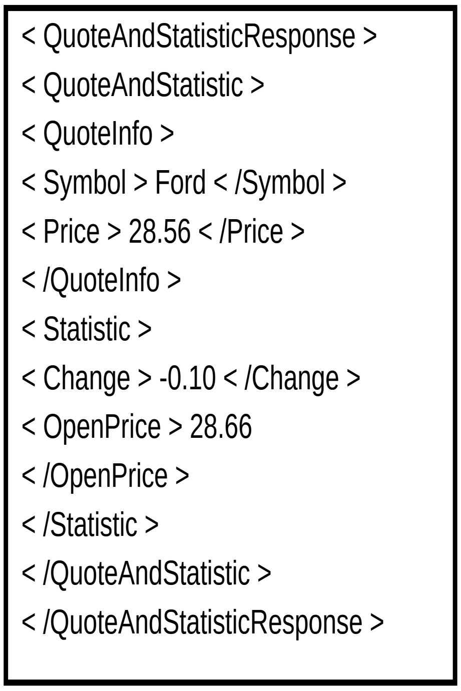
```
< StockQuoteResponse >
< StockQuote >
< Company > AMI
< /Company >
< QuoteInfo >
< Price > 31.52 < /Price >
< LastUpdated > 01/09/2010
< /LastUpdated >
< /QuoteInfo >
< /StockQuote >
< /StockQuoteResponse >
```

b. S_2



```
< QuoteAndStatisticResponse >
< QuoteAndStatistic >
< QuoteInfo >
< Symbol > Holden < /Symbol >
< Price > 24.52 < /Price >
< /QuoteInfo >
< Statistic >
< Change > +0.50 < /Change >
< OpenPrice > 24.02
< /OpenPrice >
< /Statistic >
< /QuoteAndStatistic >
< /QuoteAndStatisticResponse >
```

c. S_3



```
< QuoteAndStatisticResponse >
< QuoteAndStatistic >
< QuoteInfo >
< Symbol > Ford < /Symbol >
< Price > 28.56 < /Price >
< /QuoteInfo >
< Statistic >
< Change > -0.10 < /Change >
< OpenPrice > 28.66
< /OpenPrice >
< /Statistic >
< /QuoteAndStatistic >
< /QuoteAndStatisticResponse >
```

d. S_4

Figure 5.3: SOAP message responses to the requests getStockQuote(X) and getQuoteAnd-Statistic(Y)

5.1.3 Evaluation strategy

In order to evaluate the performance of the proposed fractal clustering technique, the compression-based aggregation models (Binary-tree, Two-bit, and One-bit aggregations) developed in chapter 4 are used to compute the achievable SOAP messages size reduction after aggregating the clustered messages by the proposed fractal technique and both the K-means and PCA combined with K-means for comparison regards. The evaluation showed that the fractal clustering technique enables the compression-based aggregation models to achieve higher messages size reduction than other techniques. Furthermore, local errors (i.e. error rates between every single message and the center message of the same cluster) and global errors (i.e. error rates between every center message and centers of other clusters) have been evaluated. Moreover, the processing time of the proposed model is investigated and compared with the processing time of other techniques and it is found to be significantly lower than other models.

5.2 Document representation

Clustering techniques are usually developed to work on a specific dataset format that represents the considered XML documents. The XML tree is the main structure of XML messages. Therefore, the first step of the proposed XML document preparation is to build the XML tree of all the XML messages. the generated XML trees for the given SOAP messages in Fig. 5.4 are shown in Fig. 5.3. Level-order traversal is used to traverse all the generated XML trees to build the matrix form of XML messages (see Fig. 5.5). Matrix form is the basic format of the transformed XML messages that is required to convert them into time series representation (as shown in Eq. 5.5).

With the aim of transforming the XML document from the textual domain to the frequency domain (time series), modification of the XML dataset starts with computing the vector template that has a unique copy of every single XML item in the XML documents.

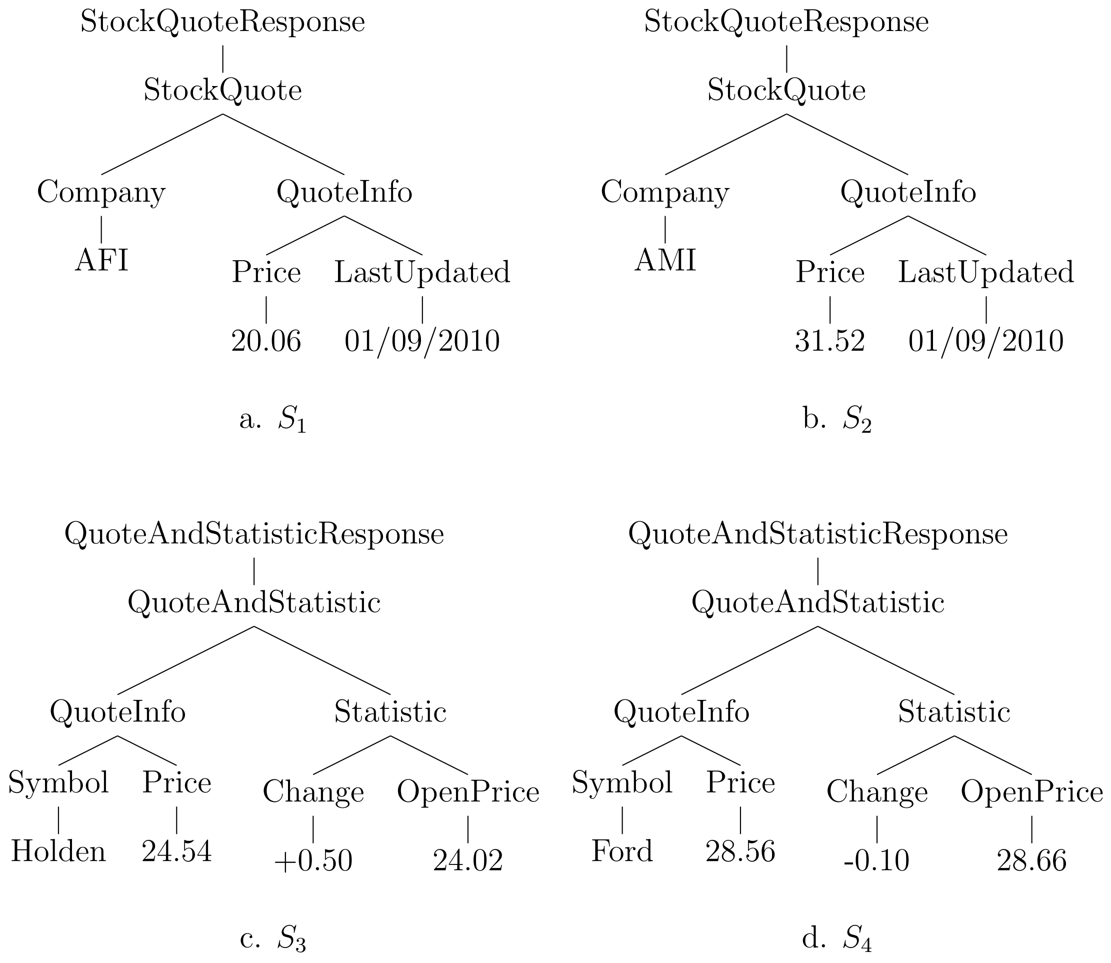


Figure 5.4: XML messages trees of SOAP messages (S_1, S_2, S_3, and S_4)

The frequencies of XML items represent the time series attributes of the XML document. The dataset vectors contain frequencies of the XML items that are ordered in the same way of their distinctive textual contents of the composed vector template. Equation 5.1 shows the general format of the vector template.

$$V_{template} = [Nd_1, Nd_2, Nd_3, ..., Nd_n] \tag{5.1}$$

where Nd_i is the node content of the ith XML item in the generated XML tree. Equation 5.2

represents the dataset system that consists of the generated time series items of the XML documents.

$$\begin{aligned} V_1 &= [X_1, X_2, X_3, ..., X_n] \\ V_2 &= [X_1, X_2, X_3, ..., X_n] \\ &. \\ &. \\ &. \\ V_m &= [X_1, X_2, X_3, ..., X_n] \end{aligned} \tag{5.2}$$

where:

- V_i is the frequency vector of the ith XML message in the dataset,
- X_i is the frequency of ith XML item of the XML message,
- n is the total number of all the distinctive XML items in the generated XML matrix forms, and
- m represents the total number of the XML messages in the dataset.

In most clustering approaches, the datasets are usually generated as a set of vectors $V = \{x_1, x_2, ..., x_n\}$, where every single element x_i refers to a single item that represents a single feature of the document. In this research, the Term Frequency with Inverse Document Frequency (TF-IDF) weights [Hwang and Gu, 2007] is used to generate the dataset of the XML documents in order to prepare them to be clustered by fractal clustering technique. The XML tag content is formalized as a frequency that shows the weight of the corresponding XML item in a two-dimensional space that consists of a number of frequency vectors. In other

Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	StockQuote	0
2	Company	1
3	QuoteInfo	1
4	AFI	2
5	Price	3
6	LastUpdated	3
7	20.06	5
8	01/09/2010	6

a. S_1

Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	StockQuote	0
2	Company	1
3	QuoteInfo	1
4	AMI	2
5	Price	3
6	LastUpdated	3
7	31.52	5
8	01/09/2010	6

b. S_2

Index	Node Content	Parent Index
0	QuoteAndStatistic Response	0
1	QuoteAndStatistic	0
2	QuoteInfo	1
3	Statistic	1
4	Symbol	2
5	Price	2
6	Change	3
7	OpenPrice	3
8	Holden	4
9	24.54	5
10	+0.50	6
11	24.02	7

c. S_3

Index	Node Content	Parent Index
0	QuoteAndStatistic Response	0
1	QuoteAndStatistic	0
2	QuoteInfo	1
3	Statistic	1
4	Symbol	2
5	Price	2
6	Change	3
7	OpenPrice	3
8	Ford	4
9	28.56	5
10	-0.10	6
11	28.66	7

d. S_4

Figure 5.5: Generated matrix form of SOAP messages (S_1, S_2, S_3, and S_4)

words, every single vector V in the dataset ($V = \{x_1, x_2, ..., x_n\}$), such that $x_i = w_i$ where $w_i(i = 1, 2, ..., n)$ represents the weight of the XML item for the term t_i in the XML document. This set of frequencies shows the significance of these terms in their XML documents. The TF-IDF scheme reflects the weight of XML items within the XML document in addition to the entire set of vectors (i.e. transformed document). In other words, the significance of XML items are determined by both local (within XML document) and global (entire set of vectors) factors. XML documents have great similarity with other documents in the dataset if they share similar frequencies of their XML items. The weight of the XML item w_i in the XML document d is computed as in Eq. 5.3 below:

$$w_i(d) = tf_i \times log\frac{D}{df_i} \tag{5.3}$$

where:

- tf_i is the XML item frequency in the document d (local information),
- df_i is the number of documents containing the *ith* XML item, and
- D is the total number of XML documents in the dataset.

Equation 5.4 represents the generated vector template of SOAP messages (S_1, S_2, S_3, and S_4), and Eq. 5.5 represents the generated dataset of the same SOAP messages.

$$\begin{aligned} V_{template} = [&StockQuoteResponse, StockQuote, Company, QuoteInfo, AFI, Price \\ &, LastUpdated, 20.06, 01/09/2010, AMI, 31.52, QuoteAndStatisticResponse \\ &, QuoteAndStatistic, Statistic, Symbol,\ Change, OpenPrice,\ Holden, 24.54 \\ &,\ +0.50, 24.02,\ Ford,\ 28.56,\ -0.10,\ 28.66] \end{aligned} \tag{5.4}$$

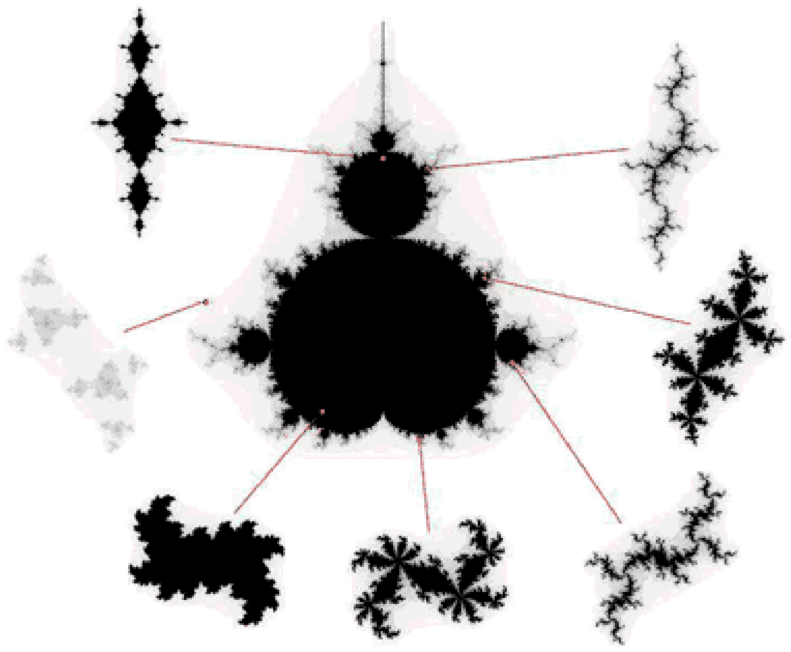
$$\begin{aligned}
V_1 &= [\ .3, .3, .3, 0, .6, 0,\ \ .3, .6, .6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\ ] \\
V_2 &= [\ .3, .3, .3, 0, 0, 0,\ \ .3, 0, 0, .6, .6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\ ] \\
V_3 &= [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .3, .3, .3, .3, .3, .3, .6, .6, .6, .6, 0, 0, 0, 0] \\
V_4 &= [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, .3, .3, .3, .3, .3, .3, 0, 0, 0, 0, .6, .6, .6, .6]
\end{aligned} \tag{5.5}$$

The process of generating vectors of the dataset in this section is summarized in algorithm 5.1. Occurrences of every single feature is counted locally in the messages and globally in other messages. Then Eq. 5.3 is applied to compute the final load of features. The complexity of algorithm 5.1 is $O(n^2)$, where n is the total distinctive XML items in the vector template.

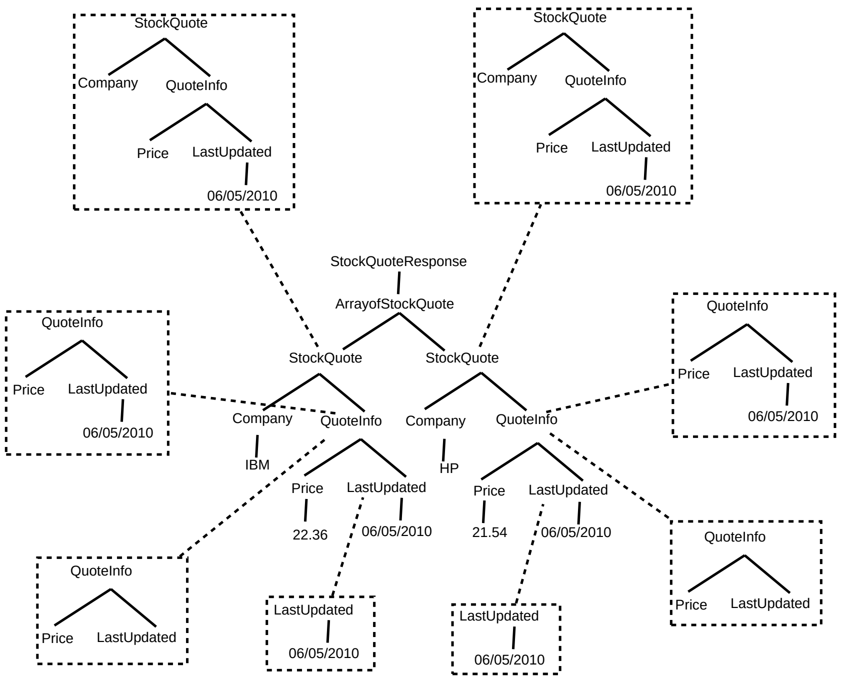
5.3 Fractal for Web service

Fractal is defined as a fragmented geometric shape that can be divided into several parts; each one is approximately a smaller copy of the whole shape [t, 1994]. Fractal has been found to form a mathematical description for the enormous and irregular shape of objects [Tao et al., 2000]. The term “fractal“ was established by Mandelbrot who derived it from the Latin fractus which is an adjective for the irregular and fragmented objects [Kumar Bisoi and Mishra, 1999]. Mandelbrot standard set shows that the fractal pattern of the whole object is the same fractal pattern of many other particular regions of the same pattern [Kumar Bisoi and Mishra, 1999]. Figure 5.6 shows Mandelbrot set and the fractal similarities within its smaller objects. These particular regions are only smaller and the same way it goes from the largest scales to the smallest. In other words, fractals are the repetition of the same structural form.

Geometric shapes are represented by fractal in a numeric form or geometric mathematical models. Fractal models are applied on geometric shapes in their numeric forms. In other words, fractal can be defined as the repetition of the same or approximately same structural form of any numeric object.



a.



b.

Figure 5.6: (a.) fractal similarity inside Mandelbrot set smaller parts and (b.) fractal similarity inside SOAP message tree branches

Fractal can be applied in web service applications using fractal self-similarity principle and other fractal characteristics that could be used in a variety of applications. In this research, the proposed fractal model suggests utilizing fractal characteristics in Web services after creating their time series representation. Fractal is proposed to compute SOAP message similarities in order to cluster them in the frequency domain of SOAP messages. This is suitable when we have large number of messages that must be clustered quickly and accurately.

5.3.1 XML fractal self-similarity

Self-similarity is the basic principle of fractals and it is the key solution to most fractal applications [Hart, 1996]. Fractals can be classified according to the type of self-similarity. There are three types of self-similarity found in fractals:

- *Exact self-similarity:* This is the strongest type of self-similarity where fractal appears identical at different scales. Fractals are defined by iterated function systems often display exact self-similarity.
- *Quasi-self-similarity:* This is a loose form of self-similarity where fractal appears approximately (but not exactly) identical at different scales. Quasi-self-similar fractals contain small copies of the entire fractal in distorted and degenerated forms. Fractals defined by recurrence relations are usually quasi-self-similar but not exactly self-similar.
- *Statistical self-similarity:* This is the weakest type of self-similarity where fractal has numerical or statistical measures which are preserved across scales. Most reasonable definitions of “fractal“ trivially imply some form of statistical self-similarity.

In the proposed technique, fractal self-similarity is applied on the numeric form of SOAP Web service messages manipulating every single message as a numeric segment. A numeric object is constructed from all the considered segments and fractal self-similarity is investigated with all the numeric segments in order to cluster them according to their similarity values.

5.3.2 Fractal iterated function system

Fractal is made up of the union of several copies of itself, each copy being transformed by a function. Iterated Function System (IFS) fractal is made up of several possibly-overlapping smaller copies of the same object, each of which is made up of copies of itself [Baharav, 1999]. Traditionally, IFS fractals are computed in 2D but they can be of any number of dimensions. For example, 3D Sierpinski triangle is a well-known example showing the self-similarity of objects in three dimensions. Fractal takes advantage of the fact that real life objects are to a great extent self-similar [Tao et al., 2000]. In other words, many parts of the object can be approximated by transforming another part of the same object by applying some affine transformation (usually linear). Based on the fractals theory, for a given object P, fractal process tries to find a Partitioned Iterated Function System (PIFS), $F = f_i : i = 1, ..., k$, which are non-overlapping tiles (usually called range blocks) of the object, where each of the "tiles" is formed by applying an affine transformation f_i on a section of P. This process can be represented as in Eq. 5.6

$$F(P) = \bigcup_{i=1}^{k} f_i(d_i) \tag{5.6}$$

where k is the number of range blocks, d_i is an arbitrary section of the numeric object, called domain. The "tile" approximated by $f_i(d_i)$ is referred to as range or r_i. Each transformation $f_i(d_i)$ gives the best possible approximation of r_i.

5.3.3 Fractal mathematical form

The general form of fractal transformation is:

$$\acute{R} = S \times D + O \tag{5.7}$$

where $\acute{R}$ is the approximated range value, D is a part of the same object (usually called domain), S and O are the scaling and shifting (offset) factors. The formula of the PIFS is applied to compute the fractals of all parts (P_i) of the object as shown in Eq. 5.8 below:

$$\acute{d} = S \times d(p_i) + O \tag{5.8}$$

where $\acute{d}$ is equivalent to the approximated range block, $d(p_i)$ is a part of the domain section. The optimal values of the coefficients can be obtained by calculating the following:

$$S = \frac{n\sum_{i=1}^{n} d(p_i)r(p_i) - \sum_{i=1}^{n} d(p_i)\sum_{i=1}^{n} r(p_i)}{n\sum_{i=1}^{n} d(p_i)^2 - \left(\sum_{i=1}^{n} d(p_i)\right)^2} \tag{5.9}$$

and

$$O = \frac{1}{n}\left(\sum_{i=1}^{n} r(p_i) - S\sum_{i=1}^{n} d(p_i)\right) \tag{5.10}$$

where n is the number of values in the object fragment, $d(p_i)$ is the value of the ith item in numeric object d, and $r(p_i)$ is the value of the ith item in numeric object r.

$$RMS = \sqrt{\frac{1}{n}[\sum_{i=1}^{n} r(p_i)^2 + S(S\sum_{i=1}^{n} d(p_i)^2 - 2\sum_{i=1}^{n} d(p_i)r(p_i) + 2O\sum_{i=1}^{n} d(p_i)) + O(nO - 2\sum_{i=1}^{n} r(p_i))]} \tag{5.11}$$

A given object is typically partitioned into k vectors. Equation 5.11 represents the criteria to investigate the similar vectors in order to assign them to their clusters. The similar messages are investigated by computing the scale and offset fractal factors for all the considered vectors derived from XML messages. Then, they are clustered based on their fractal similarity that is reflected by having the small root mean square error (RMS) inside the same group of XML messages.

Algorithm 5.1 *Build the XML Dataset*

```
01://Notation Description:
02://$D$ holds the number of XML vectors in the dataset
03://$V[D][n]$ holds the frequencies of XML items
04://$Vt[n]$ holds the nodes content of XML items (Vector template)
05://$n$ holds the total number of the distinctive XML items in the vector template
06://$Mx$ holds the current matrix form of the current XML document
07:$i \leftarrow 0 // Counter\ Initialization$
08:$j \leftarrow 0 // Counter\ Initialization$
09:Repeat
10:$Mx \leftarrow load\ the\ current\ (ith)\ matrix\ form$
11:for all $nodes\ content\ in\ the\ Vt$ do
12:$Nd \leftarrow Vt[j]$ // $get\ the\ current\ node\ in\ the\ vector\ template$
13:$j \leftarrow j + 1$
14:$F \leftarrow Count(Nd, Mx)$ //$Count\ all\ the\ occurrences\ of\ Nd\ in\ Mx$
15:$G \leftarrow Count(Nd, D)$ // $Count\ the\ number\ of\ documents\ having\ Nd$
16:$V[i][j] \leftarrow F \times log\frac{D}{G}$
17:end for
18:$i \leftarrow i + 1$
19:Until $i = n$
```

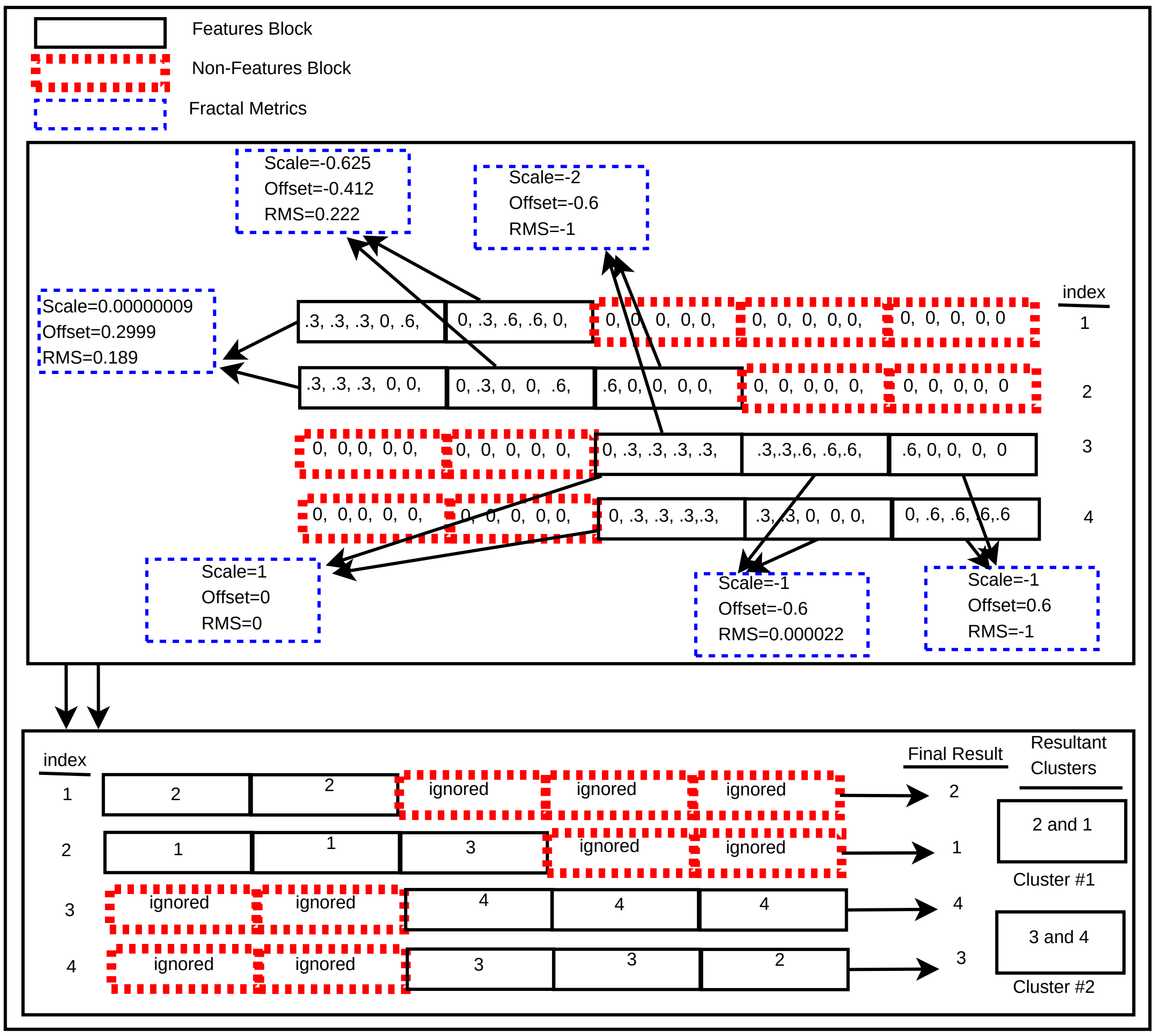


Figure 5.7: Fractal similarity of SOAP messages inside the numeric dataset and final clusters. The fractalization process starts after building the numeric form of SOAP messages with a one dimensional numeric vector for each SOAP message allocated as one row in the final two-dimensional matrix (dataset) that represent all messages. The major steps are: 1. Break the vectors into smaller blocks, 2. Flag zero blocks as ignored to be excluded from the fractal computations because these blocks mean their features are not existent in the message (non-feature blocks), 3. Compare every block with all other feature blocks located in the same column with fractal factors (scale, offset, and RMS) to find the most similar one that created the smallest RMS, 4. Assign the block with the message index of the similar block, 5. Histogram assigned indexes (indexes represent message references) for every single vector and cluster them with the message of the highest index appearance of the histogram

Algorithm 5.2 *Fractal Coefficients Computations*

```
01://Notation Description:
02://$FBS$ holds the fractal block size
03://$FBn$ holds the number of blocks in XML vector
04://$Sn$ holds the number of XML vectors in the dataset
05://$Vs$ holds the number of frequencies per vector
06://$V[Sn][Vs]$ vectors of the generated dataset
07://$Flg[Sn][FBn]$ holds the flags to recognize the ignored blocks
08:for i = 0 To Sn - 1 do// All vectors
09:for j = 0 To FBn - 1 do All blocks in vector
10:$Flg[i][j] \leftarrow False$// Flag initialization
11:for co = 0 To FBS - 1 do All frequencies in the block
12:$FSL \leftarrow j \times FBS$// Determine the start location of the required frequency
13:If $V[i][FSL + co] \neq 0$
14:$Flg[i][j] \leftarrow True$// not ignored block
15:Break the loop
16:end If
17:end for
18:If $Flg[i][j] = True$// not ignored block
19:$R[i][j] \leftarrow 0$//Initialization
20:$Rs[i][j] \leftarrow 0$//Initialization
21:for co = 0 To FBS - 1 do All frequencies in the block
22:$R[i][j] \leftarrow R[i][j] + V[i][FSL + co]$ //$\sum r_i$ fractal coefficient
23:$Rs[i][j] \leftarrow Rs[i][j] + Sqr(V[i][FSL + co])$ //$\sum {r_i}^2$ fractal coefficient
24:end for
```

....**Continue**

```
25:$Das[i][j] \leftarrow Sqr(R[i][j])$ //$(\sum d_i)^2$ $fractal\ coefficient$
26:$D[i][j] \leftarrow R[i][j]$ //$\sum d_i$ $fractal\ coefficient$
27:$Ds[i][j] \leftarrow Rs[i][j]$ //$\sum {d_i}^2$ $fractal\ coefficient$
28:end If
29:end for
30:end for
```

5.4 Fractal coefficients and RMS

Fractal mathematical models are well-known as time consuming techniques [Hart, 1996]. With the aim to reduce the required computations of fractal technique, fractal redundant coefficients are calculated in advance. As a result, the required processing time for the fractal clustering algorithm is reduced significantly as most of the major coefficients have already been computed and buffered. This process is summarized as in algorithm 5.2.

After investigating the main fractal equations **??**, **??** and 5.11, five major fractal coefficients are selected to be computed in advance as listed below:

- $\sum r_i$: summation of the ith range block in the considered vector in the generated dataset.
- $\sum {r_i}^2$: summation of the squared values of the ith range block of the XML vector.
- $\sum d_i$: summation of the ith domain block in the considered vector in the generated dataset.
- $\sum {d_i}^2$ summation of the squared values of the ith domain block of the XML vectors.
- $(\sum d_i)^2$: summation of the squared value of the ith domain block.

Another strategic step of the proposed fractal clustering model is considering the same range blocks of the generated XML vectors as domain blocks excluding the currently selected range block. Therefore, every single block in the generated dataset vectors is having the same fractal coefficients as range and domain block. Technically, the fractal coefficients $\sum r_i$ and $\sum {r_i}^2$ will be equal to $\sum d_i$ and $\sum {d_i}^2$ respectively. Therefore, only one set needs to be computed as they are duplicated from the range block coefficients.

As previously stated, the resultant frequencies in the generated vectors of the dataset represent the actual properties of the XML messages. According to the proposed fractal strategy which breaks up these vectors into equal sized blocks, some of these blocks have zeros only as some of the XML items are non-existent in their XML messages. In algorithm 5.2, these zero frequency blocks are identified by checking and flagging them as ignored blocks as they do not have any impact on the clustering distributions of the final XML messages. Flagging these blocks and removing them from the computations of the fractal coefficients can potentially minimize the processing time. Figure 5.7 shows fractal similarities inside the SOAP messages numeric particles and explains the fractal similarity based clustering process. Fractal factors (scale, offset, and RMS) are computed for the current selected feature-block with all other feature-blocks that are located on the same column in order to to find the closest matching block that has the smallest RMS. Blocks are assigned with message index of similar block in other messages.

Algorithm 5.3 *Fractal RMS metric*

```
01://**Notation Description:**
02://$FBS$ holds the fractal block size
03://$FBn$ holds the number of blocks in XML vector
04://$Sn$ holds the number of XML vectors in the dataset
05://$Vs$ holds the number of frequencies per vector
06://$V[Sn][Vs]$ vectors of the generated dataset
07://$Flg[Sn][FBn]$ holds the flags to recognize the ignored blocks
08:**for i = 0 To Sn - 1 do**// *All vectors*
09:**for j = 0 To FBn - 1 do** *All blocks in vector*
10:**If** $Flg[i][j] = True$// *not ignored block*
11:$RMSo = 500000$// *Initializing the RMS error with high value*
12:**for k = 0 To Sn - 1 do**
13:**If** $k \neq i$
14:$RD \leftarrow 0$//*Initialization*
15:$FSL \leftarrow j \times FBS$
16:**for co = 0 To FBS - 1 do**
17:$RD \leftarrow RD + V[i][FSL + co]\times$
18:$V[k][FSL + co]$
19:**end for**
20:$Scale \leftarrow (FBS \times RD - R[i][j] \times D[k][j])/(FBS \times Ds[k][j] - Das[k][j])$
21:$Offset \leftarrow (R[i][j] - Scale \times D[k][j])/FBS$
22:$RMSn \leftarrow Sqrt((Rs[i][j] + Scale \times (Scale\times$
23:$Ds[k][j] - 2 \times RD + 2 \times Offset\times$
24:$D[k][j]) + Offset \times (FBS \times Offset$
```

....Continue

```
25:$-2 \times R[i][j]))/FBS)$
26:**If** $RMSn < RMSo$
27:$S[i][j] \leftarrow k$
28:$RMSo \leftarrow RMSn$
29:**end If**
30:**end If**
31:**end for**
32:**end If**
33:**end for**
34:**end for**
```

Fractal Root Mean Square error (RMS) is the basic metric of the proposed clustering technique that determines block similarities as all the generated vectors in the dataset are broken up into equal sized blocks. The computations of the RMS metric is based on comparing the resultant RMS values of the blocks that are located on the same column in the dataset with different vectors (blocks on the same column reflect the same features). The smallest RMS value with the considered block means higher similarity of their template features (XML items). Algorithm 5.3 is required to compute the RMS metric and fractal similarity. It creates a decision matrix that refers to the closest XML sample of every single block.

Algorithm 5.4 *Histogram Vectors Distribution*

```
01://Notation Description:
02://$FBS$ holds the fractal block size
03://$FBn$ holds the number of blocks in XML vector
04://$Sn$ holds the number of XML vectors in the dataset
05://$Sindex$ holds the current sample index
06://$Hist[Sn]$ holds similar samples histogram
07://$Flg[Sn][FBn]$ holds the flags to recognize the ignored blocks
08://$FClust[Sn]$ holds the final samples distribution
09:for i = 0 To Sn - 1 do// All vectors
10:for j = 0 To FBn - 1 do All blocks in vector
11:$SIndex \leftarrow S[i][j]$
12:$Hist[Sindex] \leftarrow Hist[Sindex] + 1$
13:end for
14:$MaxIndex \leftarrow 0$
15:for j = 0 To Sn - 1 do
16:If $Hist[j] > MaxIndex$
17:$MaxIndex \leftarrow Hist[j] : SIndex \leftarrow j$
18:end If
19:end for
20:$FClust[i] \leftarrow SIndex$
21:end for
```

The generated decision matrix is the key solution to finalize allocating messages based on the maximum histogram of sample indexes in every single vector. Algorithm 5.4 is required to compute the histogram of all the existent sample indexes of every single vector in the generated dataset and distribute them according to the maximum histogram of these sample indexes in the decision matrix. For example, in Fig. 5.7, vector blocks of the first indexed messages ([0.3, 0.3, 0.3, 0, 0.6] and [0, 0.3, 0.6, 0.6, 0]) are checked with other vector blocks located on the same column and the best matched blocks are [0.3, 0.3, 0.3, 0, 0] and [0, 0.3, 0, 0, 0.6] respectively from the second indexed message. The smallest RMS for the best matches are 0.189 and 0.222 respectively and therefore both messages are clustered together based on their resultant histograms for the similar indexes.

5.5 Experiments and discussion

With the aim of showing an accurate assessment of the proposed fractal clustering technique, the experimental evaluation has considered a wide variety of SOAP messages size. These samples include real small messages (i.e. only 140 bytes) as well as very large messages that could be as large as 53 kbytes. Moreover, the compression-based aggregation model of SOAP Web messages (proposed in chapter 4) is considered as a tool to demonstrate the superior ability of the proposed fractal clustering model when compared with other standard clustering techniques. The basic testing criteria is by applying the compression-based aggregation model on the resultant clusters and measuring the achievable compression ratios on these clusters. Then, comparing them with other standard techniques as the one that could achieve higher compression ratio is the one can achieve better clustering. Furthermore, local error rate (RMS) inside clusters in addition to the global error rate which investigates the similarity level are measured for every single cluster.

A testbed of 160 SOAP messages has been configured that consists of four groups each with 40 messages. The testing SOAP messages are allocated to these groups based on their size as they are classified as small, medium, large, and very large sized messages. They have the ranges of 140-800, 800-3000, 3000-20000, and 20000-55000 bytes respectively. Both K-Means and PCA combined with K-Means [Liu et al., 2004] are implemented in this research to evaluate the proposed technique by comparing their resultant compression ratios that could be achieved on their resultant SOAP message clusters in addition to their required processing time. K-Means and PCA combined with K-Means are applied on the generated dataset with different cluster numbers that start from only two and up to ten clusters. On the other hand, fractal model is developed and applied to work on a fractal block size that can be pre-determined by the developer. In this work, fractal block sizes are 10%, 20%, 25%, 50%, and 100% of the overall considered vectors size in the generated dataset of the XML messages.

All techniques showed significant results by enabling the compression-based aggregation tools to achieve potentially high compression ratios on the clustered SOAP messages. Table 5.1 summarizes the performance of all the clustering techniques on SOAP message groups. It shows the resultant average compression ratios of all the experiments with different numbers of cluster and fractal block size percentage. Furthermore, it states the average time to cluster 40 SOAP messages of each group in the generated dataset. In this table, the fractal model displays better performance than other models in terms of enabling the aggregation model to achieve higher compression ratios. Moreover, the table shows that the processing time required to cluster SOAP messages is potentially reduced by the fractal based clustering technique in comparison with other standards. Figures 5.8, 5.9, 5.10, 5.11,and 5.12 depict the significant ability of reducing the overall size of the clustered SOAP messages. This is achieved by aggregating messages of each cluster after clustering them using fractal clustering technique with fractal block sizes: 10%, 20%, 25%, 50%, and 100% from the overall vector size by Binary-tree, Two-bit, and One-bit aggregations models.

Figures 5.13, 5.14, and 5.15 illustrate the detailed results of the achievable average compression ratios by the compression-based aggregation tools after clustering SOAP messages using K-Means, PCA combined with K-Means, and the proposed technique respectively with different numbers of clusters and block sizes. It is clear that, as the number of clusters decreases, higher compression ratio can be achieved. Both Huffman and fixed-length based

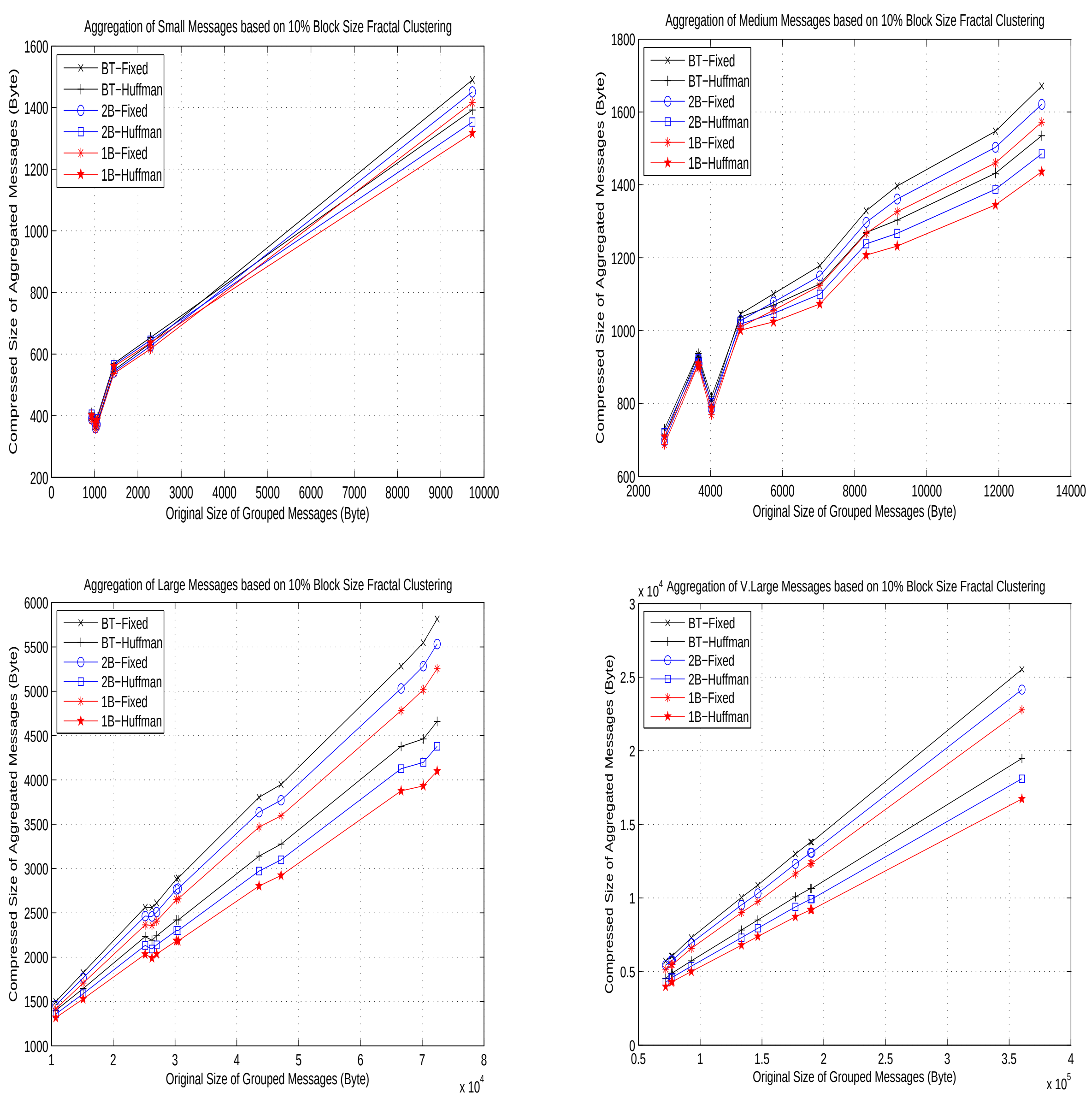


Figure 5.8: Compressed size of small, medium, large, and very large aggregated messages with 40 messages each group by BT (Binary-tree), 2B (Two-bit), and 1B (One-bit) aggregation techniques based on 10% block size of fractal clustering

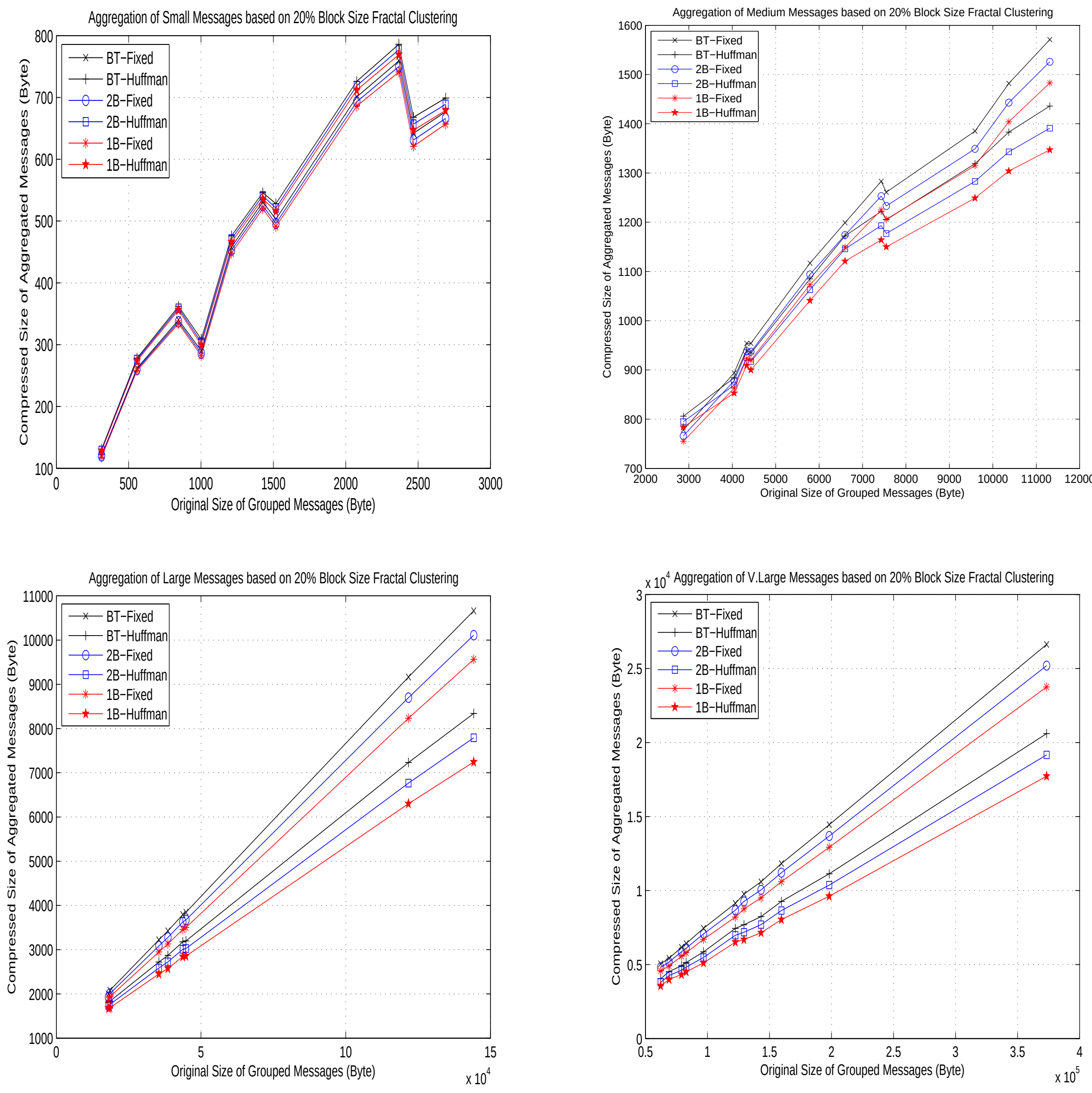


Figure 5.9: Compressed size of small, medium, large, and very large aggregated messages with 40 messages each group by BT (Binary-tree), 2B (Two-bit), and 1B (One-bit) aggregation techniques based on 20% block size of fractal clustering

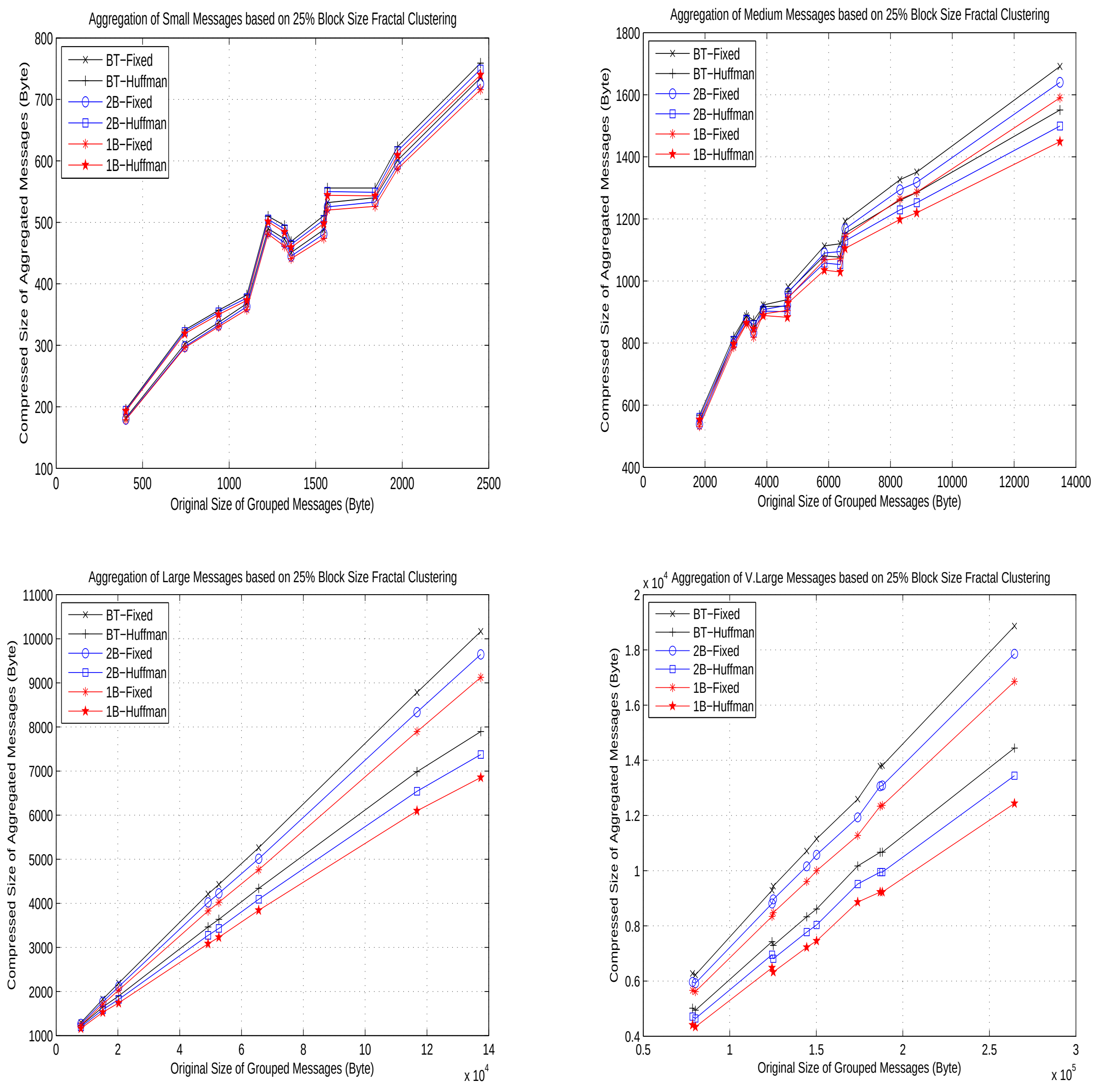


Figure 5.10: Compressed size of small, medium, large, and very large aggregated messages with 40 messages each group by BT (Binary-tree), 2B (Two-bit), and 1B (One-bit) aggregation techniques based on 25% block size of fractal clustering

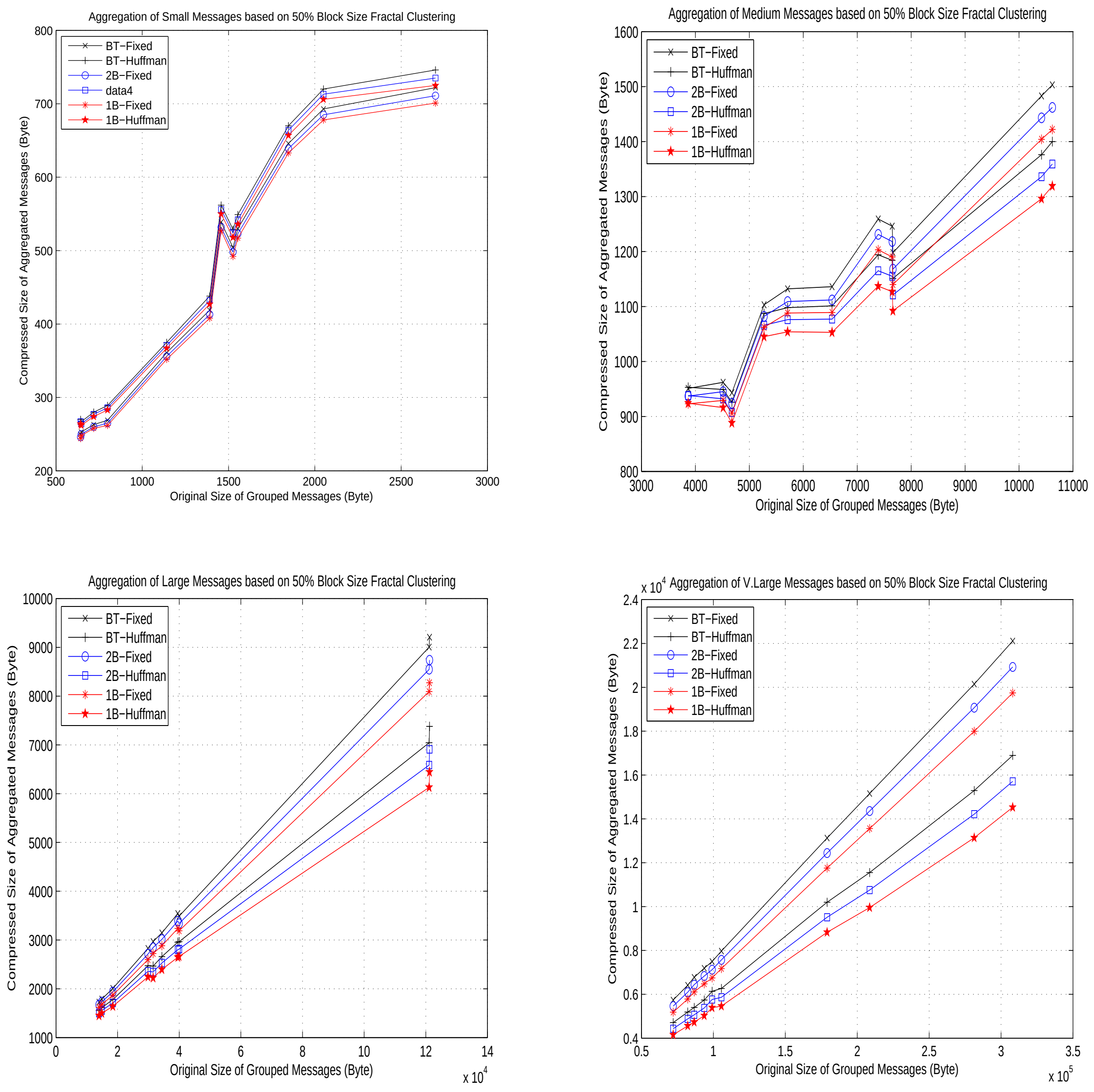


Figure 5.11: Compressed size of small, medium, large, and very large aggregated messages with 40 messages each group by BT (Binary-tree), 2B (Two-bit), and 1B (One-bit) aggregation techniques based on 50% block size of fractal clustering

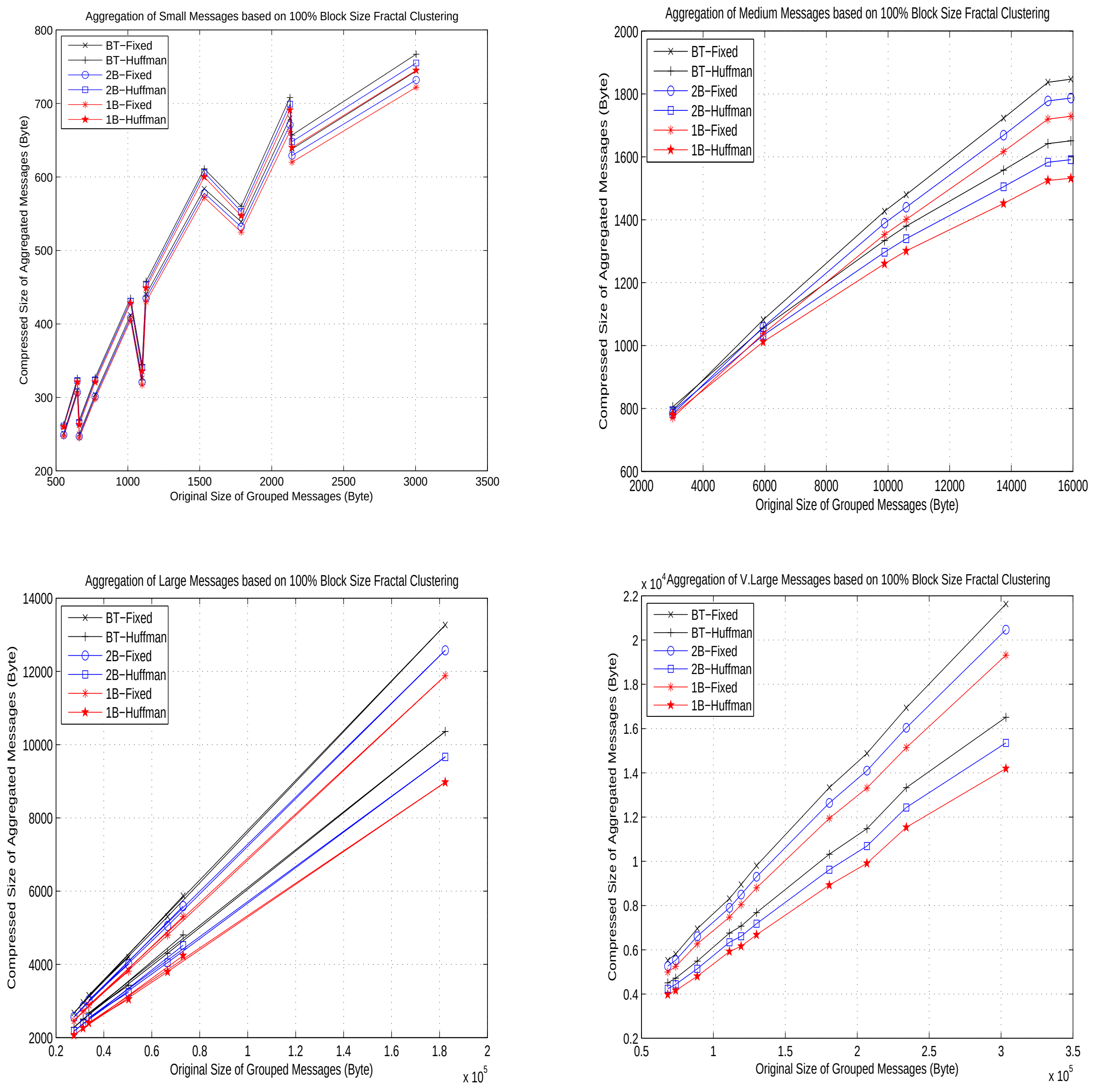


Figure 5.12: Compressed size of small, medium, large, and very large aggregated messages with 40 messages each group by BT (Binary-tree), 2B (Two-bit), and 1B (One-bit) aggregation techniques based on 100% block size of fractal clustering

aggregations are applied on the resultant clusters of both fixed cluster numbers clustering techniques and Huffman. They clearly showed better performance by achieving higher compression ratios than fixed-length encoding. On the other hand, fractal block size has significant impact on the overall performance of the fractal clustering technique as fractal block size decreases better aggregation can be achieved (i.e. higher compression ratios). The basic explanation of this fact is when XML messages are broken up into more blocks, more object features can be caught precisely leading to better fractal similarity measurements. Thus, the more similar messages are clustered in one cluster. The fractal clustering technique has shown better performance in terms of supporting the aggregation model to reduce the network traffic significantly in comparison with other techniques.

Experimental results clearly demonstrated that the proposed technique has clustered SOAP messages with high level of similarity by selecting messages with the smallest error (RMS) with the centroid point of the cluster. Tables 5.2 and 5.3 show the minimum, maximum, and average error values (RMS) in order to investigate both local and global similarities within the same cluster and with other clusters. The results showed higher RMS values for global similarities than local measurements as messages have less error rate inside their clusters.

With the aim to evaluate the processing time of the technique, clustering time has been investigated and compared with both K-Means and PCA combined with K-Means. Figure 5.16 shows the average processing time of all the considered clustering techniques in details for small, medium, large, and very large XML messages. PCA combined with K-Means requires more processing time than K-Means as it requires more computations to implement the PCA for the SOAP messages. However, the proposed technique shows a significant processing time in comparison with other techniques as it mainly requires about 15 milliseconds to cluster 40 SOAP messages while other techniques require between 50 to 70 milliseconds. Processing time for the dataset generation is investigated to give an accurate evaluation to the overall

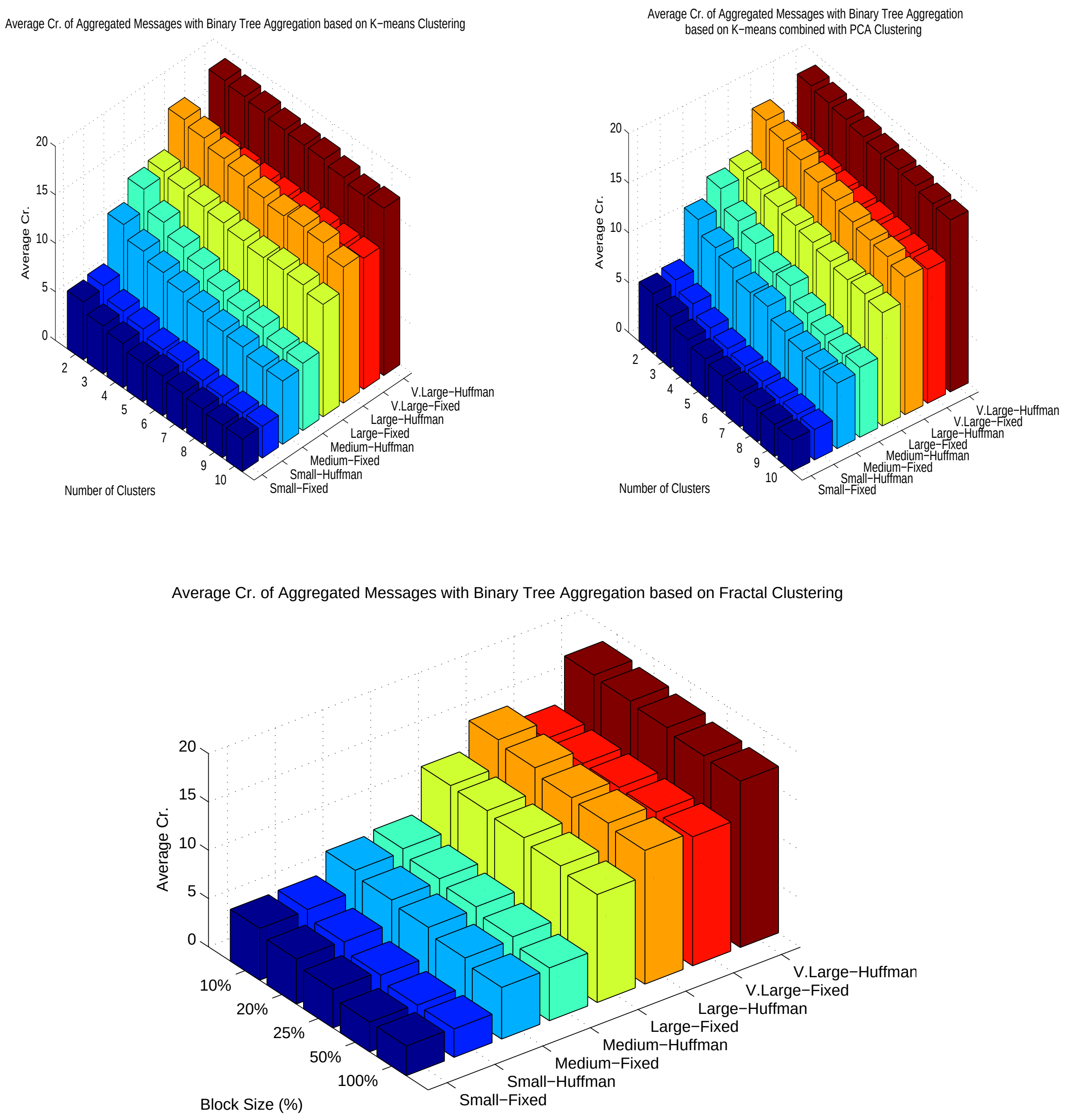


Figure 5.13: Average compression ratios of the aggregated SOAP messages using Binary-tree aggregation technique based on K-Means, K-Means combined with PCA, and fractal clustering techniques

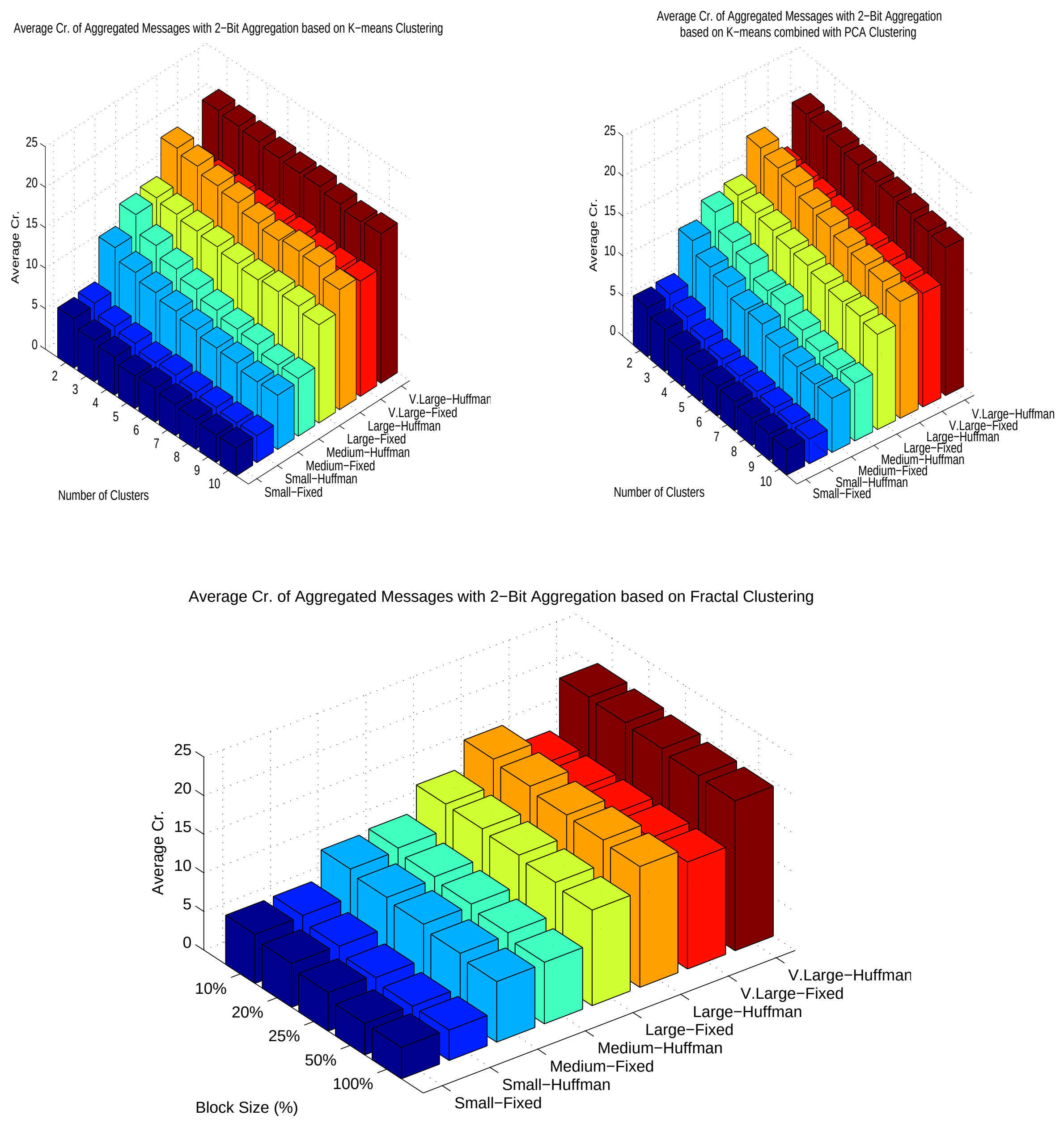
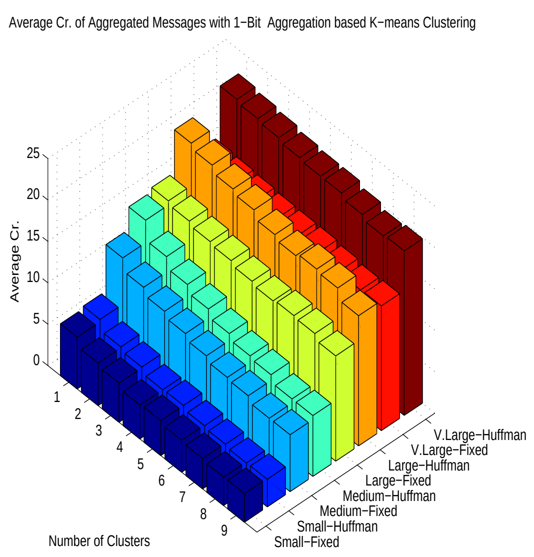
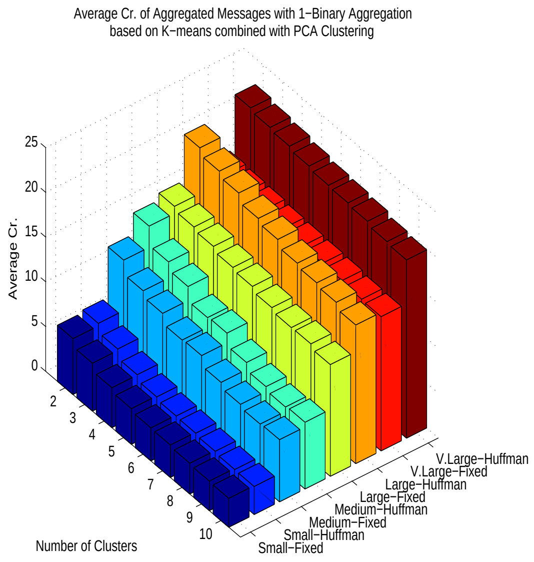


Figure 5.14: Average compression ratios of the aggregated SOAP messages using Two-bit aggregation technique based on K-Means, K-Means combined with PCA, and fractal clustering techniques





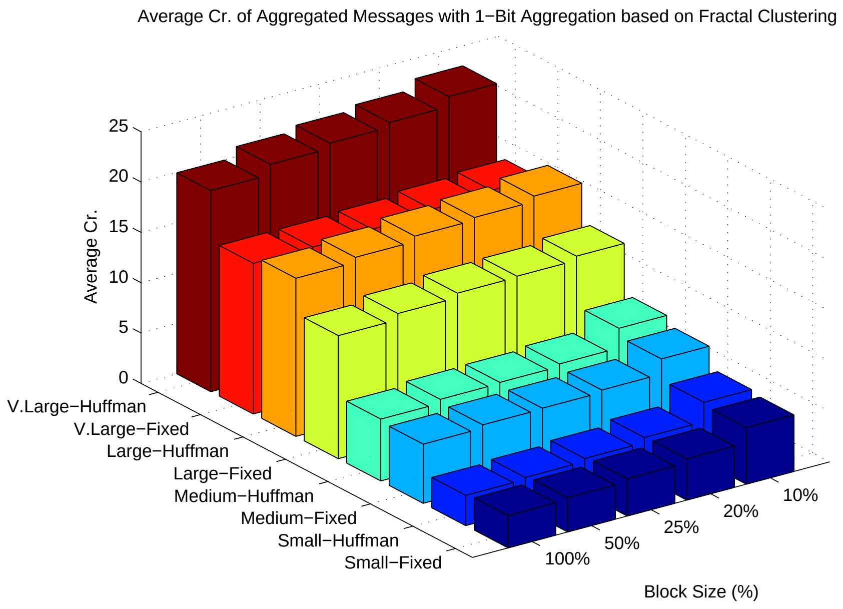


Figure 5.15: Average compression ratios of the aggregated SOAP messages using One-bit aggregation technique based on K-Means, K-Means combined with PCA, and fractal clustering techniques

Table 5.1: Average compression ratios and clustering time of K-Means, PCA + K-Means, and fractal clustering models for small, medium, large, and very large messages

		K-Means	**PCA + K-Means**	**Fractal**
40	fixed-length Average Cr.	3.920295	3.850353	4.002463
Small	Huffman Average Cr.	3.820822	3.7136433	3.92449
messages	Average Clustering Time (Ms)	50.8831	65.3333	15.6249
40	fixed-length Average Cr.	6.766314	6.797503	7.275127
Medium	Huffman Average Cr.	7.715699	7.843987	7.985582
messages	Average Clustering Time (Ms)	52.3342	62.8888	15.8441
40	fixed-length Average Cr.	12.943645	12.815021	13.100785
Large	Huffman Average Cr.	16.020012	16.279294	16.633852
messages	Average Clustering Time (Ms)	54	68.1111	15.7241
40	fixed-length Average Cr.	15.109293	15.127478	15.334334
Very large	Huffman Average Cr.	20.163554	20.253857	21.7001
messages	Average Clustering Time (Ms)	53.6231	70.4444	15.6383

Table 5.2: Clusters Local RMS errors and resultant clusters numbers and sizes

Messages Size	Block Size Percent	Min. RMS	Max. RMS	Average RMS	Min. Cluster Size	Max. Cluster Size	Clusters Number
Small messages	10%	0	0.5047	0.2605	2	27	6
	20%	0	0.9191	0.45306	2	6	11
	25%	0.3022	0.9876	0.5227	2	5	12
	50%	0.2576	0.974	0.54172	2	7	12
	100%	0	0.8181	0.53623	2	6	12
Medium messages	10%	0	1.6317	0.88507	2	7	11
	20%	0	1.8818	1.08179	2	7	11
	25%	0	2.1393	1.24371	2	7	13
	50%	0	1.821	1.18298	2	6	12
	100%	0	1.8448	1.18366	2	9	8
Large messages	10%	0	4.4824	2.46939	2	5	12
	20%	0	5.0462	2.8716	2	9	8
	25%	0	5.2648	3.04482	2	10	8
	50%	0	4.8300	3.06731	2	10	10
	100%	0	4.5297	2.92064	2	14	7
Very large messages	10%	0	6.9137	4.60908	2	9	10
	20%	0	10.079	5.72626	2	10	11
	25%	3.5036	8.7243	5.8157	2	7	10
	50%	3.3639	8.8624	5.85116	2	8	10
	100%	0	7.8088	5.56375	2	9	11

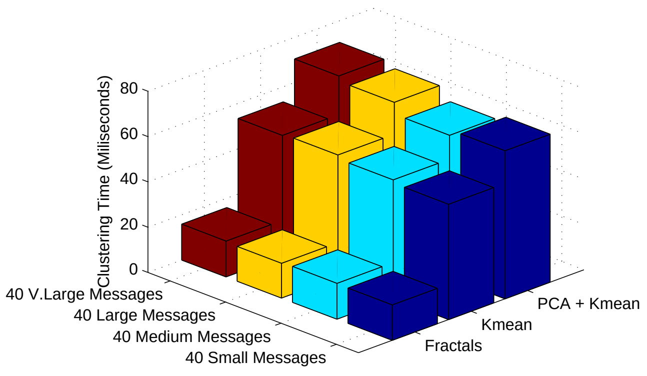


Figure 5.16: The average clustering time of K-Means, PCA combined with K-Means and fractal model of small, medium, large, and very large SOAP messages

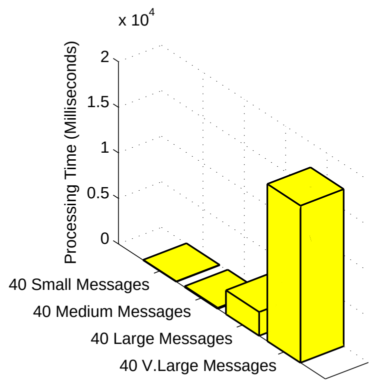


Figure 5.17: The required processing time to generate the dataset Vectors of small, medium, large, and very large SOAP messages

Table 5.3: Clusters Global RMS errors

Messages Size	Block Size Percent	Min. RMS	Max. RMS	Average RMS
Small messages	10%	0.187966	0.514816	0.330745
	20%	0.228095	0.906718	0.488379
	25%	0.188495	0.84212	0.546586
	50%	0.261493	1.040522	0.581153
	100%	0.247343	0.979189	0.652142
Medium messages	10%	0.640369	1.637905	0.960197
	20%	0.845921	2.086727	1.290225
	25%	0.809719	2.191491	1.301323
	50%	1.115513	2.503145	1.656588
	100%	1.307663	1.916698	1.614943
Large messages	10%	1.586618	5.678355	3.393421
	20%	2.389237	5.422692	3.444342
	25%	2.114001	6.160388	4.097186
	50%	3.102103	8.144498	4.965433
	100%	3.299164	5.342282	4.21083
Very large messages	10%	3.339759	8.11057	5.561411
	20%	3.475809	10.19773	6.441515
	25%	4.211633	9.365102	7.110114
	50%	5.469511	11.3655	7.910134
	100%	5.203311	9.729922	7.023571

requirements of the clustering models and it is obvious that it changes with the size of SOAP messages where large messages require more processing time (as shown in Fig. 5.17).

5.6 Conclusion

Web scenarios and applications with the aggregation of SOAP messages are significantly strengthened by clustering models as potential alternatives to the traditional simple cost similarity measurements. Network traffic would be highly reduced through the ability of aggregating large number of SOAP messages and not only pairs of messages. Fractal coefficients are introduced as efficient similarity measurements to SOAP messages utilizing the self-similarity principle of fractal mathematical model. The proposed clustering model is developed to compute fractal similarity of the proposed numeric form of SOAP messages. The experimental results have shown higher performance to the proposed technique than well-known clustering techniques such as K-means and PCA combined with K-means. Finally, the resultant improvement in the performance of SOAP would be able to support many Web scenarios and applications such as Cloud Web services, low bandwidth environments, and mainly the low connectivity devices like PDAs and regular mobiles.

Chapter 6

Distributed Aggregation and Fast Fractal Clustering

This chapter tries to answer the fourth research question posted in section 1.2 by developing the aggregation of SOAP messages to aggregate messages at several nodes (servers) working on the same network and improving the performance of fractal clustering technique in terms of the processing time.

The chapter is organized as follows: section 6.1.1 discusses the motivation for this research. Section 6.1.2 describes the proposed techniques. Section 6.1.3 illustrates the evaluation strategy for the proposed techniques. Furthermore, section 6.2 briefly describes the core aggregation model that represents the infrastructure base for the new proposed accumulating aggregation models. Section 6.3 illustrates the proposed distributed aggregation model in detail. Moreover, the proposed development to the fractal clustering technique is shown in section 6.4. Section 6.5 shows the experiments and results. Finally, section 6.6 concludes the chapter.

6.1 Introduction

SOAP has reflected the technical disadvantages of XML in generating bigger Web messages than the real payload of the requested services over the Internet.

6.1.1 Motivation

With the aim to improve the performance of Web services, new compression and aggregation approaches for SOAP messages are proposed in chapters 3 and 4 respectively. Aggregation models include computing the redundancies that can be found in SOAP messages and aggregating them with a new compression-based technique to ensure reducing their size significantly. However, these approaches are designed as standalone (at one server) techniques without sharing other servers to aggregate more messages with the same compact packet of the previously aggregated messages (at predecessor machine).

Furthermore, aggregation of SOAP requires significant similarity measurements for a potential reduction in network traffic. Similarity measurements and clustering (based on message similarity) are proposed by this research for better Web service performance. Fractal is introduced (in chapter 5) as a new potential similarity measurement for SOAP messages with the aim of supporting aggregation models with high similar clusters of SOAP messages. However, fractal models are time consuming for large dataset, representing a bottleneck for fractal clustering. Moreover, distributed aggregation will increase the number of messages for aggregation and requires efficient similarity measurements.

6.1.2 Proposed solution

A novel distributed aggregation technique is proposed to provide aggregation of SOAP messages from several nodes (servers) over the network. The aggregation starts by excluding the XML items of newly added messages that have already appeared in the core aggregated message to remove redundant items. A new modified Huffman tree encoding method has been developed, allowing the tree to be extended with more nodes without the need to rebuild the Huffman tree from scratch. Finally, the accumulating lookup table of the new accumulating aggregated packet is built. Figure 6.1 shows the main functions of the proposed distributed aggregation model.

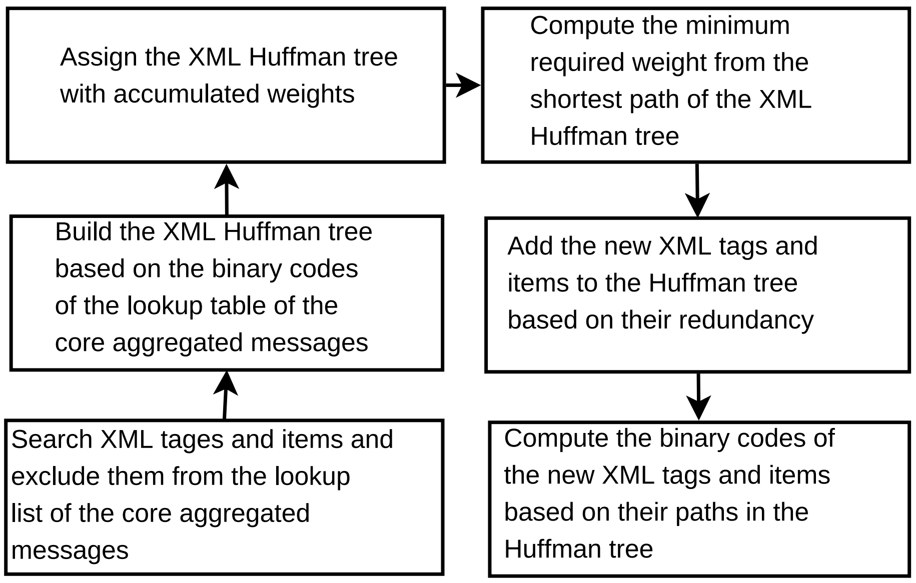

Assign the XML Huffman tree with accumulated weights
Compute the minimum required weight from the shortest path of the XML Huffman tree
Build the XML Huffman tree based on the binary codes of the lookup table of the core aggregated messages
Add the new XML tags and items to the Huffman tree based on their redundancy
Search XML tages and items and exclude them from the lookup list of the core aggregated messages
Compute the binary codes of the new XML tags and items based on their paths in the Huffman tree

Figure 6.1: Main components of the distributed aggregation model

Fractal clustering for SOAP messages is a potential similarity measurement to support aggregation with accurate predictions of SOAP clusters. However, fractal clustering is time consuming especially when dealing with a large number of messages (up to thousands). The fast fractal similarity measurements are proposed in this chapter to utilize their potential with the new proposed aggregation of SOAP messages. Fast fractal introduces new mathematical factors that are computed in advance to the regular fractal process, in order to classify fractal object blocks into distinctive segments. Then, the fractal coefficients (scale, offset, and RMS) are computed for blocks that only have the same fractal factor value. The proposed model starts by generating the dataset from the SOAP messages, creating a set of numeric vectors showing the local and global loads of XML items. Next, fast fractal factors are computed in advance, then the fractal coefficients are computed, and finally cluster the XML messages based on the maximum histograms of fast fractal indexes in every single message. Figure 6.2 shows the main functions of the proposed fast fractal clustering model.

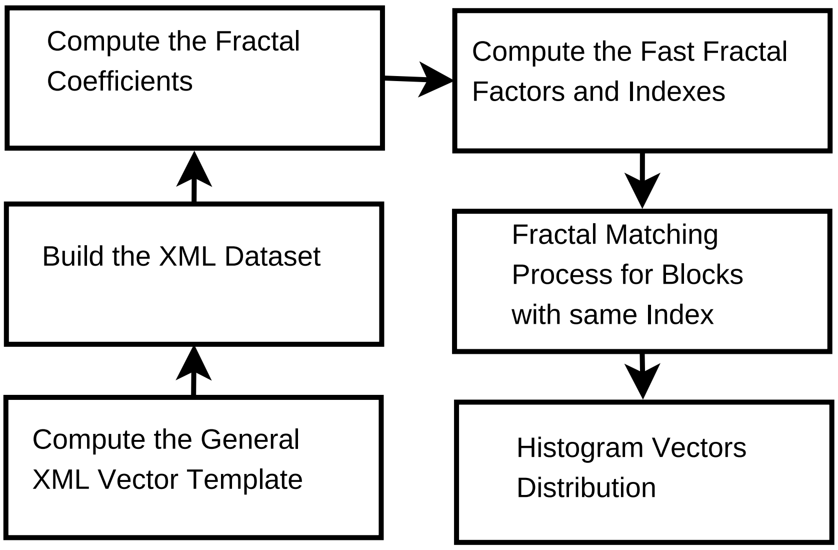

Compute the Fractal Coefficients
Compute the Fast Fractal Factors and Indexes
Build the XML Dataset
Fractal Matching Process for Blocks with same Index
Compute the General XML Vector Template
Histogram Vectors Distribution

Figure 6.2: Main components of fast fractal model

Fractal similarity measurements and aggregation of SOAP messages support Web services in several scenarios and applications including Web applications over the Cloud. Figure 6.3 illustrates the utilization of aggregation models over Cloud computing including low bandwidth constrained environments.

6.1.3 Evaluation strategy

The evaluation of the proposed distributed aggregation and fast fractal is mainly focusing on investigating the required processing time and the messages size reduction. The distributed aggregation has significantly outperformed the standalone aggregation model in terms of reducing the total size of the aggregated messages. On the other hand, the results have shown a tremendous minimization of the clustering time of the fast fractals in comparison with the classical fractal. Furthermore, the processing time of the proposed clustering models are compared with both K-means and Principle Component Analysis (PCA) combined with

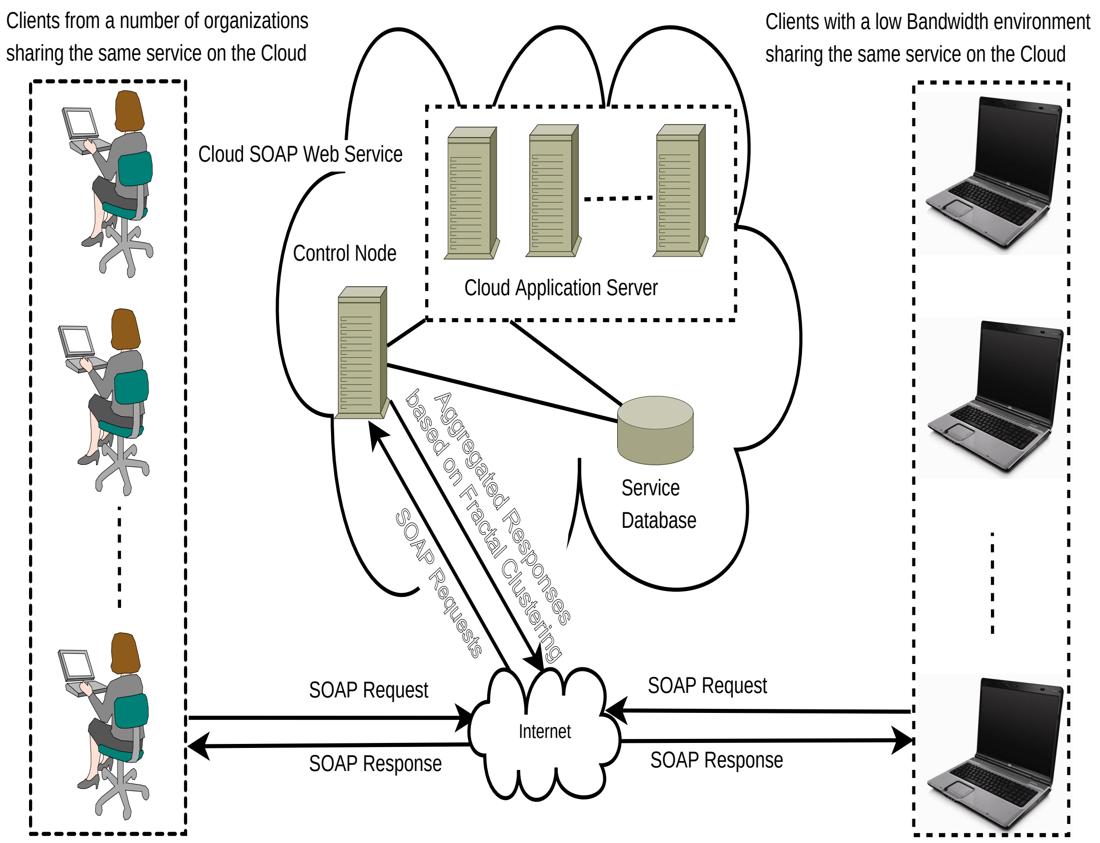

Clients from a number of organizations sharing the same service on the Cloud
Cloud SOAP Web Service
Control Node
Cloud Application Server
Service Database
Aggregated Responses based on Fractal Clustering
SOAP Requests
Clients with a low Bandwidth environment sharing the same service on the Cloud
SOAP Request
SOAP Response
Internet
SOAP Request
SOAP Response

Figure 6.3: Cloud Web services scenarios including high and low bandwidth constrained environments

K-means as two standard clustering models. The proposed fast fractal clustering models have outperformed all the other considered clustering models in terms of the clustering time.

Furthermore, the compression-based aggregation model proposed in this research is used as a tool to compute the SOAP messages size reduction after aggregating the fractal based clustered messages. The proposed fast fractal clustering models have shown almost equal efficiency of the classical fractal model in reducing the overall size of SOAP messages. Moreover, they have been compared with both the K-means and PCA combined with K-means as the fractal clustering models enable the compression-based aggregation model to achieve higher messages size reduction.

6.2 Core aggregation of SOAP messages

This section overviews the basic aggregation model suggested in chapter 4, used as the core component of the proposed aggregation model. Redundancy-aware aggregation of SOAP messages starts with building the XML trees for the aggregating messages. Figures 6.4 and 6.5 show an example of three XML messages (S_1, S_2, and S_3) and their corresponding XML trees. Next, XML trees are transformed into matrix form by indexing all the XML items of these trees and organizing them into a matrix form (see Figure 6.6). Breadth-first traversal is used to generate a textual format, assigning all the XML items with one bit (0 or 1) in order to recognize the positions and provide the ability to rebuild the matrix form again. Huffman tree encoding is used to include all the generated textual expressions and encode them with one common lookup table. Figure 6.7 illustrates the strategy of computing the aggregated textual expressions lookup table for two SOAP messages S_1 and S_2. Finally, Huffman encoding is used to finish the encoding of the textual expressions using the mapping binary codes of the XML items in the common lookup table. Figure 6.8 shows the final structure of the compact aggregated SOAP messages S_1 and S_2.

6.3 Distributed aggregation of SOAP messages

The main target of the distributed aggregation is to aggregate m messages with another n messages that have already been aggregated by another node. Figure 6.9 shows the distributed aggregation scenario for SOAP messages over the network. There are two challenges that need to be covered in order to accomplish distributed aggregation. The first is to develop Huffman tree encoding to generate more binary paths (binary codes) without rebuilding the pre-generated Huffman tree of the core aggregated messages. The second challenge is to design the compact packet structure of the core aggregated messages to be adaptive for the new added accumulating aggregated messages. Both challenges have been addressed with the

```
< StockQuoteResponse >
< StockQuote >
< Company > MSFT
< /Company >
< QuoteInfo >
< Price > 26.03 < /Price >
< LastUpdated > 27/12/2011
< /LastUpdated >
< /QuoteInfo >
< /StockQuote >
< /StockQuoteResponse >
```

a.

```
< StockQuoteResponse >
< StockQuote >
< Company > UAL
< /Company >
< QuoteInfo >
< Price > 19.85 < /Price >
< LastUpdated > 27/12/2011
< /LastUpdated >
< /QuoteInfo >
< /StockQuote >
< /StockQuoteResponse >
```

b.

```
< StockQuoteResponse >
< StockQuote >
< Company > HMC
< /Company >
< QuoteInfo >
< Price > 30.26 < /Price >
< LastUpdated > 27/12/2011
< /LastUpdated >
< /QuoteInfo >
< /StockQuote >
< /StockQuoteResponse >
```

c.

Figure 6.4: SOAP message responses to the requests getStockQuote(MSFT), getStockQuote(UAL), and getStockQuote(HMC)

new accumulating Huffman tree encoding scheme and the adaptive structure of the compact packet.

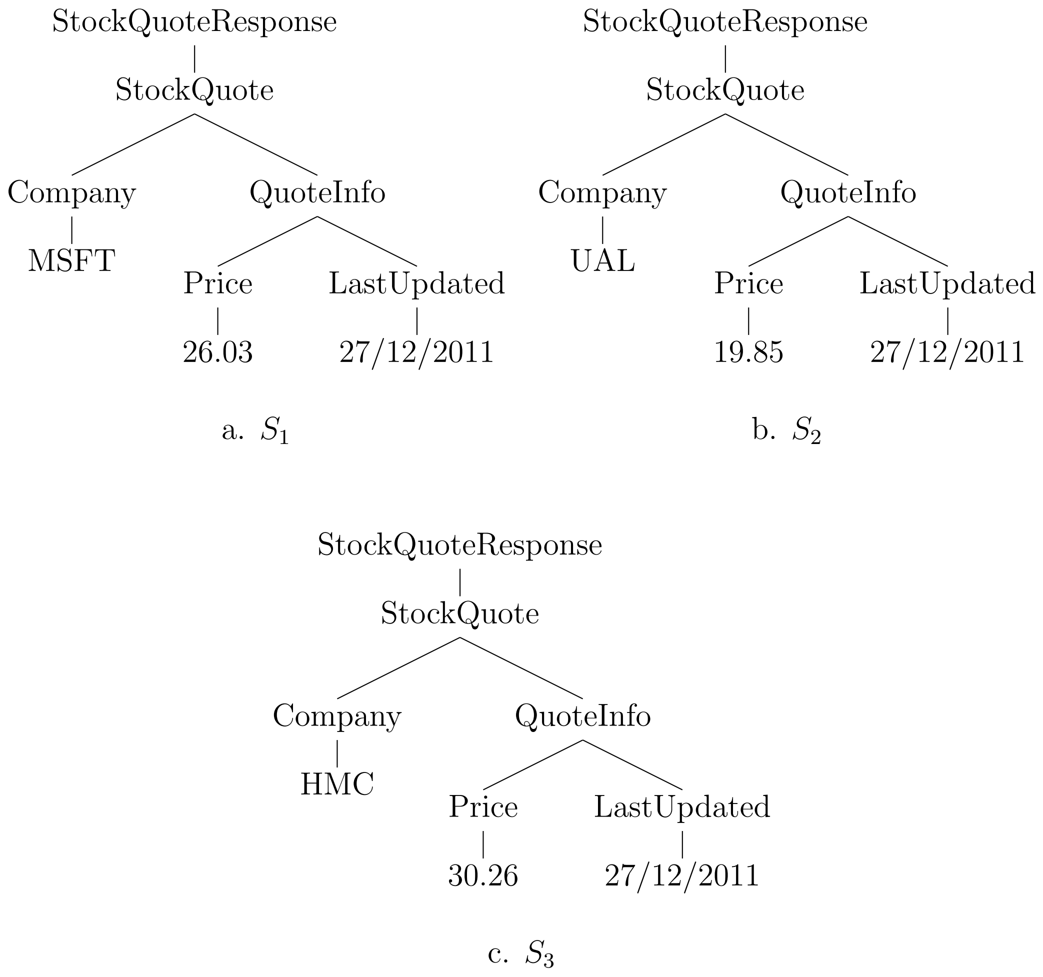


Figure 6.5: XML message trees of SOAP messages (S_1, S_2, and S_3)

6.3.1 Accumulating Huffman tree encoding

Regenerating the XML Huffman binary tree for the aggregated messages is a significantly time consuming function, as it requires computing the redundancy of all the XML items included in the lookup table. Moreover, as the compact packet has only the binary encoded forms of the aggregated messages, it adds extra high computations for searching the XML items using their mapping binary codes and not the textual contents.

Traditional Huffman binary tree encoding creates unique binary codes for the encoded

Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	StockQuote	0
2	Company	1
3	QuoteInfo	1
4	MSFT	2
5	Price	3
6	LastUpdated	3
7	26.03	5
8	27/12/2011	6

a.

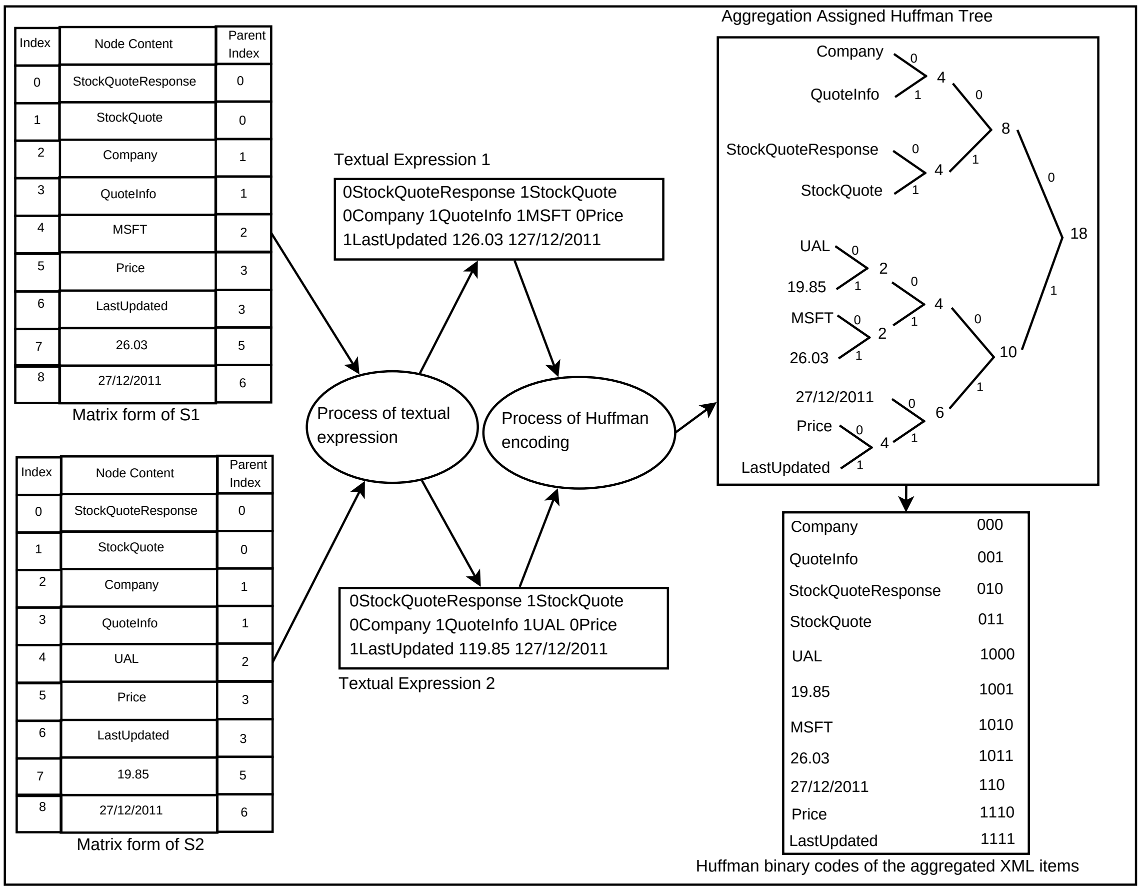
Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	StockQuote	0
2	Company	1
3	QuoteInfo	1
4	UAL	2
5	Price	3
6	LastUpdated	3
7	19.85	5
8	27/12/2011	6

b.

Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	StockQuote	0
2	Company	1
3	QuoteInfo	1
4	HMC	2
5	Price	3
6	LastUpdated	3
7	30.26	5
8	27/12/2011	6

c.

Figure 6.6: Generated matrix form of SOAP messages (S_1, S_2, and S_3)



Matrix form of S1

Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	StockQuote	0
2	Company	1
3	QuoteInfo	1
4	MSFT	2
5	Price	3
6	LastUpdated	3
7	26.03	5
8	27/12/2011	6

Matrix form of S2

Index	Node Content	Parent Index
0	StockQuoteResponse	0
1	StockQuote	0
2	Company	1
3	QuoteInfo	1
4	UAL	2
5	Price	3
6	LastUpdated	3
7	19.85	5
8	27/12/2011	6

Textual Expression 1

0StockQuoteResponse 1StockQuote 0Company 1QuoteInfo 1MSFT 0Price 1LastUpdated 126.03 127/12/2011

Textual Expression 2

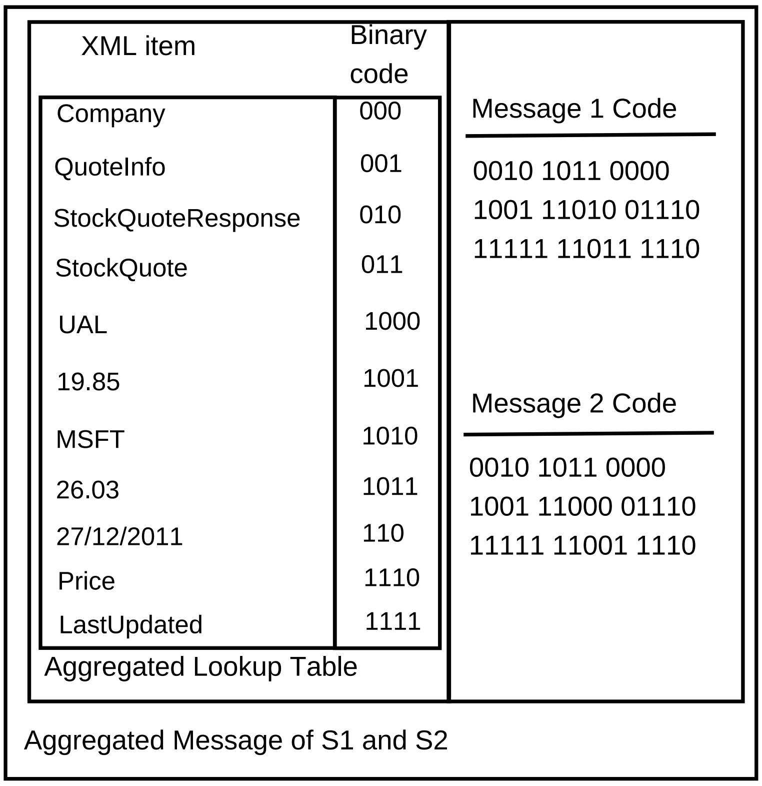
0StockQuoteResponse 1StockQuote 0Company 1QuoteInfo 1UAL 0Price 1LastUpdated 119.85 127/12/2011

Huffman binary codes of the aggregated XML items

Company	000
QuoteInfo	001
StockQuoteResponse	010
StockQuote	011
UAL	1000
19.85	1001
MSFT	1010
26.03	1011
27/12/2011	110
Price	1110
LastUpdated	1111

Figure 6.7: Computing Huffman tree as a part of the aggregation process of SOAP messages (S_1, and S_2)

items that do not share any binary combinations. This fact is a result of the general strategy of the Huffman technique and the structure of the generated binary tree as all the encoded items are located at the bottom level of the tree. Furthermore, every single Huffman generated binary code leads to a unique path over the Huffman binary tree. In this chapter, Huffman binary trees have been developed to share the shortest path of the core Huffman tree as the initial path to the newly added items to the same tree. In other words, the smallest binary combination in the tree represents the first sub combination for all newly added XML items. The structure of the new sub-tree is based on the same iterative processing of Huffman encoding. Another small lookup table that has the newly added XML items



XML item	Binary code
Company	000
QuoteInfo	001
StockQuoteResponse	010
StockQuote	011
UAL	1000
19.85	1001
MSFT	1010
26.03	1011
27/12/2011	110
Price	1110
LastUpdated	1111

Aggregated Lookup Table

Message 1 Code

0010 1011 0000
1001 11010 01110
11111 11011 1110

Message 2 Code

0010 1011 0000
1001 11000 01110
11111 11001 1110

Aggregated Message of S1 and S2

Figure 6.8: Aggregated message of S_1, and S_2

with their mapping binary codes in addition to the shared binary code (Accumulating RBC (Root Binary Code)) is created and added to the accumulating aggregation compact packet. Figure 6.10 illustrates the processing of the accumulating aggregation of SOAP message S_3 to the core aggregated packet of SOAP messages S_1 and S_2. Figure 6.11 shows the structure of the packet for the distributed aggregation of SOAP messages.

6.3.2 Optimized aggregation with binary search

As mentioned before, every new accumulating XML item would be first checked if it appears in the core aggregation lookup table. For large numbers of messages, selection search would be a very expensive function that would highly affect the overall processing time of the aggregation model. Therefore, the proposed model has been optimized by replacing the selection

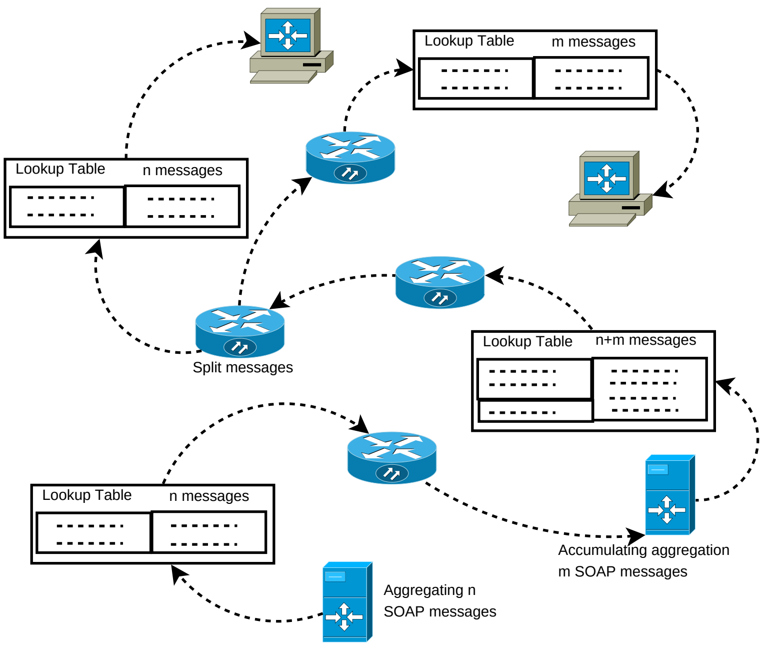


Figure 6.9: Scenario for distributed aggregation of SOAP messages over the network

search with the binary search in order to minimize the processing time. Furthermore, all the newly added XML items have been sorted using optimized bubble sort because the binary search only works with sorted lists of items.

6.3.3 Adaptive compact aggregation structure

The new development of the Huffman binary tree encoding by having shared sub combination binary codes would create technical conflicts during the de-aggregation process to extract the original SOAP messages. Technically, new accumulating items starting with the same sub combination binary code would result in the same extracted item by following the binary path over the Huffman tree. With the aim to avoid these conflicts, a few modifications have been included to the final structure of the accumulating aggregated compact packet. One extra binary bit (0 or 1) has been added to every single encoded message in order to recognize if this message has an accumulating XML items or not. Assignment bit with “0” value means no accumulating items are included and assignment bit with “1” value means accumulating

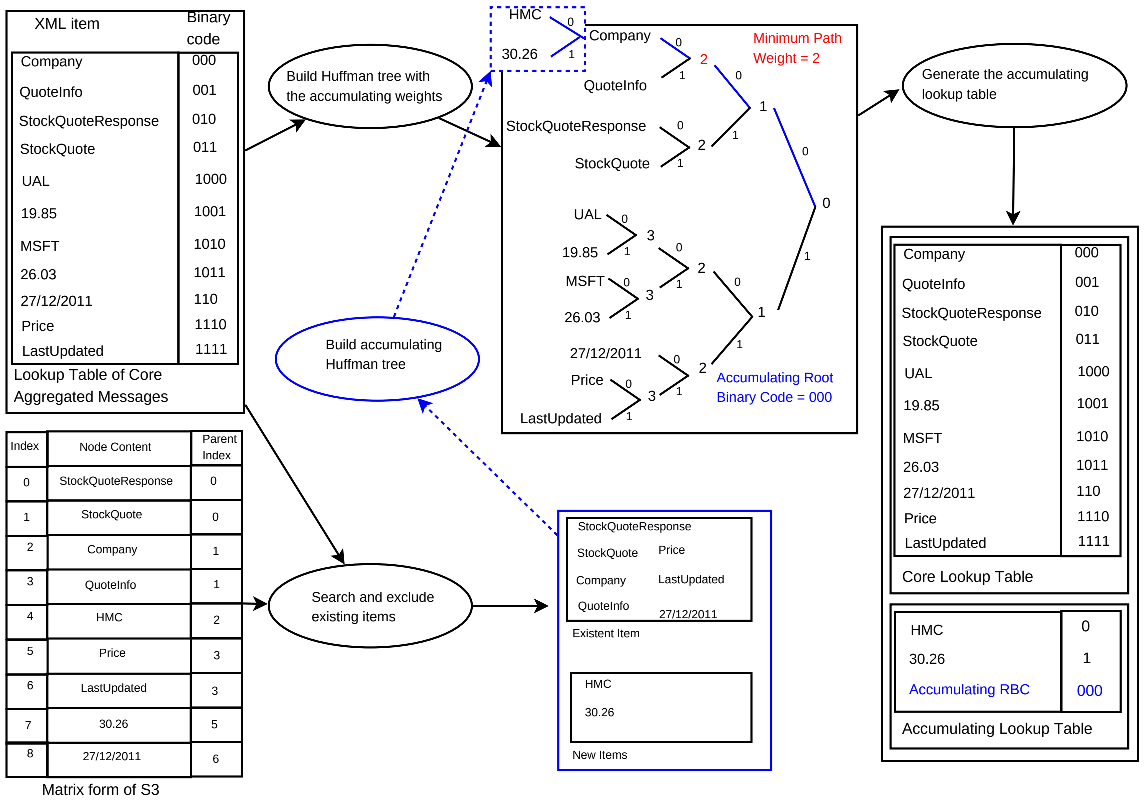


Figure 6.10: Generation of the accumulating lookup table to accumulate message S_3 to the core aggregated packet of messages (S_1, and S_2)

items are included. Messages with no accumulating items are only using the core lookup table for decoding the binary codes. On the other hand, messages with accumulating items, another extra binary bit has been injected after the shared accumulating RBC (0 or 1) to recognize the correct lookup table (core or accumulating) for decoding the binary codes. Binary bit with "0" refers to the core lookup table and with "1" refers to accumulating lookup table (see Fig. 6.11).

6.4 Fast fractals for SOAP clustering

This section illustrates the proposed fast fractal clustering model showing the mathematically derived fractal factors used to reduce the search time. Moreover, it describes the generation of

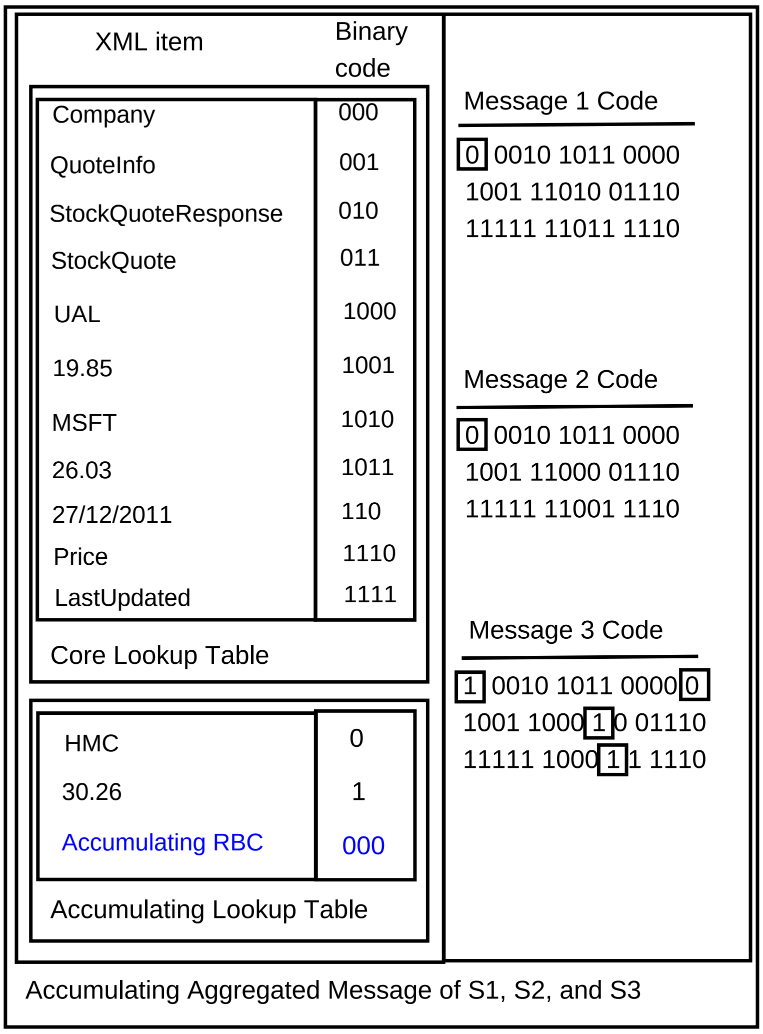


Figure 6.11: Accumulating aggregated message of S_1, S_2, and S_3

the dataset and the overall clustering strategy. Fractal similarity measurement is a potential unsupervised dynamic clustering model that can support aggregation scenarios of SOAP messages with the aim to achieve significant SOAP message size reduction [t, 1994]. However, fractal mathematical models are time-consuming techniques, this drawback represents the bottleneck of fractal applications [Hart, 1996; Kumar Bisoi and Mishra, 1999]. Therefore, new mathematical models are developed that aim to speed up the performance of regular fractals. We introduce new mathematical factors that are computed in advance of the fractal coefficients computation in order to segment data object blocks based on their fractal factors. Then, fractal coefficients (scale, offset, and RMS) are computed for only blocks that have the same fractal factor value. The proposed models first start by generating the dataset of the SOAP messages that have a set of numeric vectors showing the local and global loads of XML items. Then, fast fractal factors are computed in advance, then we compute the fractal coefficients, and finally cluster the XML messages based on the maximum histograms of fast fractal indexes in every single message.

6.4.1 Dataset generation

The generation of dataset is the first step of XML document preparations by transforming them into numerical vectors for each XML document. The same technique that has been proposed in chapter 5 is used to generate the dataset for the fast fractal clustering technique. XML tags and data values are the physical descriptive features of the XML documents. Hence, generating the dataset starts with computing the general vector of all the distinctive tags and data values and consider it as the vector template for all XML documents $V = \{X_1, X_2, ..., X_n\}$. The position of the distinctive XML tags and data values in the vector template is considered as a search key of clustering documents as it shows the impact of the considered feature in the counterpart XML messages. In the dataset, every single XML document has a vector that carry the frequencies of the XML features counted in the vector template. Term Frequency with Inverse Document Frequency (TF-IDF) technique is used to compute the dataset vectors by finding the frequencies of all the XML tags and data values for the XML documents.

6.4.2 Fractal similarity measurement model

Fractal is a mathematical approach that can approximate regular and irregular objects based on the ability to find the self similarity of smaller objects of the same overall object. Fractal self-similarity feature is the fundamental aspect exploited by applications such as digital image compression, and in audio and image pattern recognition. In this chapter, fractal is proposed as a clustering technique to classify XML messages by utilizing its self-similarity measurements.

The origin of the fractal self-similarity is the transformation of the object fragments (block) as objects are made up from smaller copies of themselves. Moreover, every single copy is made up again of smaller copies of itself. For a given object say P, fractal is systematically partitioning that object into smaller fragments (blocks) which are called range blocks (r_i). Furthermore, range blocks can be recreated (approximately) from other similar fragments (blocks) usually called domain blocks d_i from the same object using the partitioning function f_i. Equation 6.1 is to compute the approximated range blocks ($\acute{R}$).

$$\acute{R} = S \times D + O \tag{6.1}$$

The strategy to find the best match of range blocks is by computing the fractal Root Mean Square Error (RMS) and to select the domain block with the smallest RMS. In this technique, when a matching process of a range block is active, the rest of the dataset blocks are considered as domain blocks. Moreover, all vectors are divided up into same sized blocks. The block size in this work is based on the overall size of the vector length as five block sizes are considered: 10%, 20%, 25%, 50%, and 100% of the vector length. For every single block, the best match is computed and assigned to the dataset vector of that best match. Then, data objects are classified according to the maximum histograms of their blocks best matches.

6.4.3 Fast fractal similarity measurement model

Although fractal models are considered to be time consuming approaches [Tao et al., 2000], fractal can provide significant contributions in many fields like remote sensing applications [Hart, 1996]. In this chapter, new fractal mathematical derivations are introduced in order to speed up the clustering task of the XML documents. The basic strategy of the proposed derivations is to find out new common fractal factors that can be computed separately on both sides: range and domain blocks of the search area. The total number of the required computations for the matching process is reduced by segmenting all the range and domain blocks separately into a number of segments based on the values of these fractal factors. In this technique, the matching computations do not require checking the similarity of a range block with all the domain blocks, the range blocks are checked and compared with only the domain blocks that have the same fractal index (factor) value.

Fractal mathematical derivations

First, Eq. 6.1 can be formed in a different representation as:

$$\widehat{R} = S \times \widehat{D} + O \tag{6.2}$$

where $\widehat{R}$ is the average value of R range block and $\widehat{D}$ is the average value of D domain block. Then, the same equation can be formed into another representation:

$$R_i = S \times D_i + O \tag{6.3}$$

where R_i represents one cell in the R range block and D_i represents one cell in the D domain block. The next step is to consider three range block parameters α, β, and χ as:

$$\alpha = \sum_{i=1}^{i=n} \left| R_i - \widehat{R} \right| \tag{6.4}$$

$$\beta = \sum_{i=1}^{i=n} \left(R_i - \widehat{R} \right)^2 \tag{6.5}$$

and

$$\chi = \sum_{i=1}^{i=n} \left(R_i - \widehat{R}\right)^4 \tag{6.6}$$

By substituting equations 6.2 and 6.3 in Eq. 6.4, Eq. 6.7 is derived to represent the fractal parameter α in a different form.

$$\alpha = |S| \times \sum_{i=1}^{i=n} \left|D_i - \widehat{D}\right| \tag{6.7}$$

where n is the length of range blocks (R) and domain blocks (D). Now, by substituting equations 6.2 and 6.3 in Eq. 6.5, we obtain Eq. 6.8 that represents the fractal parameter β in a different representation.

$$\beta = S^2 \times \sum_{i=1}^{i=n} \left(D_i - \widehat{D}\right)^2 \tag{6.8}$$

Furthermore, by substituting equations 6.2 and 6.3 in Eq. 6.6, we obtain Eq. 6.9 that represents the fractal parameter χ in a different form as well.

$$\chi = S^4 \times \sum_{i=1}^{i=n} \left(D_i - \widehat{D}\right)^4 \tag{6.9}$$

- **First fast fractal factor (F1)**

  Equation 6.10 represents the general form of the fast fractal factor $F1$:

$$F1 = \frac{\beta^2}{\chi} \tag{6.10}$$

  By substituting equations 6.5 and 6.6 in 6.10, we obtain the first range fast fractal factor ($F1$) as stated in Eq. 6.11.

$$F1 = \frac{\left(\sum_{i=1}^{i=n}\left(R_i - \widehat{R}\right)^2\right)^2}{\sum_{i=1}^{i=n}\left(R_i - \widehat{R}\right)^4} \tag{6.11}$$

Then, by substituting equations 6.8 and 6.9 in 6.10, we obtain the first domain fast fractal factor ($F1$) as stated in Eq. 6.12.

$$F1 = \frac{\left(\sum_{i=1}^{i=n}\left(D_i - \widehat{D}\right)^2\right)^2}{\sum_{i=1}^{i=n}\left(D_i - \widehat{D}\right)^4} \tag{6.12}$$

- **Second fast fractal factor (F2)**

Equation 6.13 represents the general form of the fast fractal factor $F2$:

$$F2 = \frac{\alpha^2 \times \beta}{\chi} \tag{6.13}$$

By substituting equations 6.4, 6.5, and 6.6 in Eq. 6.13, we obtain the second range fast fractal factor ($F2$) as stated in Eq. 6.14.

$$F2 = \frac{\left(\sum_{i=1}^{i=n}\left|R_i - \widehat{R}\right|\right)^2 \times \sum_{i=1}^{i=n}\left(R_i - \widehat{R}\right)^2}{\sum_{i=1}^{i=n}\left(R_i - \widehat{R}\right)^4} \tag{6.14}$$

Then, by substituting equations 6.7, 6.8, and 6.9 in Eq. 6.13, we obtain the second domain fast fractal factor ($F2$) as stated in Eq. 6.15.

$$F2 = \frac{\left(\sum_{i=1}^{i=n}\left|D_i - \widehat{D}\right|\right)^2 \times \sum_{i=1}^{i=n}\left(D_i - \widehat{D}\right)^2}{\sum_{i=1}^{i=n}\left(D_i - \widehat{D}\right)^4} \tag{6.15}$$

6.4.4 Fast fractal based clustering strategy

The strategy of the fast fractal clustering model is to compute the fast fractal factors in advance of the fractal matching process and separately for both range and domain fractal blocks. Then, the fractal computations are computed with only range and domain blocks that have the same fractal factor value. The processing time for clustering SOAP messages is potentially minimized by avoiding the fractal computations for range and domain blocks that are different and select only the domain blocks that share the fast fractal factors value.

6.5 Experimental analysis

Both distributed aggregation and fast fractal clustering have been evaluated and compared with their non-optimized original models with respect to processing time (aggregation and clustering time) and aggregated messages size reduction (compressed size and compression ratio).

6.5.1 Distributed aggregation

In order to evaluate the performance of the distributed aggregation model, a testbed of 1800 real SOAP messages is created. These messages have been split into two main groups, the first represents the group of messages for accumulating aggregation, which includes 600 messages. The second group (1200 messages) is for the core aggregated messages that represent the base for adding the newly coming messages from network nodes over the path of these messages by accumulating aggregation. The distributed aggregation model has been evaluated for both the ability to reduce the total aggregated messages size and the aggregation time. Furthermore, both aggregations (core and accumulating) have been implemented with a varying number of messages starting with only 50 messages of each and gradually increasing in increments of 50 messages each time.

Figure 6.12 shows the ability of both classical (standalone) and distributed aggregation models in minimizing the total size of aggregated SOAP messages. Both models have been implemented to aggregate 12 groups of messages with different sizes starting from 50 messages up to 600 messages. Furthermore, the distributed aggregation model has been evaluated using a wide range of core aggregated messages that start from 50 messages up to 1200 messages, increasing the core aggregated packet by 50 messages each time. The distributed aggregation has significantly outperformed the standalone aggregation technique by adding more than 50% of the performance of the standalone aggregation technique.

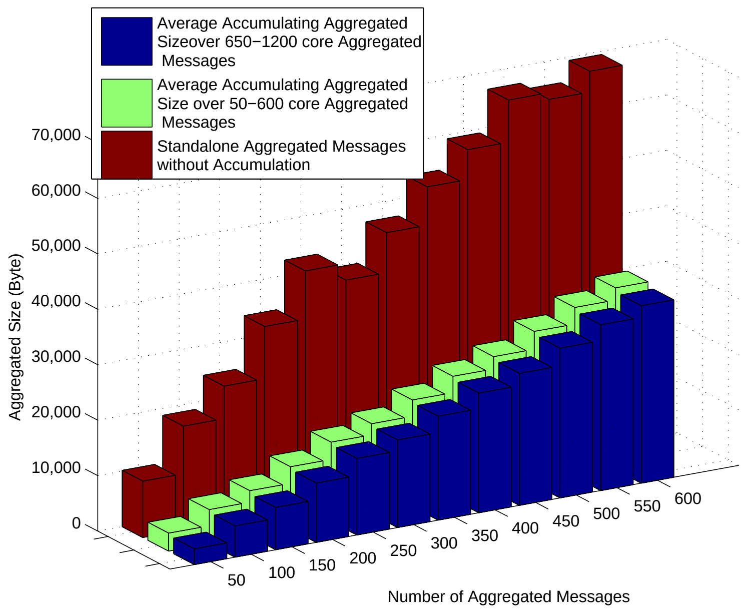


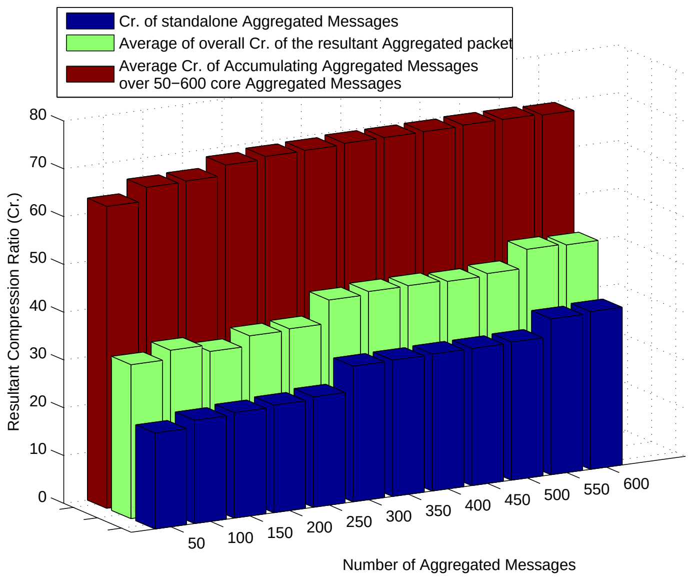
Figure 6.12: Compressed size of both standalone and aggregated (accumulated) SOAP messages

The compression ratio has been investigated for three different cases: standalone aggregation, resultant accumulation for distributed aggregated messages, and the overall compression ratio for the resultant packet size of both core and accumulating messages. Figure 6.13 shows the compression ratios of standalone and distributed aggregation for 12 groups of SOAP messages with sizes that starting from 50 up to 600 messages. Accumulating SOAP messages have been aggregated a) over 12 groups of core messages that starts from 50 up to 600 messages and b) over the same number of groups but with sizes that starting from 650 up to 1200 messages. It is clear that the distributed aggregation model has resulted in higher compression ratios than the standalone aggregation model. Moreover, results have shown the impact of the size of core messages on achieving higher compression ratio as the accumulating aggregation over 650-1200 core messages has resulted in better compression ratios than over 50-600 core messages.

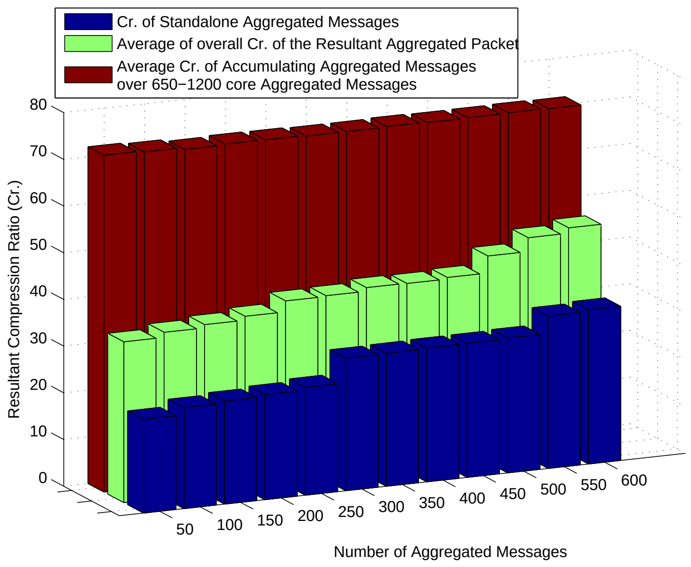
Furthermore, the processing time for the distributed aggregation model and the standalone aggregation model have been compared. Figure 6.14 shows the average aggregation time for aggregating SOAP messages over 50-600 and 650-1200 core aggregated messages with both approaches for both including and excluding the optimized bubble sort and binary search. The optimized accumulating aggregation (bubble sort and binary search) outperformed the non-optimized accumulating aggregation significantly with respect to the processing time, as it only requires approximately 50% of the non-optimized approach.

6.5.2 Fast fractal clustering

With the aim of investigating the performance of the proposed fast fractal clustering technique, the evaluation work in this chapter has been achieved using SOAP messages varying between 140 bytes and 53 Kbytes. Furthermore, a large sized testbed of real SOAP messages has been considered as it has 6000 messages that are distributed into four groups based on their size range (small, medium, large, and very large) with 1500 messages in each group. Moreover, the compression-based aggregation model of SOAP Web messages (proposed in chapter 4) is deployed as a tool to investigate the proposed clustering model in computing the similarity based clustered groups. It is found to be more efficient than other standard clustering models. The compression-based aggregation tool is used to aggregate the resultant fractal based clusters in order to evaluate the compression ratios achieved as a result of the

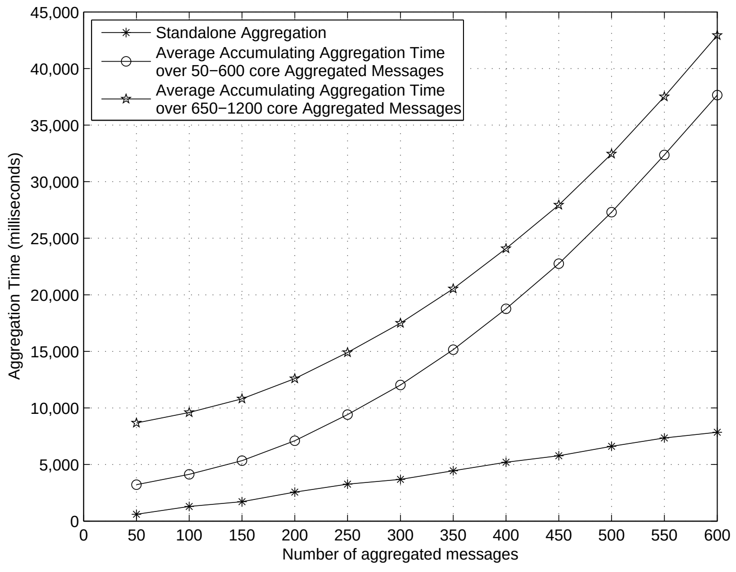


a.

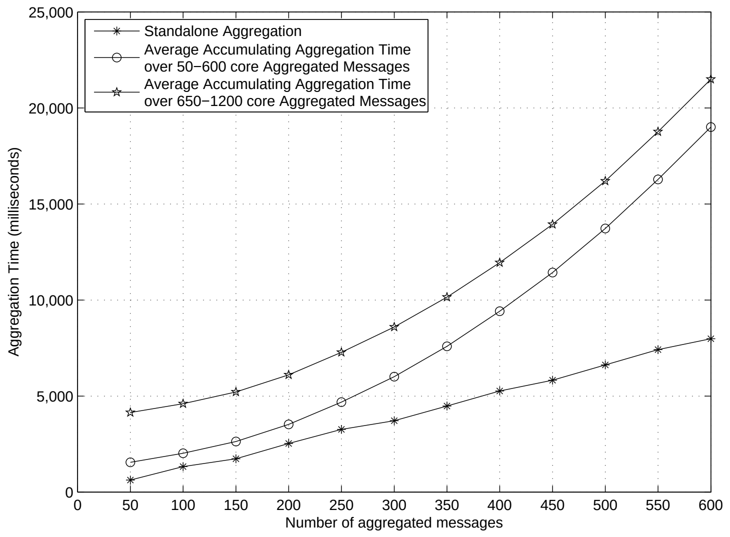


b.

Figure 6.13: Resultant compression ratios (Cr) for both standalone and aggregated (accumulated) SOAP messages over a. 50-600 core aggregated messages and b. 650-1200 core aggregated messages



a.



b.

Figure 6.14: Processing time for a. standalone and non-optimized accumulating aggregation (based on selection search) of SOAP messages and b. standalone and optimized accumulating aggregation (including optimized bubble sort and binary search) of SOAP messages

high fractal similarity of SOAP messages in each cluster.

The processing time required to generate the XML message dataset has been investigated on multi-sized datasets (varying number of messages per dataset) starting from only 50 XML messages up to 1500 XML messages. Figure 6.15 shows the consumed processing time for dataset generation for small, medium, large, and very large XML documents. It is clear that for smaller messages, less processing time is needed. Dataset generation for small messages does not need more than 1200 milliseconds for 1500 messages and not more than a few milliseconds for small sized dataset with 50 messages.

Fractal block size plays a vital role in reducing the processing time to compute the fractal similarity of XML messages. Five fractal block sizes (10%, 20%, 25%, 50%, and 100%) are investigated as they have clearly proven the impact of block size on the clustering time. The enhanced performance fractal clustering techniques ($F1$ and $F2$) have tremendously shown significant reductions in the clustering time, in comparison to the regular fractal clustering model. Figures 6.16, 6.17, 6.18, 6.19, and 6.20 show the significant efficiency of the proposed fast fractal technique with fractal block sizes of 10%, 20%, 25%, 50%, and 100% respectively. Breaking up the dataset vectors into smaller blocks leads to a higher fractal similarity of XML messages. However, small fractal block size based clustering requires more fractal computations as it increases the number of blocks in each dataset vector and therefore increases the number of fractal parameters and factors that need to be computed. As a result, small block sizes cost more in processing time. Figures 6.16 and 6.20 show the average clustering time of highly different sized datsets using 10% and 100% fractal block sizes, which are the smallest and largest block size percentage respectively that have been developed in the proposed fractal techniques. Fast fractal clustering techniques ($F1$ and $F2$) have tremendously minimized the required processing time as they require less than 2500 milliseconds for the 10% block size based model and only less than 100 milliseconds for the 100% block size based model.

Table 6.1: Average compression ratios (Cr.) of the aggregated messages using fractal, fast fractal (F1), and fast fractal (F2) clustering techniques with different fractal block sizes

		Average Compression Ratio					
		Fractal Technique		**Fast Fractal (F1)**		**Fast Fractal (F2)**	
Message Group	**Fractal Block Size**	**Fixed-length**	**Huffman**	**Fixed-length**	**Huffman**	**Fixed-length**	**Huffman**
Small							
	10%	3.96244	3.846	3.92281	3.80754	3.803941	3.69216
	20%	3.92242	3.80676	3.8832	3.768688	3.765517	3.654485
	25%	3.88239	3.78713	3.84357	3.749262	3.727094	3.635648
	50%	3.84236	3.76751	3.80394	3.729835	3.68867	3.61681
	100%	3.83436	3.74789	3.79602	3.710409	3.680985	3.597972
Medium							
	10%	7.20238	7.8259	7.13035	7.747612	6.914281	7.512836
	20%	7.12962	7.74602	7.05833	7.668554	6.844439	7.436174
	25%	7.05687	7.70609	6.9863	7.629026	6.774598	7.397843
	50%	6.98412	7.66616	6.91428	7.589497	6.704757	7.359512
	100%	6.96957	7.62623	6.89988	7.549969	6.690789	7.321182
Large							
	10%	12.96978	16.30117	12.84008	16.13816	12.45099	15.64913
	20%	12.83877	16.13484	12.71038	15.97349	12.32522	15.48944
	25%	12.70776	16.05167	12.58068	15.89115	12.19945	15.4096
	50%	12.57675	15.9685	12.45099	15.80881	12.07368	15.32976
	100%	12.55055	15.88533	12.42505	15.72648	12.04853	15.24992
V.Large							
	10%	15.18099	21.2661	15.02918	21.05344	14.57375	20.41545
	20%	15.02765	21.0491	14.87737	20.83861	14.42654	20.20713
	25%	14.8743	20.9406	14.72556	20.73119	14.27933	20.10297
	50%	14.72096	20.8321	14.57375	20.62378	14.13212	19.99881
	100%	14.69029	20.7236	14.54339	20.51636	14.10268	19.89465

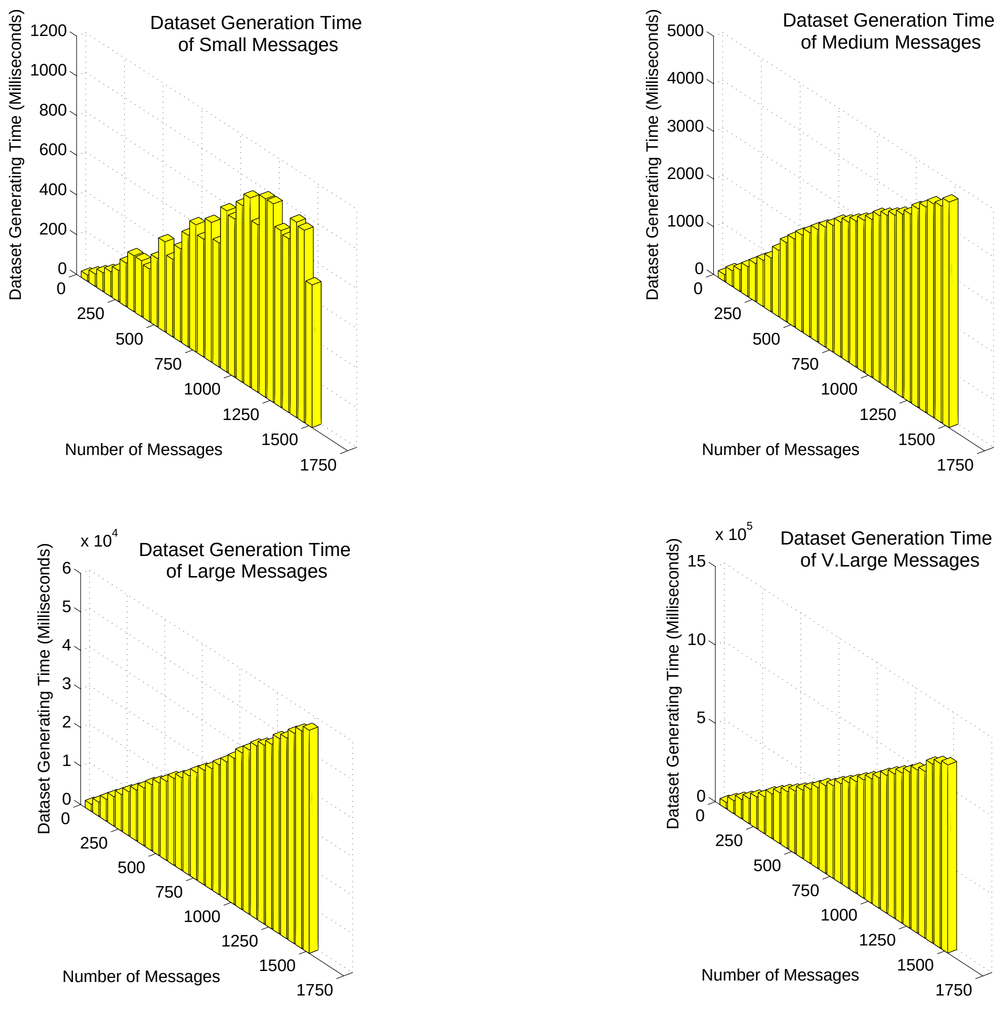


Figure 6.15: The processing time (milliseconds) for generating the data set of small, medium, large, and very large messages

Furthermore, all of the proposed fractal clustering techniques have been compared with Principle Component Analysis (PCA) combined with K-means in addition to the standalone K-means as two standard clustering models in terms of the processing time (see Fig. 6.21). Fractal clustering technique is competitive with other models when it is clustering up to 50 messages as it costs less time than both PCA + K-means and K-means. However, it costs significantly more time than other models when the dataset size is greater than 50 messages.

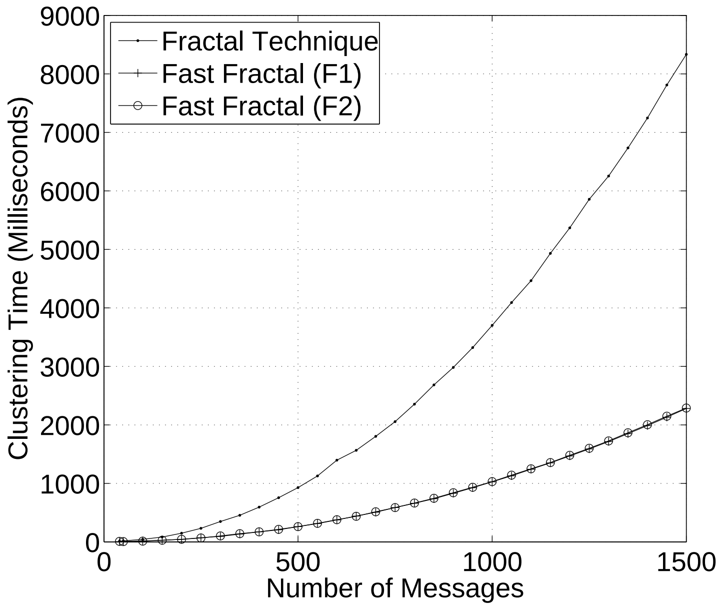


Figure 6.16: The average processing time (milliseconds) of 10% fractal block size based clustering technique

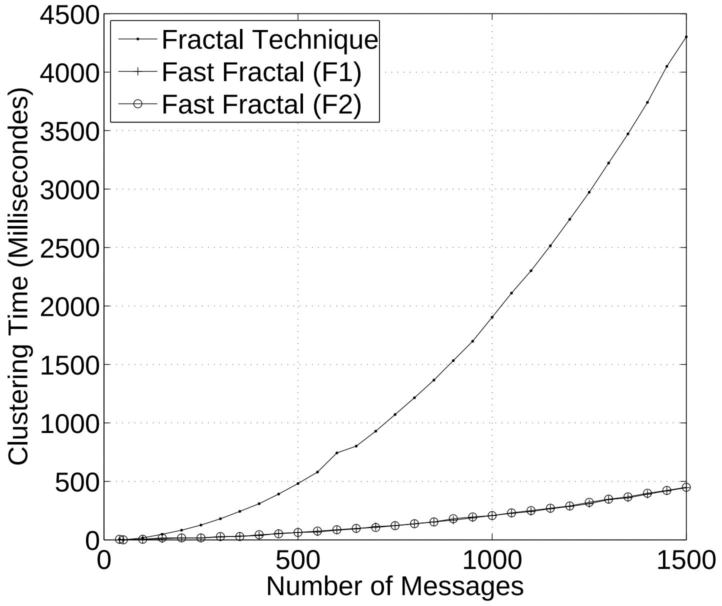


Figure 6.17: The average processing time (milliseconds) of 20% fractal block size based clustering technique

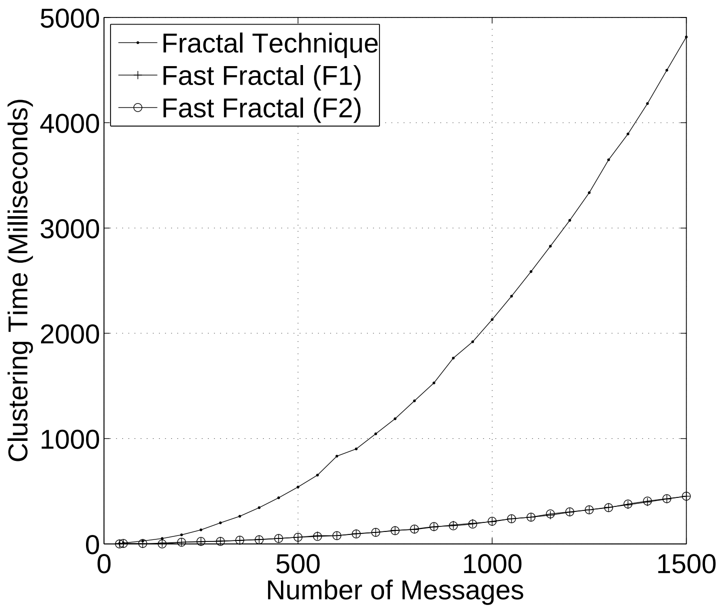


Figure 6.18: The average processing time (milliseconds) of 25% fractal block size based clustering technique

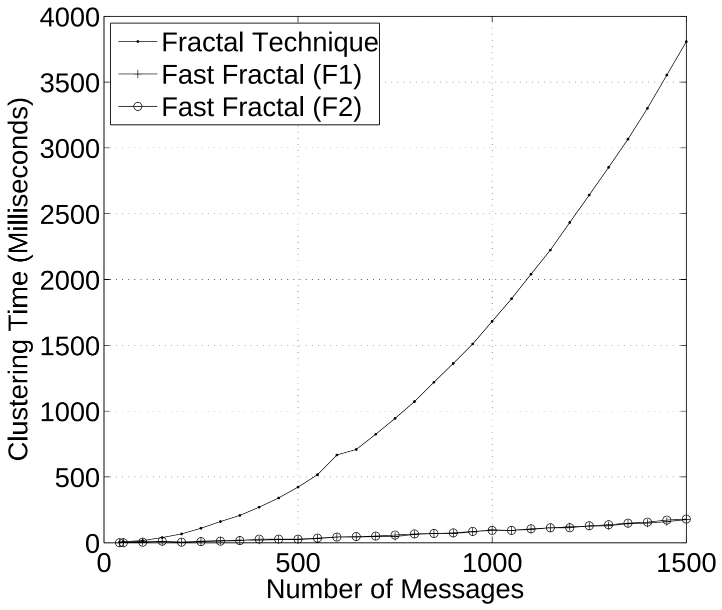


Figure 6.19: The average processing time (milliseconds) of 50% fractal block size based clustering technique

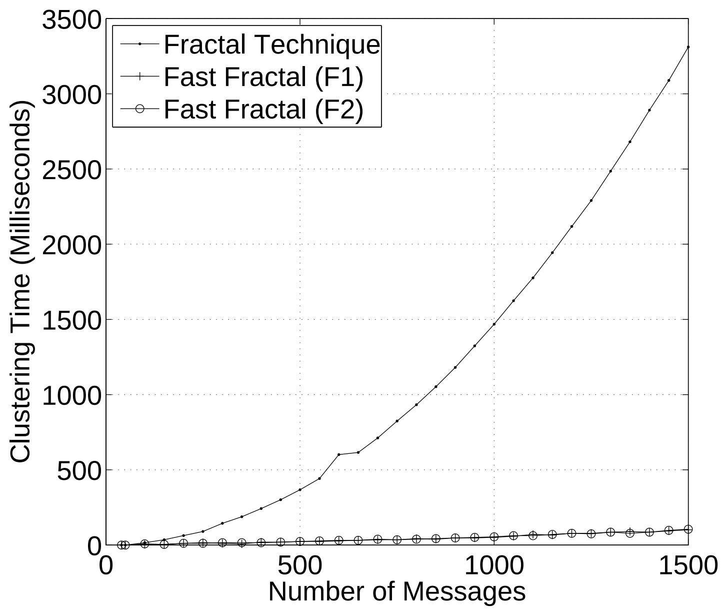


Figure 6.20: The average processing time (milliseconds) of 100% fractal block size based clustering technique

On the other hand, both fast fractal techniques ($F1$ and $F2$) have outperformed all other techniques that are involved in the evaluation. They both have shown similar efficiencies to each other and they have sharply reduced the required processing time in comparison with the classical fractal technique.

The ability of the proposed fractal clustering techniques in supporting the compression based aggregation model to achieve high network traffic reduction is investigated. Moreover, the proposed techniques are compared with other models (PCA+K-means and K-means). Table 6.1 shows the resultant compression ratios that can be achieved by getting the support of fractal clustering to aggregate the most similar XML messages. It is clear that fast fractal factor $F1$ has slightly outperformed fast fractal factor $F2$. Both fast fractal factors $F1$ and $F2$ have shown more than 98% efficiency of the classical fractal technique in achieving a high compression ratio for the aggregated messages. Furthermore, it is obvious that as long as the fractal block size is small, a higher similarity and compression ratio can be achieved.

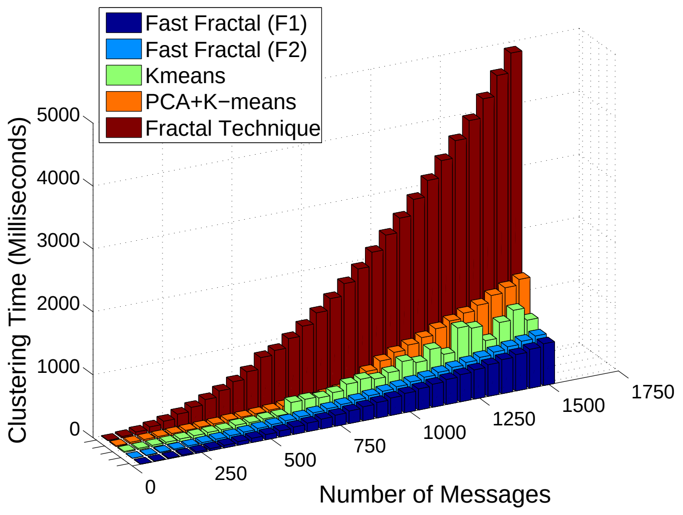


Figure 6.21: Average clustering time (milliseconds) of variant sizes of datasets of Web messages using fractal technique, PCA combined with K-means, K-means, fast fractal (F1), and fast fractal (F2) clusterings

Table 6.2: Average compression ratios (Cr.) of the aggregated XML messages based on K-means, PCA combined with K-means, fractal, fast fractal (F1), and fast fractal (F2) clusters

Messages Group	**Small**		**Medium**	
Encoding Technique	**Fixed-length**	**Huffman**	**Fixed-length**	**Huffman**
K-means	3.57769	3.544639	6.503032	7.204943
PCA + K-means	3.616578	3.525683	6.573717	7.174087
Fractal Technique	3.888793	3.791057	7.068513	7.714072
Fast Fractal (F1)	3.849905	3.753147	6.997828	7.636931
Fast Fractal (F2)	3.733241	3.639415	6.785773	7.405509
Messages Group	**Large**		**V.Large**	
Encoding Technique	**Fixed-length**	**Huffman**	**Fixed-length**	**Huffman**
K-means	11.71042	15.1042	13.88572	19.28531
PCA + K-means	11.83771	14.94352	13.85592	19.49494
Fractal Technique	12.72872	16.0683	14.89884	20.9623
Fast Fractal (F1)	12.60144	15.90762	14.74985	20.75267
Fast Fractal (F2)	12.21957	15.42557	14.30289	20.1238

Table 6.2 shows the average compression ratios of the aggregated messages based on fractal, K-means, PCA+K-means, fast fractal ($F1$), and fast fractal ($F2$) XML message clusters.

6.6 Conclusion

Distributed aggregation for SOAP messages is a new potential that can reduce the network traffic significantly, better than the regular aggregation models. It supports aggregating messages from several resources taking advantage of having higher redundancy with other accumulating messages. Furthermore, fractal similarity of SOAP messages is a potential clustering model that can efficiently cluster SOAP messages based on their high fractal similarity degree. The proposed fast fractal development has increased the performance of the fractal clustering model tremendously. The results are highly promising as the processing time has been significantly reduced in comparison to the classical fractal clustering technique. Moreover, fast fractal clustering models have outperformed K-means and PCA combined with K-means in terms of both the clustering time and the SOAP messages size reduction ability. The experimental results have shown the significance of fractal support in the aggregation scenarios by minimizing both the required processing time and bandwidth to send and receive messages over the network.

Chapter 7

Conclusion

This chapter is organized as follows. Section 7.1 summarizes the aims of this research, including the main research questions. In section 7.2, main contributions made by this thesis are presented. Furthermore, section 7.3 describes the future directions specific to the thesis. Final remarks are stated in section 7.4.

7.1 Research aims

This thesis aims to generally improve the performance of SOAP Web services. Latency of Web services has been investigated and analyzed, focusing on the fact that Web services generate larger-sized messages than the actual payloads. Furthermore, we were particularly motivated by the specific problem of creating high network traffic by Web services as a result of their verbose nature. Regarding this problem, several recent significant studies have concentrated on proposing standalone compression techniques and textual aggregation models for pairs of SOAP messages. In light of these strategies, we focused our efforts on the modeling and developing of novel and efficient compression and aggregation techniques for groups of messages that can potentially reduce network traffic.

Despite the fact that Web services are built based on the tree structure, few studies have developed tree structure-based models with the aim to reduce network traffic. The first two research questions in this thesis focused on devising more effective models to minimize the high network traffic created by SOAP Web services. Regarding these research questions, this thesis particularly aims to provide new efficient tree structure-based compression and aggregation models.

XML similarity measurements and clustering represent an important issue for improving the performance of Web services. The second two research questions in this thesis focused on the development of potential similarity-based clustering models, as well as devising efficient collaboration among multi-Web servers for novel distributed aggregation models for SOAP messages. Moreover, the fractal self-similarity principle is investigated for the first time with SOAP messages to provide a new efficient fractal clustering model.

Specifically, the following four research questions have been addressed in this thesis:

1. **How to reduce the size of SOAP messages sent/received over the network?**
2. **How to utilize the compression techniques (redundancy-aware) in developing SOAP aggregation models?**
3. **What are the cost-effective similarity parameters for clustering SOAP messages?**
4. **What are the cost-effective methods to develop a distributed Web server-based aggregation model for large numbers (hundreds and thousands) of SOAP messages?**

7.2 Research contributions

In response to the four research questions originally posed in section 1.2, the following four main contributions were made:

1. **Binary and general tree-based structure compression techniques**
   Three compression techniques are proposed with the development of both binary and general tree transformations of XML trees. The proposed transformations (binary and general tree) are technical supportive techniques to the compression operation by removing the duplicated XML tags. A binary-bit assignment process is developed in order to generate new textual expressions for the XML tree using binary and general tree traversals, such as breadth-first and depth-first traversals. Both fixed length and variable length (Huffman) are used to encode the textual expressions of XML documents. Traditionally, fixed length and Huffman techniques are developed to encode individual symbols or characters. In this research, XML tags and data values of SOAP messages are treated overall as individual parameters for both fixed length and Huffman encoding. Furthermore, the proposed compression techniques are evaluated with extensive experiments, and promising results are obtained in potentially reducing the SOAP messages size. Moreover, the resultant performance is compared with other standard compressors, such as gzip, bzip2, XMill, and XbMill, and found to be significantly better. In addition, compression and decompression times are tested using PCs, laptops, netbooks, and PDAs.

2. **Redundancy-aware aggregation and group-based similarity measurements**
   Redundancy aware aggregation models are proposed using the compression potential by exploiting the shared redundancy in SOAP messages. The new aggregation models aim to effectively minimize SOAP messages and aggregate them in one compact packet. Furthermore, network traffic has been reduced potentially through enabling the SOAP Web servers to aggregate SOAP responses with a high degree of similarity, and send them back as one compact-sized packet. SOAP responses can be extracted from the aggregated packet at the closest node to the recipient. Moreover, similarity measurements represent an important factor in achieving significant size reduction for

the aggregated messages by selecting highly similar messages for aggregation as they share large portions of redundant information. Compressibility, Jaccard coefficient, and VSM are proposed as group-based similarity measurements that cluster SOAP messages based on their textual similarities. All the proposed aggregation models and similarity measurements are evaluated as they have proven the capability of potentially reducing message size.

3. **Dynamic fractal similarity-based clustering model**
   A novel unsupervised dynamic fractal self-similarity based clustering model is proposed for SOAP messages. Technically, fractal mathematical parameters (scale and offset) are introduced as similarity metrics for SOAP messages that can express their similarities based on their numeric forms. TF-IDF technique is proposed to generate the numeric form of the considered SOAP messages focusing on their redundancies. The generated dataset is a set of numeric vectors (one vector to every SOAP message) expressing the local loads of SOAP messages. Dataset vectors are organized into a set of columns dividing all vectors into equal-sized blocks. Technically, fractal clustering represents an efficient alternative to the traditional textual similarity by computing the fractal parameters for all blocks in every single vector. Blocks located on the same column are matched with each other by computing the minimum RMS using the scale and offset for each block. Vectors are then clustered according to the fractal similarities of their blocks. The proposed clustering model is evaluated and compared with well-known clustering techniques, such as K-means and PCA combined with K-means. The results have proven the high performance of the fractal clustering in comparison with other techniques, in terms of the ability to support aggregation models in achieving higher compression ratios. Moreover, the proposed model requires less processing time than other techniques for a dataset set the size of 40 SOAP messages or less.

4. **Distributed aggregation and fast fractal similarity measurements**
   A multi-Web server based distributed aggregation model is proposed that can aggregate SOAP responses from several Web servers that are located on the same network delivery path of the aggregated packet. New developed Huffman encoding is proposed to add the encoding of more SOAP responses at other Web servers without decoding the aggregated messages. Furthermore, an adaptive structure is developed for the compact packet to suit the inclusion of new SOAP responses. On the other hand, the fractal mathematical model is modified by deriving new fractal factors with the aim to reduce the search task (fractal matching process). This has been accomplished by segmenting the vector blocks based on their similarities using the new fractal factors. Only blocks placed in the same segment are matched using fractal coefficients (scale, offset, and RMS). This development has led to a potential improvement required in cluster times compared with the classical fractal clustering model. Experimental results for large numbers of messages (hundreds and thousands) have shown a significant improvement in performance compared with K-means and PCA combined with K-means.

7.3 Limitations and future work

In this thesis, several advances were made regarding the performance of SOAP Web services. However, there is still critical work that needs to be analyzed and implemented in the area of Web services performance, and especially in reference to reducing the high network traffic created by SOAP Web services.

- In Chapters 3 and 4, several compression and aggregation models are proposed. Binary tree and general tree structures are three transformations developed to generate new reduced-size textual expressions for SOAP messages. Then, fixed length and Huffman binary tree techniques are used to encode the generated textual expressions. The limitation of these models is that they treat both XML tags and data values as the same technical weight. Data values are encoded as individual items. Generally, SOAP messages have small actual payloads and heavy loads of XML tags. Therefore, the proposed models have proven their potential in reducing SOAP message size. However, we expect the performance of the proposed models to degrade with messages having a heavy actual payload with small loads of XML tags. For example, Shakespeare's plays are represented in SOAP messages.

- Generic compressors, such as gzip and bzip2, are more efficient in compressing text. Therefore, a new combination of the XML-aware techniques proposed in this thesis with other generic compressors would be a significant future direction. Furthermore, heuristic methods are interesting suggestion to have an advance decision for involving the generic compressors or not by analyzing the considered SOAP messages before the encoding process.

- In Chapters 5 and 6, the fractal self-similarity principle is used to propose new efficient clustering techniques for SOAP messages using their numeric forms. TF-IDF weights are developed to transform the textual representation of SOAP messages into a numeric representation expressing the local redundancies in every single message. Since fractal clustering models are developed to cluster numeric objects, it is an interesting future direction to investigate their performance on other Web service contexts, such as the JSON Web service. Regarding this suggestion, it is required to develop a numeric transformation for the textual representation of the considered Web services.

- The proposed work in this thesis has been extensively evaluated and proven that potential improvement to the SOAP Web services can be achieved. Technically, several scenarios, such as Cloud Web services and Inter-Cloud, in addition to the bandwidth-constrained environments (i.e. wireless Web services), may significantly benefit from the proposed work. However, network simulations and real experiments are still needed to validate this fact.

7.4 Final remarks

This thesis has made several advances in the area of Web services. The large size of SOAP messages has been significantly minimized by the three tree structure-based compression techniques proposed in this thesis. Furthermore, the problem of having high network traffic created by the verbose nature of SOAP Web services was analyzed, and a potential solution was provided by aggregating a group of SOAP responses with the ability of utilizing the redundancy exploitation in compression techniques. Similarity measurements were shown to play a vital role in strengthening aggregation models by having a high degree of similarity in the considered SOAP messages. Both textual and numeric similarity-based clustering models are developed. Lastly, a multi-Web server distributed aggregation model is proposed with the aim to consolidate Web servers and sharply reduce network traffic.

Appendix A

WSDL Schema

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        targetNamespace="http://schemas.xmlsoap.org/wsdl/"
        elementFormDefault="qualified">
   <element name="documentation">
      <complexType mixed="true">
         <choice minOccurs="0" maxOccurs="unbounded">
            <any minOccurs="0" maxOccurs="unbounded"/>
         </choice>
         <anyAttribute/>
      </complexType>
   </element>
   <complexType name="documented" abstract="true">
      <sequence>
         <element ref="wsdl:documentation" minOccurs="0"/>
      </sequence>
```

```
</complexType>
<complexType name="openAtts" abstract="true">
   <annotation>
      <documentation>
      This type is extended by  component types
      to allow attributes from other namespaces to be added.
      </documentation>
   </annotation>
   <sequence>
      <element ref="wsdl:documentation" minOccurs="0"/>
   </sequence>
   <anyAttribute namespace="##other"/>
</complexType>
<element name="definitions" type="wsdl:definitionsType">
   <key name="message">
      <selector xpath="message"/>
      <field xpath="@name"/>
   </key>
   <key name="portType">
      <selector xpath="portType"/>
      <field xpath="@name"/>
   </key>
   <key name="binding">
      <selector xpath="binding"/>
      <field xpath="@name"/>
   </key>
```

```
    <key name="service">
      <selector xpath="service"/>
      <field xpath="@name"/>
    </key>
    <key name="import">
        <selector xpath="import"/>
        <field xpath="@namespace"/>
      </key>
    <key name="port">
      <selector xpath="service/port"/>
      <field xpath="@name"/>
    </key>
  </element>
  <complexType name="definitionsType">
    <complexContent>
      <extension base="wsdl:documented">
        <sequence>
          <element ref="wsdl:import" minOccurs="0" maxOccurs="unbounded"/>
          <element ref="wsdl:types" minOccurs="0"/>
          <element ref="wsdl:message" minOccurs="0" maxOccurs="unbounded"/>
          <element ref="wsdl:portType" minOccurs="0" maxOccurs="unbounded"/>
          <element ref="wsdl:binding" minOccurs="0" maxOccurs="unbounded"/>
          <element ref="wsdl:service" minOccurs="0" maxOccurs="unbounded"/>
          <any namespace="##other" minOccurs="0" maxOccurs="unbounded">
            <annotation>
              <documentation>to support extensibility elements
```

```
                                                      </documentation>
                  </annotation>
               </any>
            </sequence>
            <attribute name="targetNamespace" type="uriReference" use="optional"/>
            <attribute name="name" type="NMTOKEN" use="optional"/>
         </extension>
      </complexContent>
  </complexType>
   <element name="import" type="wsdl:importType"/>
   <complexType name="importType">
      <complexContent>
   <extension base="wsdl:documented">
   <attribute name="namespace" type="uriReference" use="required"/>
      <attribute name="location" type="uriReference" use="required"/>
   </extension>
  </complexContent>
  </complexType>
   <element name="types" type="wsdl:typesType"/>
   <complexType name="typesType">  <complexContent>
   <extension base="wsdl:documented">
   <sequence>
   <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
   </extension>
  </complexContent>
```

```
</complexType>
 <element name="message" type="wsdl:messageType">
    <unique name="part">
       <selector xpath="part"/>
       <field xpath="@name"/>
    </unique>
 </element>
 <complexType name="messageType">
    <complexContent>
 <extension base="wsdl:documented">
 <sequence>
 <element ref="wsdl:part" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
    <attribute name="name" type="NCName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <element name="part" type="wsdl:partType"/>
 <complexType name="partType">
    <complexContent>
 <extension base="wsdl:openAtts">
 <attribute name="name" type="NMTOKEN" use="optional"/>
    <attribute name="type" type="QName" use="optional"/>
    <attribute name="element" type="QName" use="optional"/>
 </extension>
</complexContent>
```

```
</complexType>
 <element name="portType" type="wsdl:portTypeType"/>
 <complexType name="portTypeType">
    <complexContent>
 <extension base="wsdl:documented">
 <sequence>
 <element ref="wsdl:operation" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
    <attribute name="name" type="NCName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <element name="operation" type="wsdl:operationType"/>
 <complexType name="operationType">
    <complexContent>
 <extension base="wsdl:documented">
    <choice>
       <group ref="wsdl:one-way-operation"/>
       <group ref="wsdl:request-response-operation"/>
       <group ref="wsdl:solicit-response-operation"/>
       <group ref="wsdl:notification-operation"/>
    </choice>
    <attribute name="name" type="NCName" use="required"/>
 </extension>
</complexContent>
</complexType>
```

```
<group name="one-way-operation">
   <sequence>
      <element ref="wsdl:input"/>
   </sequence>
</group>
<group name="request-response-operation">
   <sequence>
      <element ref="wsdl:input"/>
      <element ref="wsdl:output"/>
      <element ref="wsdl:fault" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
</group>
<group name="solicit-response-operation">
   <sequence>
      <element ref="wsdl:output"/>
      <element ref="wsdl:input"/>
      <element ref="wsdl:fault" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
</group>
<group name="notification-operation">
   <sequence>
      <element ref="wsdl:output"/>
   </sequence>
</group>
<element name="input" type="wsdl:paramType"/>
<element name="output" type="wsdl:paramType"/>
```

```
 <element name="fault" type="wsdl:faultType"/>
 <complexType name="paramType">
    <complexContent>
 <extension base="wsdl:documented">
 <attribute name="name" type="NMTOKEN" use="optional"/>
    <attribute name="message" type="QName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <complexType name="faultType">
    <complexContent>
 <extension base="wsdl:documented">
 <attribute name="name" type="NMTOKEN" use="required"/>
    <attribute name="message" type="QName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <complexType name="startWithExtensionsType" abstract="true">
    <complexContent>
 <extension base="wsdl:documented">
 <sequence>
 <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
 </extension>
</complexContent>
</complexType>
```

```
   <element name="binding" type="wsdl:bindingType"/>
   <complexType name="bindingType">
      <complexContent>
   <extension base="wsdl:startWithExtensionsType">
   <sequence>
   <element name="operation" type="wsdl:binding_operationType"
                       minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="type" type="QName" use="required"/>
   </extension>
  </complexContent>
  </complexType>
   <complexType name="binding_operationType">
      <complexContent>
   <extension base="wsdl:startWithExtensionsType">
   <sequence>
   <element name="input" type="wsdl:startWithExtensionsType" minOccurs="0"/>
      <element name="output" type="wsdl:startWithExtensionsType" minOccurs="0"/>
      <element name="fault" minOccurs="0" maxOccurs="unbounded">
         <complexType>
            <complexContent>
   <extension base="wsdl:startWithExtensionsType">
   <attribute name="name" type="NMTOKEN" use="required"/>
         </extension>
  </complexContent>
```

```
</complexType>
    </element>
</sequence>
    <attribute name="name" type="NCName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <element name="service" type="wsdl:serviceType"/>
 <complexType name="serviceType">
    <complexContent>
 <extension base="wsdl:documented">
 <sequence>
 <element ref="wsdl:port" minOccurs="0" maxOccurs="unbounded"/>
    <any namespace="##other" minOccurs="0"/>
</sequence>
    <attribute name="name" type="NCName" use="required"/>
 </extension>
</complexContent>
</complexType>
 <element name="port" type="wsdl:portType"/>
 <complexType name="portType">
    <complexContent>
 <extension base="wsdl:documented">
 <sequence>
 <any namespace="##other" minOccurs="0"/>
</sequence>
```

```
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="binding" type="QName" use="required"/>
   </extension>
  </complexContent>
  </complexType>
  <attribute name="arrayType" type="string"/>
</schema>
```

Appendix B

SOAP Binding Schema

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        targetNamespace="http://schemas.xmlsoap.org/wsdl/soap/">
   <element name="binding" type="soap:bindingType"/>
   <complexType name="bindingType">
      <attribute name="transport" type="uriReference" use="optional"/>
      <attribute name="style" type="soap:styleChoice" use="optional"/>
   </complexType>
   <simpleType name="styleChoice">
      <restriction base="string">
   <enumeration value="rpc"/>
      <enumeration value="document"/>
  </restriction>
   </simpleType>
   <element name="operation" type="soap:operationType"/>
   <complexType name="operationType">
```

```
      <attribute name="soapAction" type="uriReference" use="optional"/>
      <attribute name="style" type="soap:styleChoice" use="optional"/>
   </complexType>
   <element name="body" type="soap:bodyType"/>
   <complexType name="bodyType">
      <attribute name="encodingStyle" type="uriReference" use="optional"/>
      <attribute name="parts" type="NMTOKENS" use="optional"/>
      <attribute name="use" type="soap:useChoice" use="optional"/>
      <attribute name="namespace" type="uriReference" use="optional"/>
   </complexType>
   <simpleType name="useChoice">
      <restriction base="string">
   <enumeration value="literal"/>
      <enumeration value="encoded"/>
  </restriction>
   </simpleType>
   <element name="fault" type="soap:faultType"/>
   <complexType name="faultType">
      <complexContent>
   <restriction base="soap:bodyType">
   <attribute name="parts" type="NMTOKENS" use="prohibited"/>
   </restriction>
  </complexContent>
  </complexType>
   <element name="header" type="soap:headerType"/>
   <complexType name="headerType">
```

```
      <all>
          <element ref="soap:headerfault">
      </all>
      <attribute name="message" type="QName" use="required"/>
      <attribute name="parts" type="NMTOKENS" use="required"/>
      <attribute name="use" type="soap:useChoice" use="required"/>
      <attribute name="encodingStyle" type="uriReference" use="optional"/>
      <attribute name="namespace" type="uriReference" use="optional"/>
   </complexType>
   <element name="headerfault" type="soap:headerfaultType"/>
   <complexType name="headerfaultType">
      <attribute name="message" type="QName" use="required"/>
      <attribute name="parts" type="NMTOKENS" use="required"/>
      <attribute name="use" type="soap:useChoice" use="required"/>
      <attribute name="encodingStyle" type="uriReference" use="optional"/>
      <attribute name="namespace" type="uriReference" use="optional"/>
   </complexType>
   <element name="address" type="soap:addressType"/>
   <complexType name="addressType">
      <attribute name="location" type="uriReference" use="required"/>
   </complexType>
</schema>
```

Bibliography

N. Abu-Ghazaleh and M. Lewis. Differential deserialization for optimized soap performance. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, page 21, nov. 2005. doi: 10.1109/SC.2005.24.

N. Abu-Ghazaleh, M. Lewis, and M. Govindaraju. Differential serialization for optimized soap performance. In *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*, pages 55 – 64, june 2004. doi: 10.1109/HPDC.2004.1323489.

S. AjayKumar, C. Nachiappan, K. Periyakaruppan, and P. Boominathan. Enhancing portable environment using cloud and grid. In *2009 International Conference on Signal Processing Systems*, pages 728 –732, may 2009. doi: 10.1109/ICSPS.2009.106.

D. Andresen. Enhancing cluster application performance via smarter scheduling and stronger soap. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, page 8 pp., april 2005. doi: 10.1109/IPDPS.2005.202.

D. Andresen, D. Sexton, K. Devaram, and V. Ranganath. Lye: a high-performance caching soap implementation. In *Parallel Processing, 2004. ICPP 2004. International Conference on*, pages 143 – 150 vol.1, aug. 2004. doi: 10.1109/ICPP.2004.1327914.

Z. Baharav. Fractal arrays based on iterated functions system (ifs). In *Antennas and Prop-*

agation Society International Symposium, 1999. IEEE, volume 4, pages 2686 –2689 vol.4, aug 1999. doi: 10.1109/APS.1999.789361.

P. Brebner and A. Liu. *Performance and Cost Assessment of Cloud Services*, volume 6568 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011. URL http://dx.doi.org/10.1007/978-3-642-19394-1_5. 10.1007/978-3-642-19394-1_5.

V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli. Enhancing a web-server cluster with quality of service mechanisms. In *Performance, Computing, and Communications Conference, 2002. 21st IEEE International*, pages 205 –212, 2002. doi: 10.1109/IPCCC.2002.995152.

M. Chen and Y. Song. Summarization of text clustering based vector space model. In *Computer-Aided Industrial Design Conceptual Design, 2009. CAID CD 2009. IEEE 10th International Conference on*, pages 2362 –2365, nov. 2009. doi: 10.1109/CAIDCD.2009.5375265.

J. Y. Chung, B. Park, Y. Won, J. Strassner, and J. Hong. An effective similarity metric for application traffic classification. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 286 –292, april 2010. doi: 10.1109/NOMS.2010.5488477.

X. Cui, T. Potok, and P. Palathingal. Document clustering using particle swarm optimization. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 185 – 191, june 2005. doi: 10.1109/SIS.2005.1501621.

E. Damiani and S. Marrara. Efficient soap message exchange and evaluation through xml similarity. *In Proceedings of the 2008 ACM Workshop on Secure Web Services. Alexandria, Virginia, USA*, pages 29–36, October 31 - 31, 2008.

A. Davis and D. Zhang. A comparative study of dcom and soap. In *Multimedia Software*

Engineering, 2002. Proceedings. Fourth International Symposium on, pages 48 – 55, 2002. doi: 10.1109/MMSE.2002.1181595.

D. Davis and M. Parashar. Latency performance of soap implementations. In *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, page 407, may 2002.

K. Devaram and D. Andresen. Soap optimization via client-side caching. *In Proceedings of the International Conference on Web Services. Las Vegas, Nevada, USA*, pages 520–, June 23 - 26, 2003.

M. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud computing: Distributed internet computing for it and scientific research. *Internet Computing, IEEE*, 13(5):10 –13, sept.-oct. 2009. ISSN 1089-7801. doi: 10.1109/MIC.2009.103.

R. Elfwing, U. Paulsson, and L. Lundberg. Performance of soap in web service environment compared to corba. In *Software Engineering Conference, 2002. Ninth Asia-Pacific*, pages 84 – 93, 2002. doi: 10.1109/APSEC.2002.1182978.

M. Ericsson. The effects of xml compression on soap performance. *World Wide Web*, 10:279–307, 2007. ISSN 1386-145X. URL http://dx.doi.org/10.1007/s11280-007-0032-y. 10.1007/s11280-007-0032-y.

S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Fast detection of xml structural similarity. *Knowledge and Data Engineering, IEEE Transactions on*, 17(2):160 – 175, feb. 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.27.

V. Gamini Abhaya, Z. Tari, and P. Bertok. Building web services middleware with predictable service execution. In L. Chen, P. Triantafillou, and T. Suel, editors, *Web Information Systems Engineering WISE 2010*, volume 6488 of *Lecture Notes in Computer*

Science, pages 23–37. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-17615-9. URL http://dx.doi.org/10.1007/978-3-642-17616-6_5. 10.1007/978-3-642-17616-6_5.

M. Gudgin, N. Mendelsohn, M. Nottingham, and H. Ruellan. Soap message transmission optimization mechanism. *World Wide Web Consortium (W3C)*, January 25, 2005a. URL http://www.w3.org/TR/soap12-mtom/.

M. Gudgin, N. Mendelsohn, M. Nottingham, and H. Ruellan. Xml-binary optimized packaging. *World Wide Web Consortium (W3C)*, January 25, 2005b. URL http://www.w3.org/TR/xop10/.

J. Hart. Fractal image compression and recurrent iterated function systems. *Computer Graphics and Applications, IEEE*, 16(4):25 –33, jul 1996. ISSN 0272-1716. doi: 10.1109/38.511849.

S. Heinzl, M. Mathes, T. Friese, M. Smith, and B. Freisleben. Flex-swa: Flexible exchange of binary data based on soap messages with attachments. In *Web Services, 2006. ICWS '06. International Conference on*, pages 3 –10, sept. 2006. doi: 10.1109/ICWS.2006.65.

J. Helander and S. Sigurdsson. Self-tuning planned actions time to make real-time soap real. In *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, pages 80 – 89, may 2005. doi: 10.1109/ISORC.2005.51.

J. H. Hwang and M. S. Gu. Clustering xml documents based on the weight of frequent structures. pages 845 –849, nov. 2007. doi: 10.1109/ICCIT.2007.101.

L. Johnsrud, D. Hadzic, T. Hafsoe, F. Johnsen, and K. Lund. Efficient web services in mobile networks. In *on Web Services, 2008. ECOWS '08. IEEE Sixth European Conference*, pages 197 –204, nov. 2008. doi: 10.1109/ECOWS.2008.24.

R. H. S. Julio Cezar Estrella, Marcos Jose Santana and F. J. Monaco. Real-time compression of soap messages in a soa environment. *In Proceedings of the 26th Annual ACM International Conference on Design of Communication. Lisbon, Portugal*, pages 163–168, September 22-24, 2008.

W. Jun, H. Lei, and N. Chunlei. Speed-up soap processing by data mapping template. In *Proceedings of the 2006 international workshop on Service-oriented software engineering*, SOSE '06, pages 40–46, New York, NY, USA, 2006. ACM. ISBN 1-59593-398-0. doi: http://doi.acm.org/10.1145/1138486.1138495. URL http://doi.acm.org/10.1145/1138486.1138495.

A. Kumar Bisoi and J. Mishra. Enhancing the beauty of fractals. In *Computational Intelligence and Multimedia Applications, 1999. ICCIMA '99. Proceedings. Third International Conference on*, pages 454 –458, 1999. doi: 10.1109/ICCIMA.1999.798573.

H.-p. Leung, F.-l. Chung, and S. C.-f. Chan. On the use of hierarchical information in sequential mining-based xml document similarity computation. *Knowledge and Information Systems*, 7:476–498, 2005. ISSN 0219-1377. URL http://dx.doi.org/10.1007/s10115-004-0156-7. 10.1007/s10115-004-0156-7.

R. Levy, J. Nagarajarao, G. Pacifici, A. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. In *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on*, pages 247 – 261, march 2003. doi: 10.1109/INM.2003.1194184.

W. Lian, D.-l. Cheung, N. Mamoulis, and S.-M. Yiu. An efficient and scalable algorithm for clustering xml documents by structure. *Knowledge and Data Engineering, IEEE Transactions on*, 16(1):82 – 96, jan. 2004. ISSN 1041-4347. doi: 10.1109/TKDE.2004.1264824.

H. Liefke and D. Suciu. An extensible compressor for xml data. *SIGMOD Rec.*, 29:57–62, March 2000a. ISSN 0163-5808. doi: http://doi.acm.org.ezproxy.lib.rmit.edu.au/10.1145/344788.344815. URL http://doi.acm.org.ezproxy.lib.rmit.edu.au/10.1145/344788.344815.

H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. *SIGMOD Rec.*, 29: 153–164, May 2000b. ISSN 0163-5808. doi: http://doi.acm.org.ezproxy.lib.rmit.edu.au/10.1145/335191.335405. URL http://doi.acm.org.ezproxy.lib.rmit.edu.au/10.1145/335191.335405.

H. Liu, H. Bao, J. Wang, and D. Xu. A novel vector space model for tree based concept similarity measurement. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*, pages 144 –148, april 2010. doi: 10.1109/ICIME.2010.5477749.

J. Liu, J. Wang, W. Hsu, and K. Herbert. Xml clustering by principal component analysis. pages 658 – 662, nov. 2004. doi: 10.1109/ICTAI.2004.122.

X. Liu and R. Deters. An efficient dual caching strategy for web service enabled pdas. *In Proceedings of the 22nd Annual ACM symposium on Applied computing. Seoul, Korea*, pages 788–794, 2007.

W. Lu, K. Chiu, and D. Gannon. Building a generic soap framework over binary xml. In *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 195 –204, 0-0 2006. doi: 10.1109/HPDC.2006.1652150.

O. Luoma and J. Teuhola. Predictive modeling in xml compression. In *Digital Information Management, 2007. ICDIM '07. 2nd International Conference on*, volume 2, pages 565 –570, oct. 2007. doi: 10.1109/ICDIM.2007.4444283.

K. N. M. Nakagawa and S. Shimojo. Web-based distributed simulation and data management services for medical applications. *In Proceedings of the 19th IEEE International*

Symposium on Computer-Based Medical Systems, 2006. CBMS 2006., page pp. 125 130, 2006.

Y. Ma and R. Chbeir. Content and structure based approach for xml similarity. In *Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference on*, pages 136 –140, sept. 2005. doi: 10.1109/CIT.2005.91.

E. Mancini, M. Rak, and U. Villano. Perfcloud: Grid services for performance-oriented development of cloud computing applications. In *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE '09. 18th IEEE International Workshops on*, pages 201 –206, 29 2009-july 1 2009. doi: 10.1109/WETICE.2009.47.

T. Miyamoto, M. Hayashi, and H. Tanaka. Customizing network functions for high performance cloud computing. In *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, pages 130 –133, july 2009. doi: 10.1109/NCA.2009.59.

N. Nagwani and A. Bhansali. Clustering homogeneous xml documents using weighted similarities on xml attributes. In *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pages 369 –372, feb. 2010. doi: 10.1109/IADCC.2010.5422926.

S. Oh and G. C. Fox. Hhfr: A new architecture for mobile web services: Principles and implementations. *Technical report, Indiana University Community Grids Laboratory, Bloomington, IN, USA*, 2005. URL http://grids.ucs.indiana.edu/ptliupages/publications/HHFR_ohsangy.pdf.

K. A. Phan, Z. Tari, and P. Bertok. Similarity-based soap multicast protocol to reduce bandwith and latency in web services. *Services Computing, IEEE Transactions on*, 1(2): 88 –103, april-june 2008. ISSN 1939-1374. doi: 10.1109/TSC.2008.8.

M.-C. Rosu. A-soap: Adaptive soap message processing and compression. In *Web Services,* *2007. ICWS 2007. IEEE International Conference on*, pages 200 –207, july 2007. doi: 10.1109/ICWS.2007.29.

C. A. Shaffer. *A Practical Introduction to Data Structures and Algorithm Analysis.* Alan APT, USA, 1st edition, 1997.

P. Skibinski, S. Grabowski, and J. Swacha. Fast transform for effective xml compression. In *CAD Systems in Microelectronics, 2007. CADSM '07. 9th International Conference - The Experience of Designing and Applications of*, pages 323 –326, feb. 2007. doi: 10.1109/CADSM.2007.4297567.

V. Stantchev. Performance evaluation of cloud computing offerings. In *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP '09. Third International Conference on*, pages 187 –192, oct. 2009. doi: 10.1109/ADVCOMP.2009.36.

T. Suzumura, T. Takase, and M. Tatsubori. Optimizing web services performance by differential deserialization. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages 185 – 192 vol.1, july 2005. doi: 10.1109/ICWS.2005.87.

P. R. M. t. *Fractal Functions, Fractal Surfaces, and Wavelets.* Academic Press, Inc., San Diego, California, 1994.

T. Takase, H. MIYASHITA, T. Suzumura, and M. Tatsubori. An adaptive, fast, and safe xml parser based on byte sequences memorization. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 692–701, New York, NY, USA, 2005. ACM. ISBN 1-59593-046-9. doi: http://doi.acm.org/10.1145/1060745.1060845. URL http://doi.acm.org/10.1145/1060745.1060845.

Y. Tao, W. Lam, and Y. Tung. Extraction of fractal feature for pattern recognition. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 527 –530 vol.2, 2000. doi: 10.1109/ICPR.2000.906128.

J. Tekli, R. Chbeir, and K. Yetongnon. A hybrid approach for xml similarity. In J. van Leeuwen, G. Italiano, W. van der Hoek, C. Meinel, H. Sack, and F. Plil, editors, *SOFSEM 2007: Theory and Practice of Computer Science*, volume 4362 of *Lecture Notes in Computer Science*, pages 783–795. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-69506-6. URL http://dx.doi.org/10.1007/978-3-540-69507-3_68. 10.1007/978-3-540-69507-3_68.

D. D. Terry and V. Ramasubramanian. Caching xml web services for mobility. *Queue, New York, NY, USA*, 1:70–78, 2003. ISSN 1542-7730. doi: http://doi.acm.org/10.1145/846057.864024. URL http://doi.acm.org/10.1145/846057.864024.

M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Freie. A concept for qos integration in web services. In *Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on*, pages 149 – 155, dec. 2003. doi: 10.1109/WISEW.2003.1286797.

M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller. Performance considerations for mobile web services. *Computer Communications*, 27(11):1097 – 1105, 2004. ISSN 0140-3664. doi: 10.1016/j.comcom.2004.01.015. ¡ce:title¿Applications and Services in Wireless Networks¡/ce:title¿.

C. Vecchiola, S. Pandey, and R. Buyya. High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pages 4 –16, dec. 2009. doi: 10.1109/I-SPAN.2009.150.

C. Wang, X. Yuan, H. Ning, and X. Lian. Similarity evaluation of xml documents based on weighted element tree model. In R. Huang, Q. Yang, J. Pei, J. Gama, X. Meng, and X. Li, editors, *Advanced Data Mining and Applications*, volume 5678 of *Lecture Notes in Computer Science*, pages 680–687. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-03347-6. URL http://dx.doi.org/10.1007/978-3-642-03348-3_71. 10.1007/978-3-642-03348-3_71.

H. Wang, Y. Tong, H. Liu, and T. Liu. Application-aware interface for soap communication in web services. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1 –8, sept. 2006. doi: 10.1109/CLUSTR.2006.311886.

Y. Wang and Y. hua Li. Deep web entity identification method based on improved jaccard coefficients. In *Research Challenges in Computer Science, 2009. ICRCCS '09. International Conference on*, pages 112 –115, dec. 2009. doi: 10.1109/ICRCCS.2009.36.

C. Werner and C. Buschmann. Compressing soap messages by using differential encoding. In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 540 – 547, july 2004. doi: 10.1109/ICWS.2004.1314780.

C. Werner, C. Buschmann, Y. Brandt, and S. Fischer. Compressing soap messages by using pushdown automata. In *Web Services, 2006. ICWS '06. International Conference on*, pages 19 –28, sept. 2006. doi: 10.1109/ICWS.2006.46.

K. Xiong and H. Perros. Service performance and analysis in cloud computing. In *Services - I, 2009 World Conference on*, pages 693 –700, july 2009. doi: 10.1109/SERVICES-I.2009.121.

C. Yanping, L. Zengzhi, W. Li, and Y. Huaizhou. Service-cloud model of composed web services. In *Information Technology and Applications, 2005. ICITA 2005. Third International Conference on*, volume 2, pages 97 –100, july 2005. doi: 10.1109/ICITA.2005.251.

G. Yongming, C. Dehua, and L. Jiajin. Clustering xml documents by combining content and structure. volume 1, pages 583 –587, dec. 2008. doi: 10.1109/ISISE.2008.301.