ABSTRACT

SAHIJWANI, VINEET. Optimization of the Processing in a Middleware Environment for a Set of XML Variants. (Under the direction of Dr. Rudra Dutta).

The middleware has become an important product category since the 1990s, when companies started to adopt client-server architectures and had to deal with distributed computing problems. It employs the component based design to distribute the processing across the network. The middleware components use proprietary formats or more popularly XML for data exchange. There are different formats or variants of XML available which when used can improve the processing speed at a middleware component. However it is possible that the same format may not be suitable or be the best for all the components the XML goes through. Thus from a given set of XML variants or formats, there can be a particular XML format that will process faster as compared to others in a given middleware component. That same XML format however may not be suitable for other middleware components and some other format from the set might be.

In this thesis we consider the problem of improving the processing speed in the middleware, given a set of different XML formats. We propose the use of local optimization to achieve global optimization. We find the best suited XML format for each component. This is done with the help of a feedback based software monitoring system. The software collects data about the processing time taken in a particular component by different formats with the help of feedback from that component and then process that data to decide which format suits that component best. Once the formats are known, the data can be sent to the components in that format and it would result in faster processing in the system as a whole, as compared to, if a single format was used for all the components. We also numerically investigate the performance of a middleware environment with and without the feedback based software monitoring system.

Optimization of the Processing in a Middleware Environment for a Set of XML Variants

by

Vineet Sahijwani

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2006

Approved By:

Dr. Rada Chirkova	Dr. Matthias F. Stallmann

Dr. Rudra Dutta
Chair of Advisory Committee

To my parents ...

Biography

Vineet Sahijwani was born on June 23rd, 1979 in New Delhi, India. He graduated with a Bachelor of Engineering degree in Electronics Engineering from Thadomal Shahani Engineering College, Mumbai University, India, in June 2001. He worked as an automation engineer in Samsung Electronics India Limited for a year after graduating. He then joined Infosys Technology Limited and worked as a Software Engineer for 1.5 years. In fall 2004 he joined the joined the Master of Science program in Computer Networking of the Computer Science department at North Carolina State University (NCSU), Raleigh, NC. After graduating from NCSU, he plans to join Cisco Systems, as a Software Engineer.

Acknowledgements

I would like to express my sincere appreciation to my adviser Dr. Rudra Dutta, for his mentorship through the research process and my masters degree. This thesis would not have been possible without his ideas, support and encouragement. I thank Dr. Rada Chirkova and Dr. Matthias F. Stallmann for participating on my advisory committee.

I would like to thank my parents, Mr. Lachman Sahijwani and Mrs. Sheela Sahijwani, and my sister, Leena Sahijwani, for their love, support and encouragement during all the phases of my life.

I would also like to thank my roommates, Pritesh Patwa, Bhushan Bhatt and Pranav Parikh and my friends Harshvardhan Joshi, Tanu Sharma and Rachana Gupta for being caring, supportive and understanding. They made my graduate life much more fun and interesting than it otherwise would have been. Also I would like to thank my friends, Abhinandan Karmakar, Ashish Mahajan, Amrit Rajani, Purwa Sakaria, Tarun Bhatia, Ashish Bakhru and Amit Aggarwal for all the full filled moments we shared.

Finally, I would like to thank the faculty and staff at the Computer Science Department of the North Carolina State University, for providing me with an opportunity and a great environment to pursue my graduate studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Middleware is a computer software that connects software components or applications and is used most often to support complex, distributed applications. It adopts the component based design to distribute the processing across the network. Middleware gained popularity in the 1980s and became an important product category in the 1990s when companies started to adopt client-server architectures and had to deal with distributed computing problems. Message-based or message-oriented middleware (MOM), introduced as a product in the mid 1990s, is a type of middleware which uses asynchronous messaging for communication and data exchange between the components. It has a large share of the middleware market with various MOMs available such as IBM's Websphere MQ [19] , Sun Java System Message Queue [13] based on Java Message Service [12], Microsoft Message Queuing Server [24] and BEA's MessageQ [18].

The MOM components use proprietary formats or more popularly Extensible Markup Language (XML) [28] for data exchange. XML has become the de-facto standard for data exchange over the Internet. There are different formats of XML available which perform faster for a particular operation as compared to other operations. Hence using a particular format of XML can result in an improvement of the processing speed at a particular middleware component. However it is possible that the same format may not be suitable or the best for all the components the XML goes through. Thus from a given set of XML variants or formats, there can be a particular XML format that will process faster as compared to others in a given middleware component. That same XML format however

may not be suitable for other middleware components and some other format from the set might be.

In this thesis we consider the problem of improving the processing speed in the middleware, given a set of different XML formats. We propose the use of local optimization to achieve global optimization. We find the best suited XML format for each component. This is done with the help of a feedback based software monitoring system. The software collects data about the processing time taken in a particular component by different formats with the help of feedback from that component and then process that data to decide which format suits that component best. Once the formats are known, the data can be sent to the components in that format and it would result in faster processing in the system as a whole, as compared to, if a single format was used for all the components.

While a lot of research has been done on various formats of XML, the application of these formats to optimize the processing speed in middleware has not been addressed before. Thus this is a novel contribution in the area of middleware and XML.

The rest of this thesis is organized as follows. Chapter 2 describes the concepts and terminology used and also gives an outline of the related work done in this area. Chapter 3 gives a more formal and detailed description of the identified problem . Chapter 4 presents the various design approaches for the monitoring software, which provides a solution to the problem. Chapter 5 discusses the numerical results obtained by running the monitoring software in a simulated middleware environment. Chapter 6 discusses possible directions for future research and concludes this thesis.

Chapter 2

Context

This chapter discusses the various concepts, terminologies and research related to this thesis. It begins by discussing the concepts and terminologies used and how it is applicable to this thesis. It then proceeds to discuss the research work done to optimize XMLs specially in the fields of XML specific compression and query over XML data. The chapter is concluded by discussing the various commercial applications available to measure the performance of a XML based middleware.

2.1 Concepts and Terminology

The various concepts along with the terminology used are as follows:

2.1.1 Middleware

Middleware is a software that functions as a translation layer and sits between an application residing on one server and any number of clients that want access to that application. In short, middleware allows users to interact with one another and with applications in a heterogeneous computing environment. Middleware also automates business operations, tying together a company's back-end and front-end operations. Middleware functions can be divided into three main categories: application-specific, information-exchange and management and support middleware.

Application-specific middleware provides services for various classes of applications such as distributed-database services, distributed-data/object-transaction processing and specialized services for mobile computing and multimedia. Information-exchange middleware handles the exchange of information across a network. It is used for tasks such as transferring data, issuing commands, receiving responses, checking status and resolving deadlocks. Management and support middleware is responsible for locating resources, communicating with servers, handling security and failures and monitoring performance.

This thesis is based on the processing that happens in the middleware. We focus mainly on the application-specific middlware and the information exchange middleware.

2.1.2 Componentization of Software

Componentization refers to a modular design architecture that structures and constrains the interactions among elements of a software system. This design architecture prescribes the pathways along which components communicate, and the precise manner in which one component request information or processing services from another. This approach applied to software development is referred to as Component Based Development (CBD).

The middleware adopts the component based development to distribute the processing across the network. In this thesis we consider the processing at each middleware component separately.

2.1.3 Document Object Model (DOM)

The DOM [5] specification defines the Document Object Model as a platform - and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them.

In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data

rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

In the DOM, documents have a logical structure which is very much like a tree. Each document contains zero or one doctype nodes, one root element node, and zero or more comments or processing instructions; the root element serves as the root of the element tree for the document.

DOM is the data structure that most XML parsers (e.g., Xerces [2]) have been developed to make use of. Such an implementation requires that the entire content of a document be parsed and stored in memory. Hence, DOM is best used for applications where the document elements have to be accessed and manipulated in an unpredictable sequence and repeatedly.

Another method of parsing the XML is by using SAX [17] stream parsing. The SAX parsers process the XML document sequentially, is faster than DOM and uses less memory. However it does not provide random access and is read only. It cannot modify an XML document. It however can create a new document with modifications. Since we need to access the XML randomly and the XML processing requires modifications on the XML, we are using DOM to parse the XML. Also typically in a middleware environment memory is not a constraint.

2.1.4 Serialization

The basic concept of object serialization is the ability to read and write objects to byte streams . The serialization process deals with flattening out an object tree such that all the raw data that make up an object, including all the objects referenced by that object, get saved. When it's time to read back an object, it should be possible to recreate the original object. The process of recreating the original object is called de-serialization.

2.1.5 Extensible Markup Language (XML)

XML was created mainly for ease of data exchange over the Internet. Today it has become the de-facto standard of data exchange over the Internet. There are various attributes which can be used to characterize a XML. Some of the attributes used in our reseach are as follows:

Size

It is the size of the XML file in bytes. Generally speaking the larger the XML, the more it can be compressed because of the possibility of the presence of redundant information. Also it takes more time to parse as the whole document has to be loaded into the memory.

Depth

When the XML is represented as a DOM tree, the depth of the XML is the depth of the DOM tree from the root element. Usually an increase in the depth of an XML tends to decrease the compression performance, since the redundancies across separated subtrees cannot be used in compression. Also the parsing time increases with the increase in depth of XML.

Number of Distinct Tags

It is the total number of distinct tags in the XML. An increase in the number of distinct tags tends to decrease the compression performance in terms of compression ratio as there is less redundant information.

Total number of Elements

It is the total number of elements in the XML. The performance of the compression and parsing is dependent on a combination of number of distinct tags and the total number of elements cause the combination determines the redundancy in the markup.

2.2 Related Work

This section discusses the prior research work done in the optimization of XML specially in the fields of XML specific compression and query over XML data.

2.2.1 XML Compression

There have been several proposals for XML specific compression formats including XMill [15], XGrind [25], XPRESS [11], WBXML [16], Millau [23] and XCQ [27].

XMill

XMill doesn't propose a new compression algorithm but a framework in which existing algorithm can be leveraged to compress XML data. XMill incorporates and combines the existing compressors to apply them to heterogeneous XML data. It uses zlib (the library function version of gzip), a few simple, data type compressors and can also include user defined compressors for complex, application specific data types. It is not designed to work with query processor. It is targeted at data exchange and data archiving. XML consists of tags, attributes and data. They are assigned to different containers. The structure consisting of XML tags and attributes are assigned to structure container and are compressed separately from data. Data values are grouped into data container containers based on the data value's path and user defined container expressions. Each container is then compressed separately. Users can also associate semantic compressors with containers. A decompresser XDemill is also proposed. After loading and unzipping the containers, XDemill parses the structure container and invokes the semantic decompresser for data items and generates the output.

Using default settings XMill compressed to 45% - 60 % of gzip. Using semantic compressors XMill reduced the size to 35 % - 47 % of gzip's. For more text-like data sets, XMill performs only slightly better than gzip. XMill's compression ratio is better than gzip only in the case when the file is typically over 20KB because of additional bookkeeping overhead involved with XMill. Under default setting XMill is generally as fast as gzip. Using semantic compressor, XMill is faster than gzip. XDemill speed is comparable to gunzip. XMill has slight advantage over gzip in data exchange for slow networks. Otherwise the difference is not significant.

XGrind

XGrind is a compression tool that directly supports queries in the compressed domain. Instead of compressing at the granularity of the entire document, it compresses

at the granularity of individual element/attribute values using a context-free compression scheme based on Huffman encoding. This means that exact-match and prefix-match user queries can be entirely executed directly on the compressed document, with decompression restricted to only the final results provided to the user. Further, range or partial-match queries require on-the-fly decompression of only those element/attribute values that feature in the query predicates, not the entire document. A special feature of XGrind is that the compressed document retains the structure of the original document, permitting reuse of the standard XML techniques for processing the compressed document.

XPRESS

XPRESS is a XML compressor which supports direct updates and efficient evaluations of queries on compressed XML. XPRESS is a homomorphic compressor which preserves the structure of the original XML data in compressed XML data. XPRESS adopts a encoding method called the reverse arithmetic encoding, which is intended for encoding label paths of XML data and applies diverse encoding methods depending on the type of data value. The compression scheme is labeled as semi-adaptive which uses a preliminary scan of the input file to gather statistics and compresses using those statistics in the next scan of the data. It uses a type inference engine to infer the type of data values of each distinct element. For numeric typed data values, XPRESS applies binary encoding first and then differential encoding with the minimal value. For textual data, XPRESS adopts the arithmetic encoder and the dictionary encoder.

The core modules of XPRESS are XML Analyzer and XML Encoder. XML Analyzer consists of two submodules: the statistics collector and the type inference engine. The query processor consists of a query parser, a query transformer and a query executor. The query parser separates the values from the label path expression in the query. The query transformer transforms the single path expression to intervals. The query executor evaluates the tokens of encoded elements in compressed XML data whether their encoded values are in an interval of the sequence or not. When inserting or deleting XML elements, the frequencies of tags are changed. The statistics for reverse arithmetic encoding is changed when new tags appear in updates. The update processor renews the statistics by analyzing the newly inserted XML fragment and then according to the between the renewed statistics and the current statistics, a partial decompression of compressed XML data is performed.

Using the renewed statistics, the XML fragment and the partially decompressed XML data are re compressed.

The compression ratio is defined as 1-(Size of compressed XML data/Size of original XML data). The average compression ratio of XPRESS is 71% while that of XMill is 90% . XMill and gzip have the fastest compression time since they compress in one scan. XPRESS shows better compression time than XGrind. On an average, the query performance of XPRESS is 2.13 times better than that of XGrind and 4.31 times better than that of XMill. The performance of the update processor of XPRESS is about 5.7 times faster than that of the naive approach with XMill.

WBXML

WBXML specification of Wireless Application Protocol (WAP) defines a compact binary representation of XML. It is designed to allow for compact transmission with no loss of functionality or semantic information. The format preserves the element structure of XML, allowing a browser to skip unknown elements or attributes. It encodes the parsed physical form of an XML document, ie, the structure and content of the document entities. Meta-information, including the document type definition and conditional sections, is removed when the document is converted to the binary format.

It proposes a table based encoding of element names and attributes with tokens into what is called a code space. There are two classifications of tokens: global tokens and application tokens. Global tokens are assigned a fixed set of codes in all contexts and are unambiguous in all situations. Global codes are used to encode inline data (eg, strings, entities, opaque data, etc.) and to encode a variety of miscellaneous control functions. Application tokens have a context-dependent meaning and are split into two overlapping code spaces. These two code spaces are the tag code space and the attribute code space. The tag code space represents specific tag names. Each tag token is a single-byte code and represents a specific tag name. The attribute code space is split into two numeric ranges representing attribute prefixes and attribute values respectively. The character data is transmitted as strings inline or as a reference in a string table which is transmitted at the beginning of the document.

Millau

Millau is an extensive WBXML implementation and is extended by separating the structure and content. It take advantage of the associated schema if available to achieve better compression. The structure is encoded by WAP WBXML encoding and the content using the standard text compression techniques. Instead of the two overlapping application code spaces, there are three - the third one being the attribute value code space. Millau extends the DOM APIs to support BDOM (Binary DOM) by allowing lookup and return using tokens. Also SAX [17] is extended to support encodings instead of string names in the SAX API methods.

XCQ

XML Compression and Querying System (XCQ) is based on a technique called DTD Tree and SAX Stream Parsing (DSP). It consists of two parts - XCQ Compression Engine and XCQ Query Engine. The XCQ Compression Engine provides a efficient compression of XML documents that conform to a given DTD without involving user expertise. It is based on two ideas (1) to extract the structural information from the XML document that cannot be inferred from a given DTD during the parsing process (2) to group the data elements in document based on the corresponding tree paths in the DTD tree. Both of them are done by use of a SAX event stream generated by the XML Parser and DTD tree generated by a module which uses outputs from DTD parser. The compressed documents in XCQ adopt a partitioned path-based data grouping which supports evaluating queries without running a full decompression and also helps to achieve a higher compression ratio. The Structure stream and the blocks in the Data Stream are then compressed individually using a text compressor like gzip. A better compression ratio than XMill can be achieved at the expense of compression time. XCQ also achieves a better compression ratio and compression time than XGrind. The XCQ Querying Engine finds the data stream to which the query results belongs and then decompresses the whole data group to find the exact match.

2.2.2 XML Query

The World Wide Web Consortium (W3C) has proposed a standard for querying over XML - XQuery [4]. It provides flexible query facilities to extract data from many various types of XML data sources. It uses the structure of XML intelligently and can express queries across all kinds of data, whether physically stored in XML or viewed as XML via middleware.

There are also several schemes in the literature for indexing XML and semi-structured data for both simple [14] and branching [10, 22] path expressions. Such indexing assists in accelerating the time to retrieve the results of a query on a XML.

2.2.3 XML Parsers

XTalk [21] is a pseudo-binary XML format intended to make the XML parsing task even more simple than what was originally envisioned by the XML creators. XTalk attempts to deviate from the human readability criteria as little as possible by representing only structural aspects of the document in binary, and leaving all data components in the standard UTF-8 character format. The XTalk specification was developed as a serialization format of the XPath XML data-model [3].

The Apache Xerces [2] is the most popular high performance parser and has set the standard in XML parser implementation, providing both Java and C++ versions that implement both the DOM [5] and SAX [17] standard parsing APIs. On the C side, the expat [6] parser is the fastest publicly available XML parser.

2.3 Applications for performance measurement of XML based middleware

IBM Tivoli provides a few applications for measuring the perfomance of XML in middleware components.

2.3.1 IBM Tivoli Composite Application Manager for Response Time Tracking

IBM Tivoli Composite Application Manager (ITCAM) for Response Time Tracking [7] is a tool that can be used to measure the end to end user response time over the middleware. It helps to visualize the transaction path through the middleware including the response time at each step. Hence it can be used to get the processing time of each component and the overall response time of the system.

2.3.2 IBM Tivoli Composite Application Manager for WebSphere

IBM Tivoli Composite Application Manager (ITCAM) for WebSphere [8] is a tool that can be used to obtain the key performance metrics over WebSphere. Hence it can be used to get the response time at each component and the overall response time of the system for WebSphere middleware.

Chapter 3

Problem Description

The high volume of traffic handled by a typical enterprize-level middleware system makes it desirable to streamline the processing as much as possible. Our goal is to develop a performance monitoring and tuning algorithm for use in a middleware system by focusing on the suitability of different XML variants for different middleware processing. The proposed performance monitoring function will reside individually at each software component of a middleware system, and use observed correlation between XML variants and processing time to decide on the variant to use in transmitting to other components. The envisioned process is described in more detail below.

As discussed in Chapter 2, a lot of research has been done in the field of optimizing XMLs. The research focuses on a particular operation like compression, query, parsing etc. as the primary objective. Some papers like XPRESS focuses on secondary objectives also like query over compressed data and update over compressed data. The resultant of this research has been the different formats of XMLs optimized for different operations. The Table 3.1 shows the various formats and their optimizations

The formats to use can be chosen based on the parameters to be optimized. For example if data exchange is the main objective, then XMill, gzip or WBXML can be used to improve the transmission time. In real world, the operations are not as simple as addition, compression, transformation etc. but a series of such operations combined to form a complex operation. However the complex operations can have more of one particular operation than others. For example a complex operation can have multiple queries. In such a case, the

Table 3.1: The formats and their optimizations

Sr. No	Objective	Sub Objective	Format/tool
1	Compression Time		compress, gzip, XMill, XCQ
		Query Time	XPRESS, XGRIND, XCQ
		Modification Time	XPRESS
2	Decompression Time		gunzip, XMill
3	Compression Ratio		XMill, XCQ
4	Parsing Time		XTalk, expat, Xerces
5	Search/Query Time		XQuery, toxin
6	Data Exchange Time		gzip, XMill, WBXML

format which optimizes the query operation might be more suitable. Since there is no single format which is best suited for all operations, we can instead find the most suitable format from a given set. This can be done by using all the formats to perform the operation and comparing the processing time of each format.

There are applications available to measure the response time of each component in middleware as mentioned in section 2.2. The purpose of these applications is to monitor the overall performance of the system. These applications helps in analyzing and pin pointing the problem areas in the middleware. However any analysis and subsequent changes has to be done manually. The result of the changes can be analyzed and a decision can me made to keep the changes or make further changes to fine tune the performance of the system. Figure 3.1 shows the components in an online trading system. The image was taken from IBM website.

The diagram shows the different paths the message can take from the client to the database in the back tier. The blue, red and green lines shows three possible paths for the message. The performance measuring applications can measure the end to end time taken and also the time taken at each component. Any analysis of these measurements have to be done manually. Also all the changes to improve the performance has to be applied manually and their results also have to be analyzed manually.

We take the process of measuring the performance of the system a step further by introducing a feedback based monitoring software in the middleware components. We apply the different formats of XML at each component and collect the feedback of the processing time. The XML can be characterized by certain attributes like size, depth,

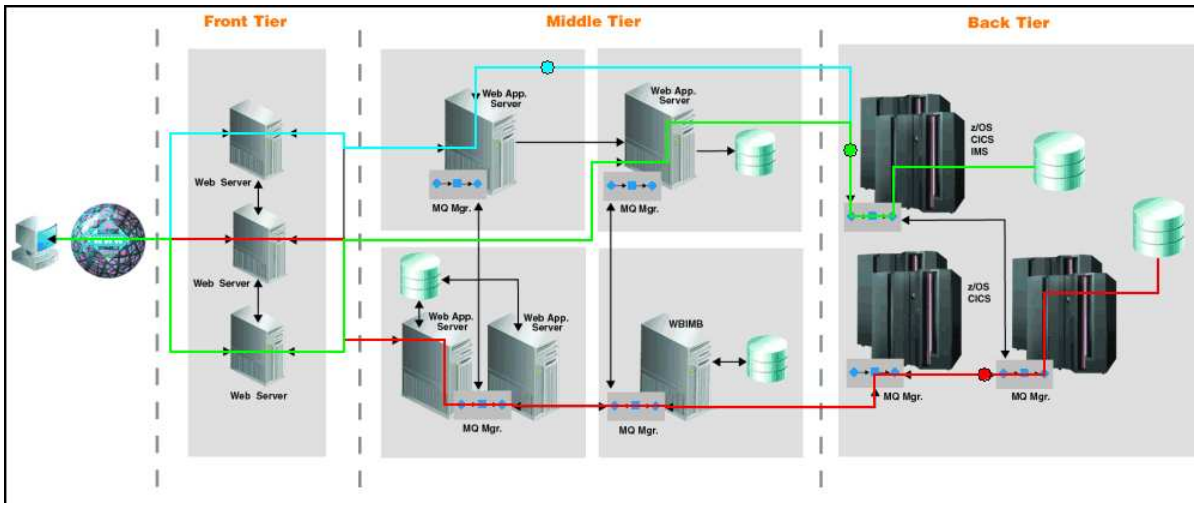


Figure 3.1: The components in an online trading system

number of distinct tags, number of elements etc. All these attributes are collected by the software along with the processing time the XML took at a particular component. Once sufficient data is collected, a decision can be made regarding which format suits a particular component best. This decision is made based on the characteristic of the incoming XML. The attributes are matched with the previously collected data and only that data is used which has similar attributes.

In the real world scenario, it is possible that a particular component is shut down and brought up in a different environment causing a change in the performance of the component. It is also possible that the implementation of the processing in a component is changed causing a change in the performance of the system. In such a case all the data that is gathered is rendered useless since it is no longer applicable to the changed component. If the same data is continued to be used it might degrade the performance of the system, if the changed component favors a different format of XML. Hence it is important that the system is adaptive. It should detect and react to any changes in a component. Our software is designed to be adaptive and senses any changes in a component. It then gathers data of the changed component and makes decision based on the new data.

Hence the goal of the feedback based monitoring software is not only to measure the performance of the system, but also to analyze it and automatically improve or fine tune the performance of the system, based on the given XML formats.

Chapter 4

Design of The Feedback Based Monitoring Software

Chapter 3 briefly discusses the design of the feedback based monitoring system. In this chapter we will describe the design in detail, including the various approaches that could be followed for the implementation. We commence by looking at a typical middleware software component and the various operations done inside it. We then proceed to explain the modules inserted in the middleware component for the feedback bassed monitoring system.

4.1 Middleware Software Component

A typical middleware software component is as shown in Figure 4.1. It receives messages from the previous components connected to it via the message queue. Any message received is de-serialized to obtain the XML transmitted. The XML is then processed. The processing can be looked as a complex operation which can be broken down into simple XML operations like Query, Addition, Update, Deletion, Transformation to a different format, etc. Most of the operations require the XML to be parsed into a DOM or use SAX parsing. After the XML processing, the next component to send the XML to is decided based on

certain attributes of XML. The XML is then serialized to a byte stream and send to the next component.

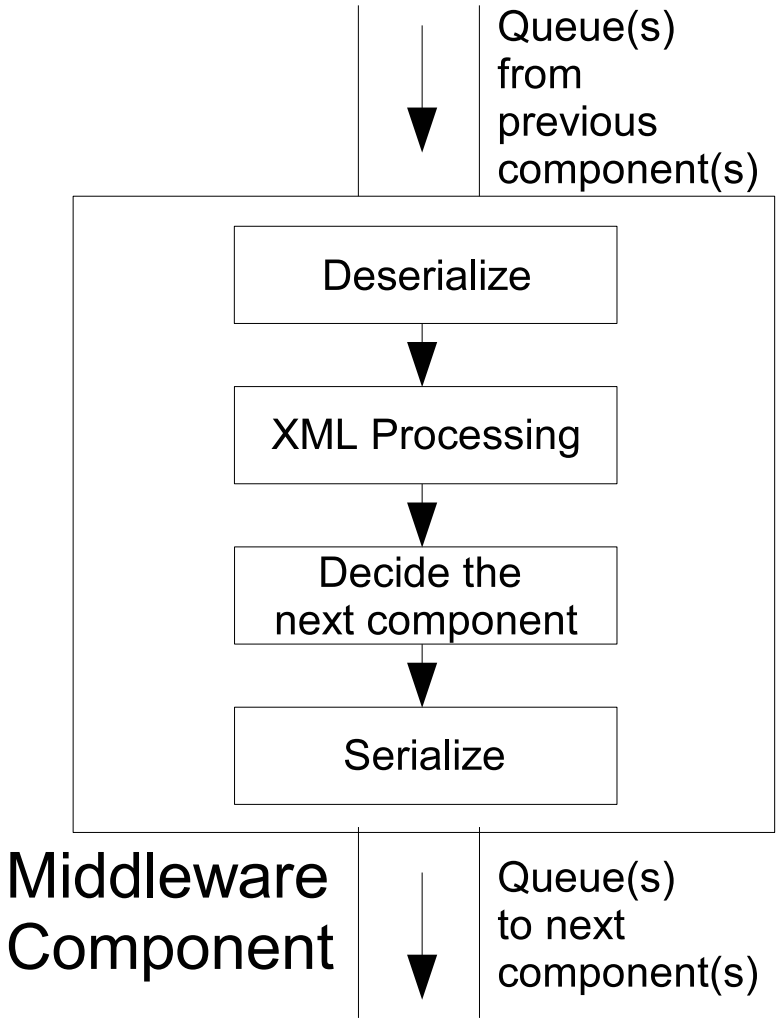


Figure 4.1: The processing in a middleware component

4.2 Feedback based monitoring software

The feedback based monitoring software is introduced in the middleware software component. It is as shown in Figure 4.2.

It receives messages from the previous components connected to it via the message

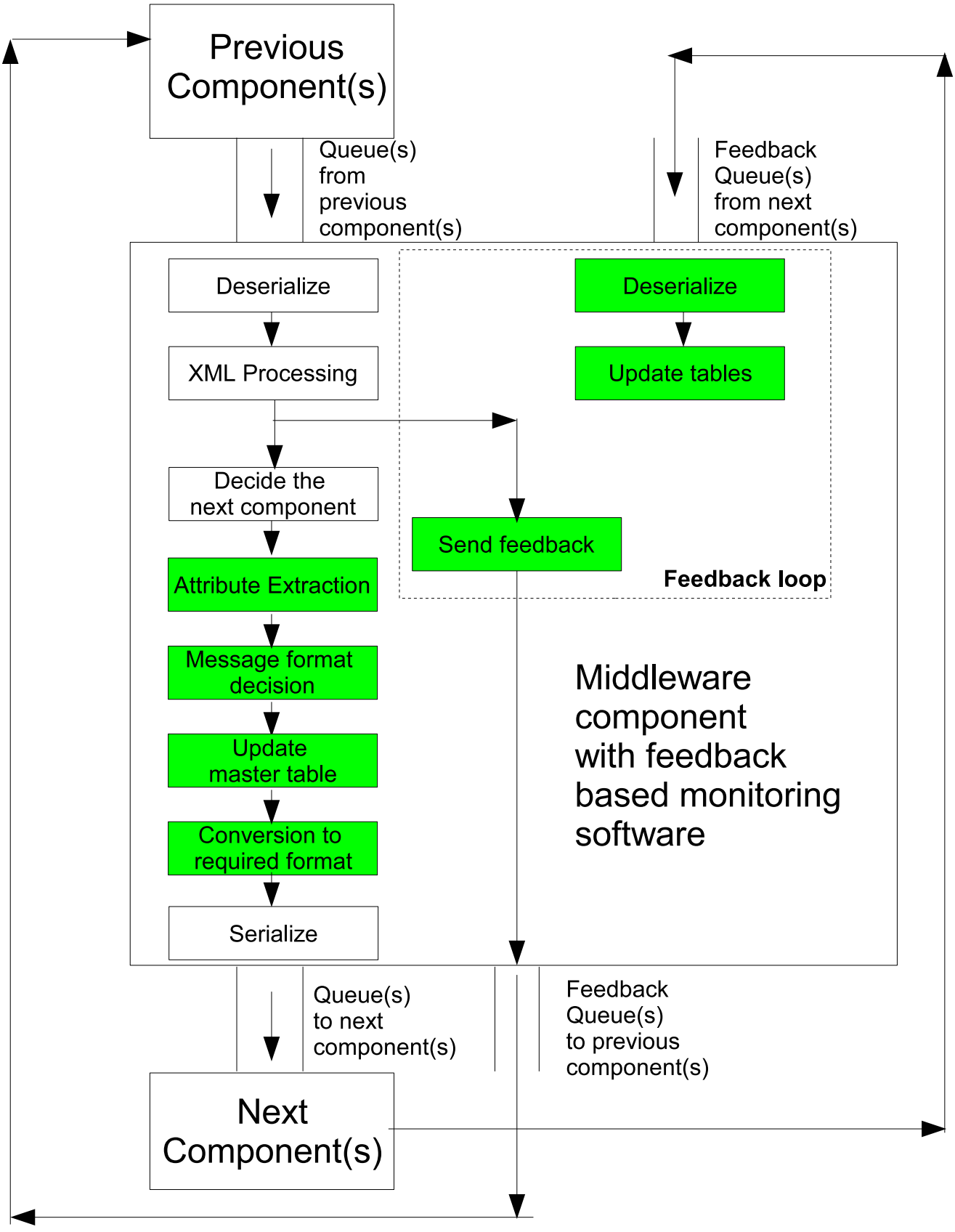


Figure 4.2: Feedback based monitoring software

queue. Any message received is de-serialized. The queue from which the message came is examined. If it is from the feedback queue the it is the feedback message and is handled by the feedback loop. The feedback loop extracts the message id and processing time and updates the master tables and all the tables which are used to make the decision of choosing the format to send the message in.

If the message is from a regular queue then the control information (message id and format type) and the XML is extracted. The XML is then processed. The processing time is recorded and a feedback message is sent to the component from which the message came. The feedback message contains the processing time and the control information to recognize the XML which was sent. After the XML processing, the next component to send the XML to, is decided based on certain attributes of XML.

The attributes namely size, depth, number of distinct tags and total number of elements are extracted to characterize the XML. A decision is made regarding the format in which the XML has to be sent based on the previous collected data and the characteristic of the current XML. The master table is updated with the attribute values and the format decided. The XML is converted to the chosen format if it is not the same. The XML is then serialized along with the control information to a byte stream and send to the next component.

The detailed design constructs and algorithms are explained as follows:

4.2.1 Tables

There are various tables used in the design to collect the data and make the calculations faster. The tables are updated for each message and intermediate calculations are performed with the update so when the data is required, it is just a lookup from the tables and there is no latency there. The tables used are as follows:

Master Table

As the name suggests it is the master table where the records of all the messages sent are kept. Each component assigns a unique message id to each xml that is sent out to the next component. This message id is used to identify the xml on receiving the feedback. The table keeps a record of the message id, the component the message is sent to, the

format in which it was sent, the values of the attributes of the XML, the overhead time in the current component, the transmission time of the message and the processing time of the next component. The table is designed as show in Table 4.1. This table is used to keep the record for the message so when the feedback comes the values of the attributes can be extracted and used to update the other tables used.

Table 4.1: Master Table

Message ID	Component No	Format Type	Time Taken	Overhead Time	Transmission Time	Attr 1 Value	...	Attr N Value

Correlation Table

This table is used for calculation of the correlation coefficients between the processing times and the attribute values for each format. Its is used to store the intermediate values of the expression used for calculating the correlation coefficient. The correlation coefficient ρ_{XY} of the random variables X and Y is defined as [26]

$$\rho_{XY} = \frac{cov(X,Y)}{\sigma X \sigma Y}$$

Where cov is the covariance of the random variables and σ_X and σ_Y is the standard deviation of X and Y respectively. They are defined as

$$\text{cov(X,Y)} = \frac{\Sigma XY}{N} - \overline{X} * \overline{Y}$$
$$\sigma^2{}_X = \frac{\Sigma X^2}{N} - \overline{X}$$
$$\sigma^2{}_Y = \frac{\Sigma Y^2}{N} - \overline{Y}$$

Where $\overline{X} = \frac{\Sigma X}{N}$ and $\overline{Y} = \frac{\Sigma Y}{N}$

Hence the table stores the values of Σ X, Σ Y, Σ X^2 , Σ Y^2 and Σ XY so the covariance and standard deviation can be calculated faster with each update. A minimum of 50 values are required for the correlation coefficient to make a confident decision based on it. This number was established after running the simulation and analyzing the behavior. The results in section 5.3.1 proves this value is sufficient to make a decision. The table is designed as shown in Table 4.2.

Table 4.2: Correlation Table

Component No	Attribute	Format Type	Correlation Coefficient	N	Σ X	Σ Y	Σ X^2	Σ Y^2	Σ XY

Attribute Table

This table is used for the calculation of the mean time taken at the next component for a range of attribute values for each format type. It stores the intermediate values so the calculation for the mean and standard deviation is faster with each update. In practice, 30 values are required for the mean processing time to make a confident decision based on it [26]. This approximation is the cornerstone of statistics. All the fundamental distributions and tests in the theory and practice of statistical inference use such approximations. Hence the table also stores the last 30 values for each attribute and format for the adaptive algorithm explained later. The table is designed as shown in Table 4.3.

Table 4.3: Attribute Table

Attribute Range Min	Attribute Range Max	Σ Time	Σ $Time^2$	N	Mean	Standard Deviation	Last 30 Values

4.2.2 Feedback Loop

Our model is feedback based. The component informs the previous component regarding the performance of the received XML format. The feedback consists of the message id which is the same as the one sent by the previous component, the format type and the processing time it took at the current component. It is send as a separate message to the previous component using the feedback queue. The queue at which the message arrives is used to identify if it is a regular message or a feedback message.

4.2.3 Attribute Extraction

As explained earlier, the attributes namely size, depth, number of distinct tags and total number of elements are extracted to characterize the XML. While making the decision regarding the format, these attributes are used to match the characteristic of the current xml with the previously collected data and the decision is made considering only the relevant data. The algorithms for the extraction of each attribute is as follows:

Size of XML

The algorithm is as shown in Algorithm 1. Calculating the size of the xml is very simple. The input is received in a byte buffer. The size of the byte buffer is the size of the xml. In case the xml is compressed, it is first uncompressed to the original size and then the size is measured.

Algorithm 1 Determining the size of the XML

```
{Input : byte stream b_stream containing the XML}
{Output : size i_size of the XML}
convert b_stream to byte array b_buffer
i_size ← size of b_buffer
return i_size
```

Maximum Depth of the XML

The algorithm is as shown Algorithm 2 and Algorithm 3. Calculating the maximum depth requires a recursive algorithm. Initially the input is parsed to a DOM and the root element is extracted. Then the children of the root element are extracted. The depth count becomes 1. Then recursively the children of each child are extracted till the last node in the DOM tree is reached. That defines the depth of that particular child. The depths of each of the children of the root element are calculated and the maximum depth is found.

Number of Distinct Tags

The algorithm to calculate the distinct tags is as shown Algorithm 4 and Algorithm 5. It also requires a recursive algorithm. A data structure is maintained which stores each of the distinct tags. The DOM tree is traversed recursively as discussed above. Along the

Algorithm 2 Determining the maximum depth of the XML

{*Input* : byte stream b_stream containing the XML}
{*Output* : maximum depth i_maxdepth of the XML}
parse b_stream to a DOM d_xml
d_root ← root element of d_xml
i_maxdepth ← return value of the recursive algorithm to find the depth of a node with input as d_root
i_maxdepth ← i_maxdepth - 1 since the root element is at depth 0
return i_maxdepth

Algorithm 3 Recursive algorithm to find the depth of a node

{*Input* : node d_node of a DOM}
{*Output* : depth i_depth of d_node}
n_child ← set of all children of d_node
c_num ← number of n_child
if c_num > 0 **then**
  i_depth ← 0
  for i_count ← 1 to c_num **do**
    d_childnode ← i_count child of d_node
    i_tempdepth ← return value of the recursive algorithm to find the depth of a node with input as d_childnode
    i_tempdepth ← i_tempdepth + 1
    if i_tempdepth > i_depth **then**
      i_depth ← i_tempdepth
    end if
  end for
else
  i_depth ← 1
end if
return i_depth

way whenever a new tag is encountered, it is stored in the data structure. After the full tree is traversed, the number of elements in the data structure gives the number of distinct tags.

Algorithm 4 Determining the number of distinct tags of the XML

```
{Input : byte stream b_stream containing the XML}
{Output : number of distinct tags i_num of the XML}
parse b_stream to a DOM d_xml
d_root ← root element of d_xml
ds_distincttags ← data structure to store the distinct tags
ds_distincttags ← return value of the recursive algorithm to store the distinct tags in a
data structure with input as d_root, ds_distincttags
i_num ← the number of elements in ds_distincttags
return i_num
```

Algorithm 5 Recursive algorithm to store the distinct tags in a data structure

```
{Input : Node d_node of a DOM, Data structure ds_distincttags containing the distinct
tags}
{Output : ds_distincttags}
c_name ← tag name of d_node
if c_name is not in ds_distincttags then
   add c_name in ds_distincttags
end if
n_child ← set of all children of d_node
c_num ← number of n_child
for i_count ← 1 to c_num do
   d_childnode ← i_count child of d_node
   ds_distincttags ← return value of the recursive algorithm to store the distinct tags in
   a data structure with input as d_childnode, ds_distincttags
end for
return ds_distincttags
```

Number of Elements

The algorithm to calculate the number of elements is as shown Algorithm 6 and Algorithm 7. It also requires a recursive algorithm. The DOM tree is traversed recursively as discussed above. Along the way for each node encountered, the count of the number of elements is increased. After the full tree is traversed, the number of elements counted is the total number of elements in the xml.

Algorithm 6 Determining the total number of elements of the XML

{*Input* : byte stream b_stream containing the XML}
{*Output* : Total number of elements i_num of the XML}
parse b_stream to a DOM d_xml
d_root ← root element of d_xml
i_num ← return value of the recursive algorithm to find the number of elements with input as d_root
return i_num

Algorithm 7 Recursive algorithm to find the number of elements

{*Input* : Node d_node of a DOM}
{*Output* : Number of elements i_num of d_node}
i_num ← 1
n_child ← set of all children of d_node
c_num ← number of n_child
for i_count ← 1 to c_num **do**
  d_childnode ← i_count child of d_node
  i_num ← i_num + return value of the recursive algorithm to find the number of elements with input as d_childnode
end for
return i_num

4.2.4 Deciding the message format

This is the crux of the feedback based monitoring system. This module decides the format in which the message has to be send to the next component. All the data collected in the tables are used to make the decision.

Initially all the correlation coefficients are examined to determine the attribute which has the maximum correlation. This operation has the time complexity of O(n). A minimum of 50 values are required to consider the correlation coefficient. If 50 values are not present for a particular format, the message is sent in that format. If more than one format does not have enough data, the incoming format is given the preference as the time to convert to another format is saved.

Once the attribute is identified, the correlation coefficient is checked for consistency across the formats. If the correlation coefficient is inconsistent, the message is sent in the incoming format as we cannot base the decision on inconsistent data. If it is consistent, the xml's attribute is matched with the data ranges in the attribute table. For the particular attribute range, the format having the minimum mean processing time is selected. However for the particular attribute range a minimum of 30 values needs to be present format in

order to make an intelligent decision. If any format does not have the 30 values, then it is selected. In case there are more than one format not having the 30 values, the preference is given to the incoming format as the time to convert to another format is saved.

As discussed in chapter 3, in the real world scenario it is possible that a particular component is shut down and brought up in a different environment causing a change in the performance of the component. It is also possible that the implementation of the processing in a component is changed causing a change in the performance of the system. In such a case all the data that is gathered is rendered useless since it is no longer applicable to the changed component. If the same data is continued to be used it might degrade the performance of the system, if the changed component favors a different format of XML. Hence it is important that the system is adaptive. It should detect and react to any changes in a component.

Our software is designed to be adaptive and is designed to detect and react to such changes. There is a switch in the software which is used to turn on or turn off the adaptive mechanism. Hence the software runs in three modes : non-adaptive, adaptive scheme 1 and adaptive scheme 2 which are explained as follows.

In the non adaptive mode, the software actively does not detect any such changes. If there is a change in the component then it can go undetected. This is because after the initial data is collected and the best format is decided, it does the not send the other formats and only the best format. Hence even if the processing speed of the other formats improves drastically, it will not be detected by the system since that format is no longer sent.

To overcome the shortcomings of the non adaptive mode, the adaptive scheme 1 was designed. In this scheme, once in a while the other formats are sent randomly. The algorithm is designed such that if the number of messages sent for any format falls below 25% of the number of messages sent for best format, then the message is sent for that format with a 50% probability. In this way it is ensured that the other formats are also sent. If there is a change in the component then it will be detected. After sufficient messages are sent, the data will reflect the change and if another format is more favorable suitable then it will sent. The drawback of this method is that it is very slow to react to the change after it has happened. The earlier gathered data is not discarded and it will take a large number of messages to change the mean processing time to reflect the processing time after the change.

The adaptive scheme 2 was designed to further refine the adaptive process. It keeps

the algorithm for adaptive scheme 1 and adds additional logic on top of it. In this scheme, a history of the processing time of the last 30 messages for each format and attributes range is kept. The number 30 was chosen according to the Central Limit Theorem as discussed previously. If the mean processing time of these last 30 messages falls out of the 95% confidence interval of the overall mean processing time, we can conclude that a change has occurred in the next component. In this case all the previously gathered data is discarded and the procedure of gathering data is started all over again. As soon as sufficient data is gathered the software can make an intelligent decision on the best format for the next component. Hence this scheme reacts as soon as a change is detected.

The algorithm is as shown in Algorithm 8.

4.2.5 Conversion

The conversion from one XML format follows a very simple logic. It uses the available encoders and decoders to convert. In case direct conversion is not possible from one format to another, then it is converted to a intermediate format which is supported by both the formats.

Algorithm 8 The message format decision

```
{Input : The current format i_currformat}
{Output : The next format i_nextformat}
{i_num: Number of messages per format}
{i_numattr: Number of messages for each attribute range and each format}
{i_adaptive: 0 for non adaptive, 1 for adaptive scheme 1 and 2 for adaptive scheme 2}
if i_adaptive <> 0 then
  i_maxnum ← max(i_num for all formats)
  i_random ← random number between 0 and 1
  if i_num < 0.25 * i_maxnum and i_random > 0.5 for any format i_adaptformat then
    i_nextformat ← i_adaptformat
    return i_nextformat
  end if
end if
if i_num < 50 for any format i_format then
  i_nextformat ← i_format
  return i_nextformat
end if
Read all the correlation coefficients i_corr in a data structure ds_correlation
i_maxcorr ← max value in ds_correlation
i_attr ← attribute corresponding to i_maxcorr
if i_corr is consistent for all formats then
  if i_numattr < 30 for any format i_attrformat then
    i_nextformat ← i_sizeformat
    return i_nextformat
  end if
  i_min ← min(processing time for all formats)
  i_nextformat ← format corresponding to i_min
  i_mean ← mean processing time corresponding to i_min
  if i_adaptive = 2 then
    if i_mean is not consistent with last 30 readings then
      flush all tables
      i_nextformat ← i_currformat
    end if
  end if
end if
return i_nextformat
```

Chapter 5

Demonstration in Simulated Environment

5.1 Setup

The feedback based monitoring system was implemented using Java 2 Platform, Standard Edition 5.0 (J2SE 5.0) to make it portable across platforms. Java API for Xml Processing (JAXP) was used for XML related operations. A message oriented middleware environment was simulated using Java Messaging Server (JMS) which is present in the Java Enterprise Edition Software Development Kit (Java EE SDK). The feedback based monitoring system was run in the simulated middleware environment to obtain the numerical results. Sun Java System Application Server 9.0 was used as the JMS implementation provider as it was easily available. The packages were built and deployed using Java BluePrints [27] build system and application layout structure as it was available with Java EE and Sun Java System Application Server 9.0.

5.2 Implementation

The program has three main modules named the initiator, components and terminator. The initiator program is the xml generator and converts the xml to a byte message and send it to the components. The components is the simulated middleware environment. It also has the logic of the feedback based monitoring systems embedded in it. The terminator is a terminator for the messages. All the messages coming out of the system are fed to this module.

5.2.1 Input

The input to the middleware software components were a set of XMLs. The XMLs were generated based on four attributes size, depth, number of distinct tags, number of elements. Each of the attributes are divided into 3 categories small, medium and large. The range of values for the attributes are as shown in Table 5.1.

Table 5.1: The ranges of the various attributes used

Attributes/Category	**Small**	**Medium**	**Large**
Size (in bytes)	1000-15000	35000-65000	350000-650000
Depth	2-3	4-6	8-10
No of Distinct Tags	2-10	20-30	45-55
Number of Elements	2-50	51-200	201-1000

Each of the category is chosen randomly for each of the attributes. Once the category is chosen then the value is chosen from the range of that category. Once the values of the attributes are chosen then the xml generator is called with the attribute values as the parameters. It returns the xml as a byte buffer. The algorithm of the xml generator is explained in Appendix Section 7.1.

5.2.2 Formats

The software is implemented to work with 3 formats. These formats were chosen because their implementation was available in Java which allowed the integration with our software. New formats can be added with minimal change in the software. The formats supported are as shown in Table 5.2

Table 5.2: Formats

Format 0	Plain XML
Format 1	GZipped XML (The XML file compressed using GZip)
Format 2	WBXML

The XML is compressed to GZip format using the java.util.zip library. The encoding and decoding to WBXML is done using the open source jWAP package [9] which is the Java implementation of the Wireless Application Protocol (WAP) . It is available at www.sourceforge.net. The package had a problem decoding the XML entered. Since the package was open source, on investigation it was found that there was a synchronization problem in the encoding and decoding. The decoder was reading some extra bytes causing an error. There was another problem where the encoder accepted only 1024 bytes as the inline text in the XML. Both the problems were fixed and then it worked perfectly with our test set.

5.2.3 Time Measurement

We use the Java Virtual Machine Profiler interface to measure the processing, overhead and transmission time. It is the java.lang.management.ThreadMXBean interface which provides the implementation and allows to measure the CPU time spent by the thread. The precision of the clock that measures this processing time is in microseconds even though it is in nanoseconds in the API. The time it takes to complete a single operation like Add, Query, Update is too small to measure with the clock of that precision. Hence all the operations are done repetitively so that we can measure the time confidently.

5.2.4 Output

The output of the simulation was a series of comma seperated files (.csv) containing data on correlation, mean processing time for different formats and attributes for each component. The .csv file were generated after every 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500 and 5000 messages in the system.

5.3 Results

5.3.1 Test Scenario

In the test scenario the components are arranged as shown in Figure 5.1.

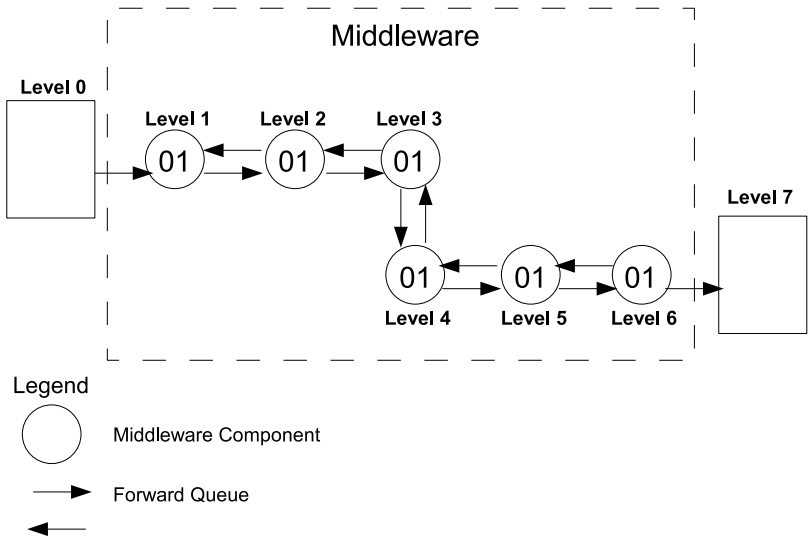


Figure 5.1: The arrangement of components in scenario 1

The operations that takes place at each component are as shown in Table 5.3. All the operations are done repetitively so that we can measure the time confidently as the clock is not very accurate. The data used to generate the graphs is available in Appendix Section 7.3

Table 5.3: Operations at each component

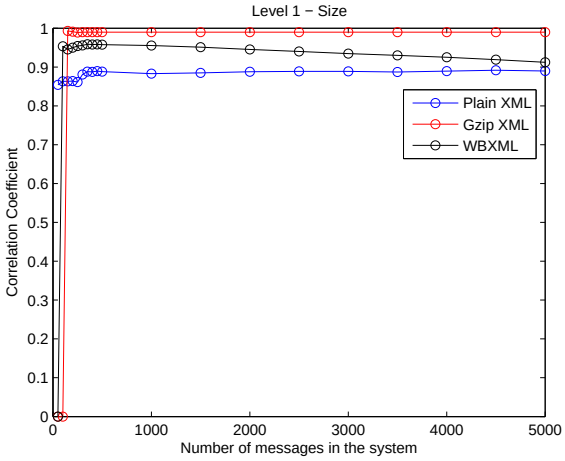
Level 1 Component 01	Parse, Parse, Parse, Parse, Parse, Parse, Parse, Parse, Parse, Serialize
Level 2 Component 01	Parse, Query, Query, Query, Query, Query, Query, Query, Query, Serialize
Level 3 Component 01	Parse, Add, Add, Add, Add, Add, Add, Add, Add, Serialize
Level 4 Component 01	Parse, Update, Update, Update, Update, Update, Update, Update, Update, Serialize
Level 5 Component 01	Parse, Delete, Delete, Delete, Delete, Delete, Delete, Delete, Delete, Serialize
Level 6 Component 01	Parse, Transform, Transform, Transform, Transform, Transform, Transform, Transform, Transform, Serialize

Non Adaptive

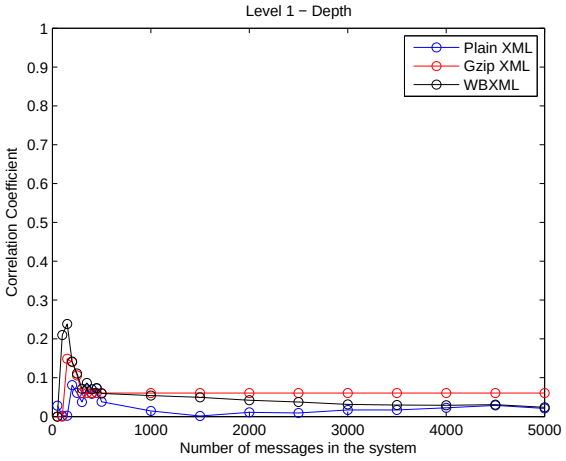
The Figure 5.2 shows the correlation coefficient plotted against the number of messages sent in the system at level 1 component 01 which collects the feedback and data for level 2 component 01. The graphs are plotted for the correlation coefficient of all the three formats supported. Figure 5.2(a) is for the size attribute, Figure 5.2(b) is for the depth attribute, Figure 5.2(c) is for the distinct tags attribute and Figure 5.2(d) is for the number of elements attribute. There is the training period of 50 values for each format. We do not make any decisions based on data as the data is not sufficient till then. From the graphs it can be seen that for each attribute and format the correlation coefficient remains reasonable stable after the training period . There are some small changes but since the coefficient value does not vary much, we can confidently use these values after the first 50 values are gathered. This proves the assumption we had stated in 4.2.1 that 50 values are enough for the correlation coefficient to stabilize.

Figure 5.3 shows the results obtained at level 1 component 0. Figure 5.3(a) shows the correlation coefficients for each attribute and format after the run. It can be seen that the size attribute has the maximum correlation and is very consistent across the formats. Hence in the software the size attribute is chosen to make decisions. Figure 5.3(b) shows the mean processing time in microseconds for the different ranges and formats. It can be noted that WBXML has the minimum time for small XMLs while plain XML has minimum time for medium and large sized XMLs. Hence the software should send WBXML for small XMLs and plain XML for medium and large sized XMLs. This is can shown with the help of Figure 5.3(c) which plots the number of messages sent per format for the different ranges of size attribute. For small XMLs all the messages are sent in WBXML format while for medium and large XMLs, the plain format is sent. Figure 5.3(d) shows the total number of messages sent to the next component in the different formats. The number of messages sent in plain XML is roughly double of WBXML since it is favored for two ranges as compared to one for WBXML.

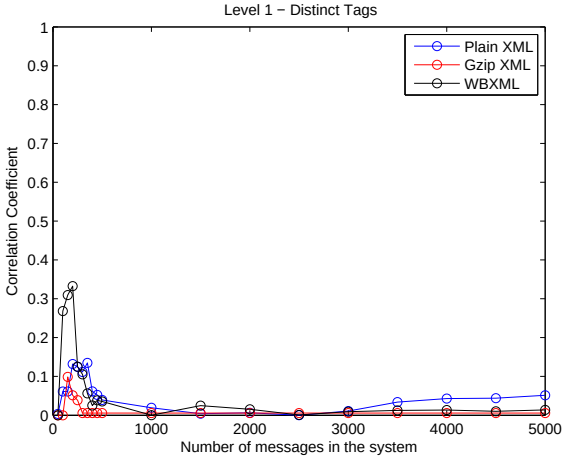
Figure 5.4 shows the results obtained at level 2 component 01. From Figure 5.4(a) it can be seen that the size attribute has the maximum correlation and is very consistent across the formats. Hence again in the software the size attribute is chosen to make decisions. From Figure 5.4(b) it can be noted that plain XML has the minimum time for all sizes of XMLs. Hence the software should send plain XML irrespective of the size



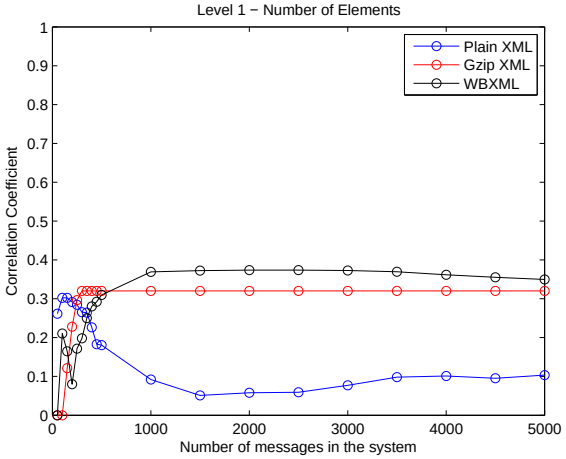
(a) For Size attribute



(b) For Depth attribute



(c) For Distinct Tags attribute



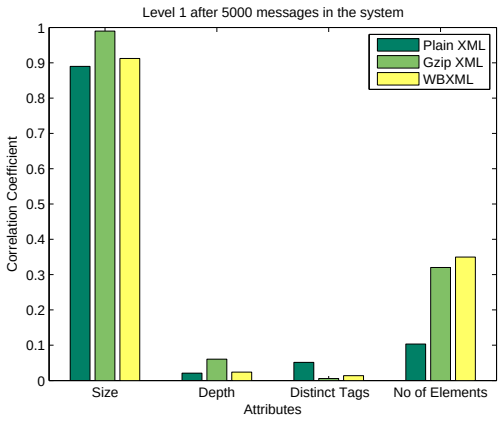
(d) For Number of Elements

Figure 5.2: The correlation coefficients at level 1 for test scenario

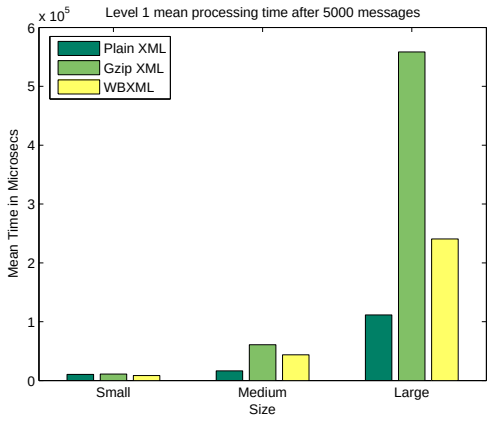
of the XML. This is shown with the help of Figure 5.4(c). For all XMLs, the messages are sent in plain format. Figure 5.4(d) further shows the all the message sent are in plain XML.

Similar results are obtained at level 3 component 01 and level 4 component 01. The graphs are available in Appendix Section 7.2.

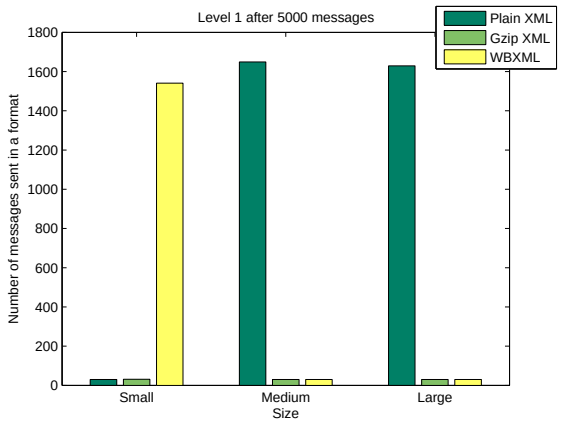
Figure 5.5 shows the results obtained at level 5 component 01. From Figure 5.5(a) it can be seen that the size attribute has the maximum correlation and is fairly consistent across the formats. Hence again in the software the size attribute is chosen to make decisions. From Figure 5.5(b) it can be noted that GZip XML has the minimum time for small XMLs, plain XML has minimum time for medium XMLs and WBXML has the minimum time for large sized XMLs. Hence the software should send GZip XML for small XMLs, plain plain XML for medium XMLs and WBXML for large sized XMLs. This is shown with the help of Figure 5.5(c). Even after the even random distribution among the size ranges of XML, there are a larger number of small sized XMLs. This is due to the fact that at level 5 component 01 the deletion operation is performed which reduces the size of the XML and makes medium sized XMLs fall in the small size range. Hence in Figure 5.5(d) it is shown that there are a greater number of Gzip XMLs sent since it was favorable for small sized XMLs.



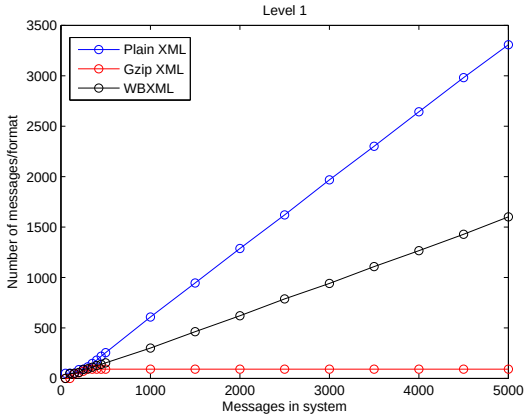
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

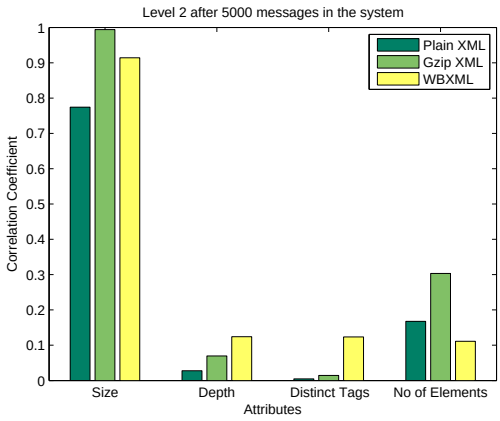


(c) Number of messages for each format as per the size range

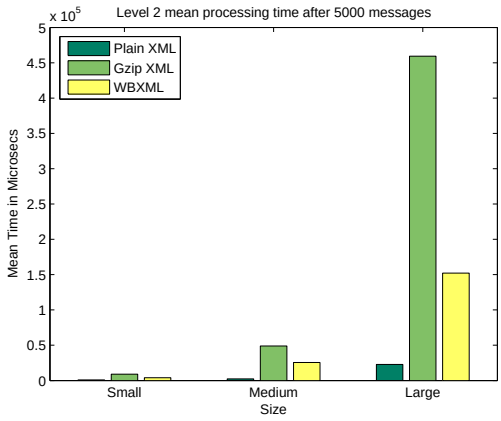


(d) Number of messages in the system for each format

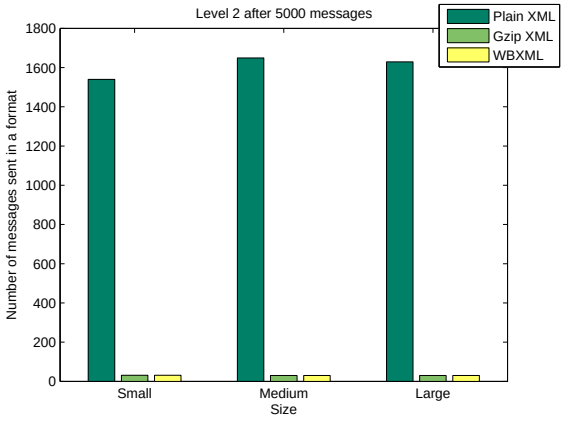
Figure 5.3: Level 1 - Test Scenario - Non Adaptive



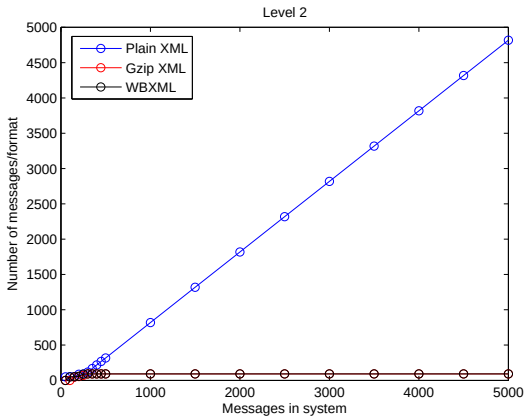
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

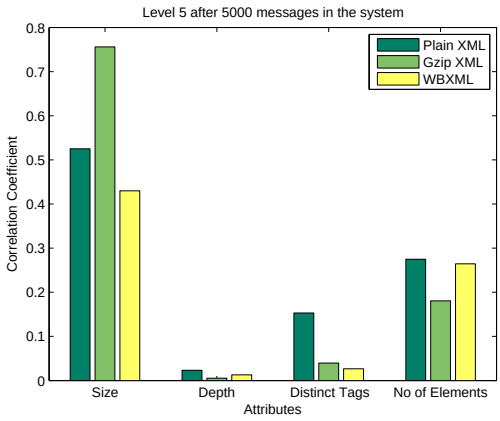


(c) Number of messages for each format as per the size range

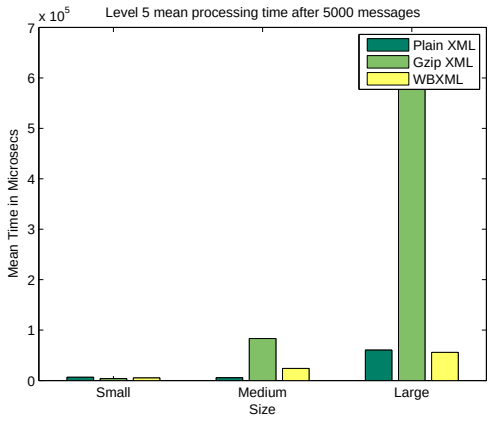


(d) Number of messages in the system for each format

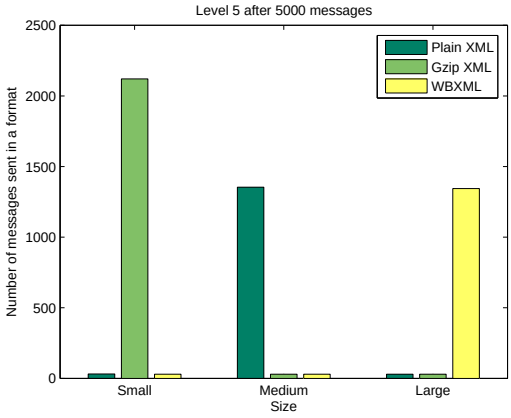
Figure 5.4: Level 2 - Test Scenario - Non Adaptive



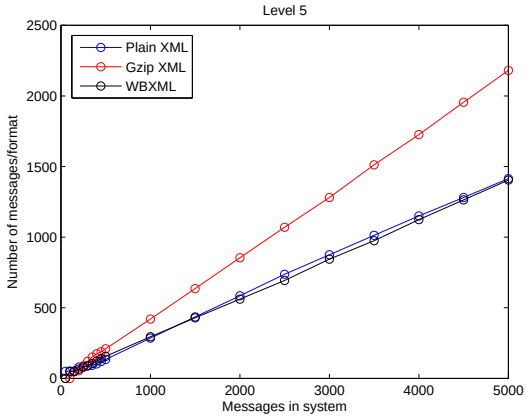
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute



(c) Number of messages for each format as per the size range



(d) Number of messages in the system for each format

Figure 5.5: Level 5 - Test Scenario - Non Adaptive

Adaptive Scheme 1

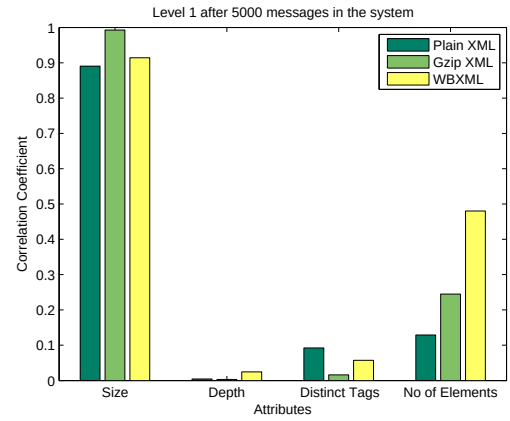
Figure 5.6 shows the results obtained at level 1 component 0 for the adaptive scheme 1. From Figure 5.6(a) it can be seen that the size attribute has the maximum correlation and is very consistent across the formats. Hence in the software the size attribute is chosen to make decisions. From Figure 5.6(b) it can be noted that WBXML has the minimum time for small XMLs while plain XML has minimum time for medium and large sized XMLs. Hence the software should send WBXML for small XMLs and plain XML for medium and large sized XMLs. This is can shown with the help of Figure 5.6(c). For small XMLs all the messages are sent in WBXML format while for medium and large XMLs, the plain format is sent. Figure 5.6(d) shows the total number of messages sent to the next component in the different formats. The number of messages sent in plain XML is roughly double of WBXML since it is favored for two ranges as compared to one for WBXML. There are a large number of messages sent in Gzip format even though it is not suitable because of the adaptive algorithm.

Figure 5.7 shows the results obtained at level 2 component 01. From Figure 5.7(a) it can be seen that the size attribute has the maximum correlation and is very consistent across the formats. Hence again in the software the size attribute is chosen to make decisions. From Figure 5.7(b) it can be noted that plain XML has the minimum time for all sizes of XMLs. Hence the software should send plain XML irrespective of the size of the XML. This is shown with the help of Figure 5.7(c). For all XMLs, the messages are sent in plain format. Figure 5.7(d) further shows the all the message sent are in plain XML. There are a large number of messages sent in Gzip and WBXML format even though it is not suitable because of the adaptive algorithm.

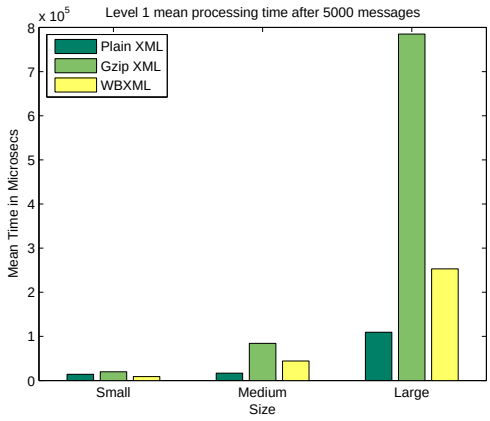
Similar results are obtained at level 3 component 01 and level 4 component 01. The graphs are available in Appendix Section 7.2.

Figures 5.8 shows the results obtained at level 5 component 01. From Figure 5.8(a) it can be seen that the size attribute has the maximum correlation and is fairly consistent across the formats. Hence again in the software the size attribute is chosen to make decisions. From Figure 5.8(b), which is a snapshot of the mean processing time after 1000 messages have been passed through the system, it can be noted that WBXML has the minimum time for all sizes of XMLs. Hence the software should send WBXML format irrespective of the size of the XML. This is shown with the help of Figure 5.8(c).

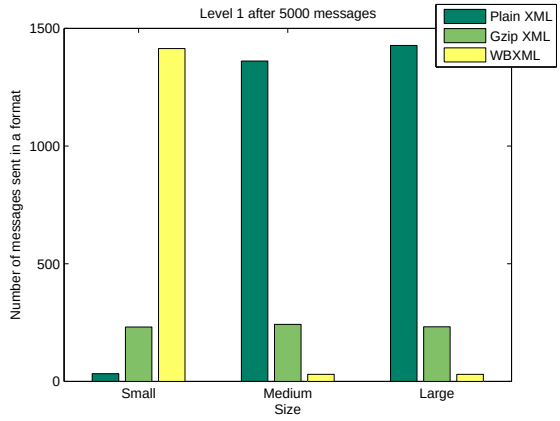
It can be noted that plain XML has more messages than WBXML for large XML sizes. This is because till the first 500 messages, the plain XML format was more suitable. A change occurred in the component and WBXML became more suitable. Since the system is designed to be adaptive, it detected the change and started sending out messages in WBXML format. After 1000 messages another change occurs in the component causing plain XML to be more suitable for small sized XMLs. This change is detected by the system because of the adaptive nature and plain XML is used for the small sized XMLs. This can be verified by the Figure 5.8(e) which clearly shows that for small sized XMLs, plain XML is the most sent format while for medium and large sized XMLs WBXML is sent more. Hence in Figure 5.8(f) it is shown that the number of messages sent in WBXML is roughly double of plain XML since it is favored for two ranges as compared to one for plain XML.



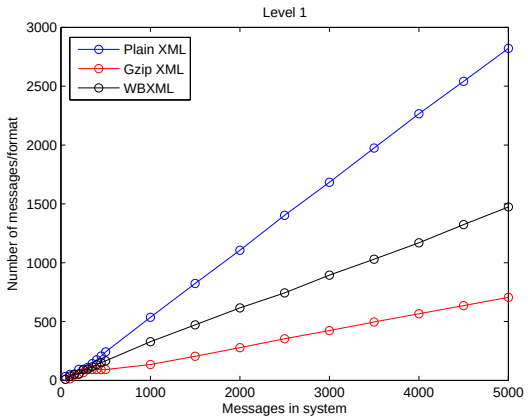
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

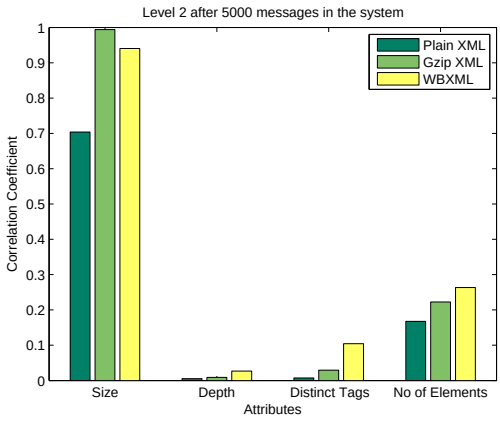


(c) Number of messages for each format as per the size range

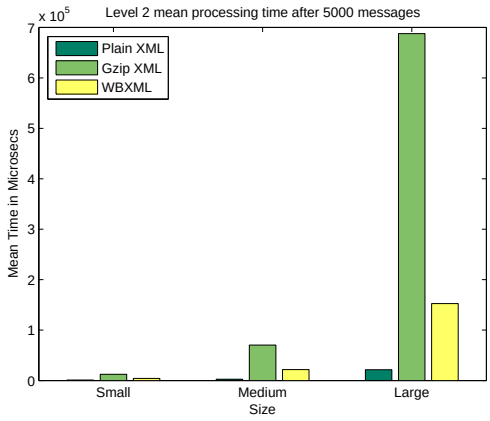


(d) Number of messages in the system for each format

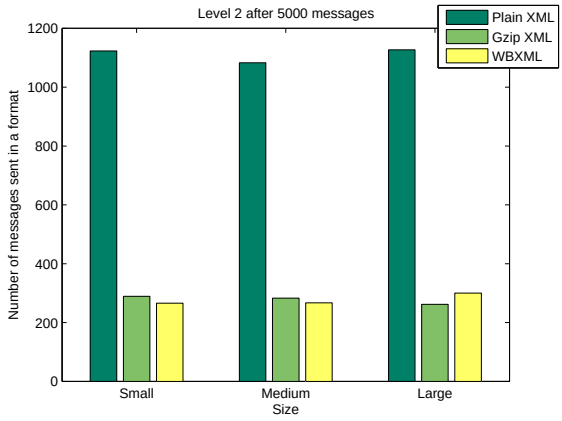
Figure 5.6: Level 1 - Test Scenario - Adaptive Scheme 1



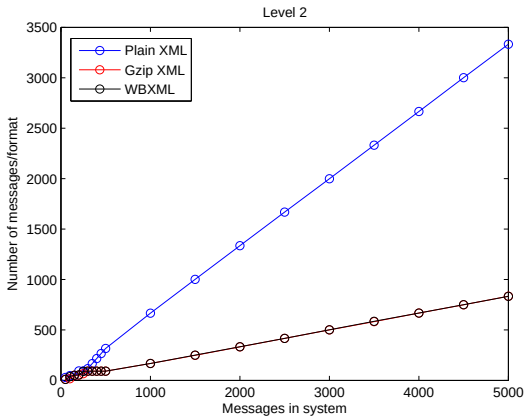
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

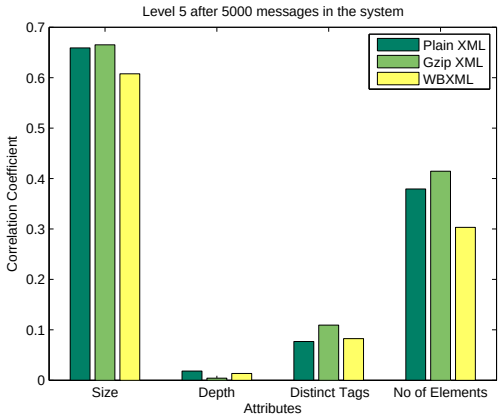


(c) Number of messages for each format as per the size range

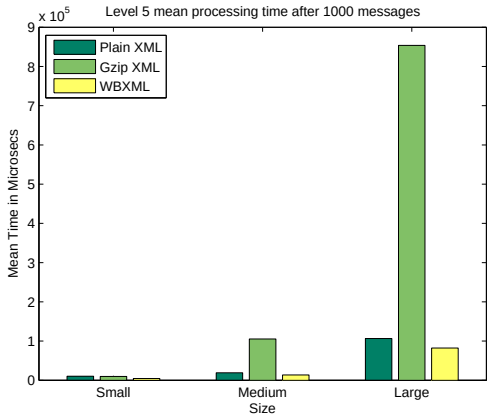


(d) Number of messages in the system for each format

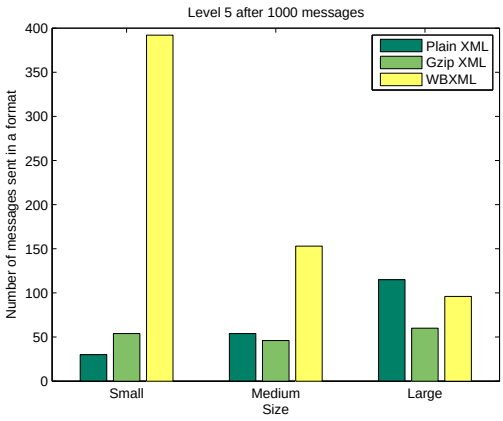
Figure 5.7: Level 2 - Test Scenario - Adaptive Scheme 1



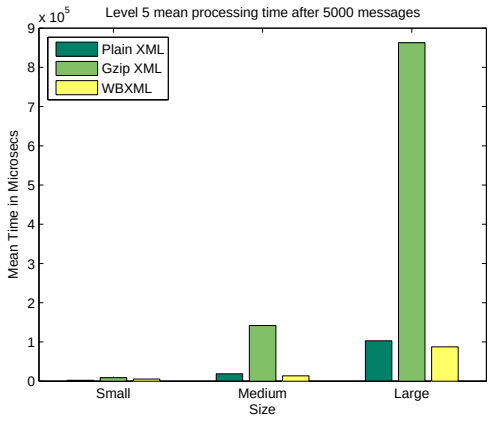
(a) Correlation coefficients



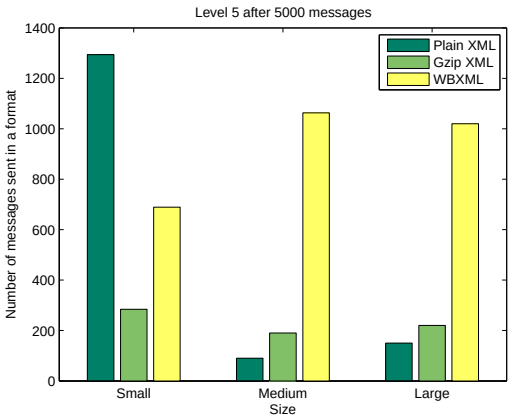
(b) Mean processing time for the three ranges of size attribute



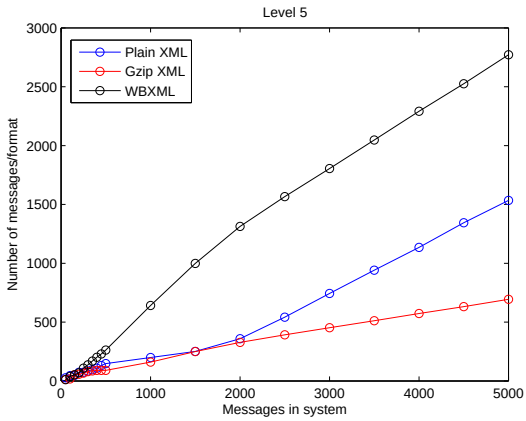
(c) Number of messages for each format as per the size range



(d) Mean processing time for the three ranges of size attribute



(e) Number of messages for each format as per the size range



(f) Number of messages in the system for each format

Figure 5.8: Level 5 - Test Scenario - Adaptive Scheme 1

Adaptive Scheme 2

Figure 5.9 and 5.10 shows the results obtained at level 1 component 0 for the adaptive scheme 2. From Figure 5.9(a) it can be seen that the size attribute has the maximum correlation and is very consistent across the formats. Hence in the software the size attribute is chosen to make decisions. From Figure 5.9(b) it can be noted that plain XML has minimum time for all sizes of XMLs. Hence the software should send plain XML irrespective of the size of XMLs. This is can shown with the help of Figure 5.9(c). Figure 5.9(d) shows the total number of messages sent to the next component in the different formats after 1000 messages.

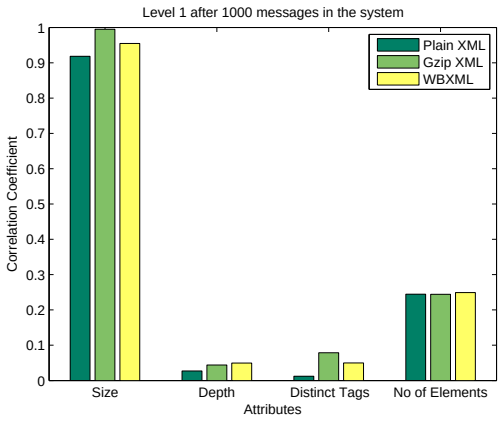
After 1000 messages, a change was made in level 2 component 01. This made the data inconsistent and caused a dump of the old data collected. After enough new data was collected, the decisions were made using this new data. This all can be shown with Figure 5.10.Figure 5.10(a) shows the correlation. From Figure 5.10(b) it can be noted that plain XML has minimum time for small and mid sized XMLs while WBXML has minimum time for large sized XMLs. Hence the software should send plain XML for small and mid sized XML and WBXML for large sized XML. This is can shown with the help of Figure 5.10(c). Figure 5.10(d) clearly shows the dump of data after 1000 messages and the data collection begins back from zero.

Figure 5.11 shows the results obtained at level 2 component 01. From Figure 5.11(a) it can be seen that the size attribute has the maximum correlation and is fairly consistent across the formats. Hence again in the software the size attribute is chosen to make decisions. From Figure 5.11(b) it can be noted that plain XML has the minimum time for all sizes of XMLs. Hence the software should send plain XML irrespective of the size of the XML. This is shown with the help of Figure 5.11(c). For all XMLs, the messages are sent in plain format. One thing can be noted from Figure 5.11(d) that there is a dump of data after around 250 messages. This is caused by the initial mismatch of data.

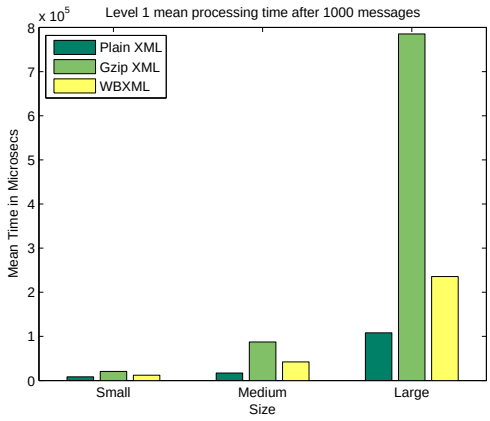
Similar results are obtained at level 3 component 01 and level 4 component 01. The graphs are available in Appendix Section 7.2.

Figures 5.12 shows the results obtained at level 5 component 01. From Figure 5.12(a) it can be seen that the size attribute has the maximum correlation and is fairly consistent across the formats. Hence again in the software the size attribute is chosen to make decisions. From Figure 5.12(b) it can be noted that plain XML has the minimum time

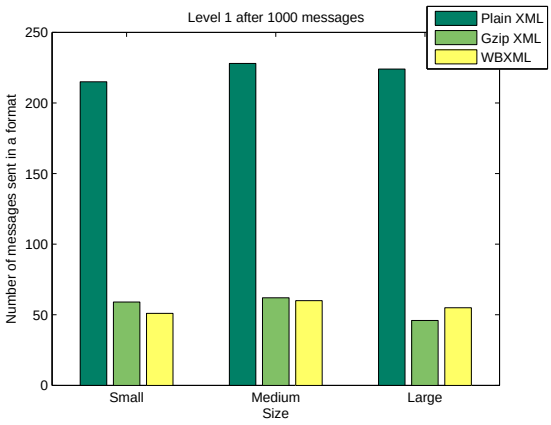
for small XMLs and WBXML has minimum time for medium XMLs and large sized XMLs. Hence the software should send plain XML for small XMLs and WBXML for medium XMLs and large sized XMLs. This is shown with the help of Figure 5.12(c). Even after the even random distribution among the size ranges of XML, there are a larger number of small sized XMLs. This is due to the fact that at level 5 component 01 the deletion operation is performed which reduces the size of the XML and makes medium sized XMLs fall in the small size range. Hence in Figure 5.12(d) it is shown that there are a almost an equal number of plain XMLs and WBXMLs even though WBXML is suitable for two ranges



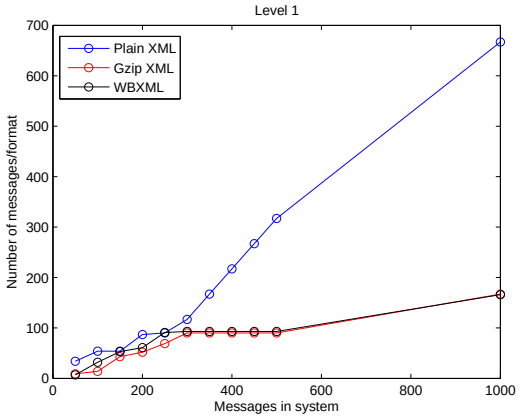
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

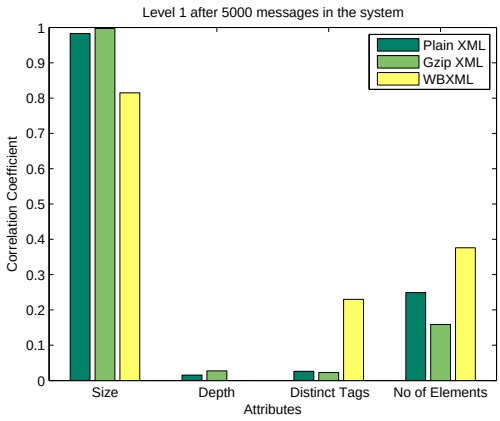


(c) Number of messages for each format as per the size range

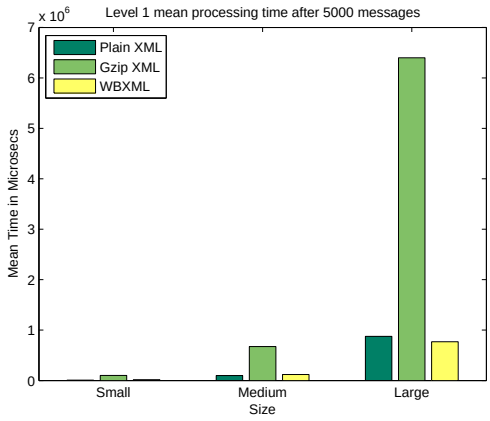


(d) Number of messages in the system for each format

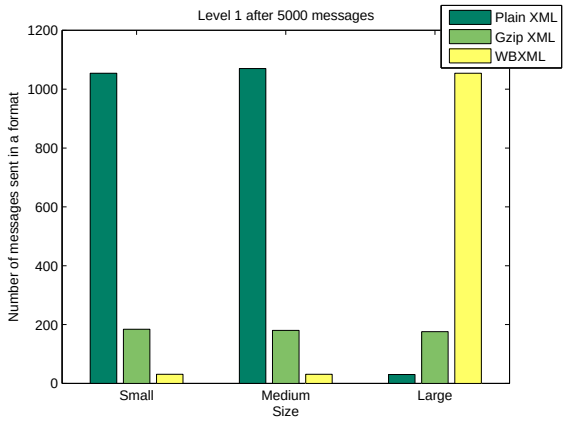
Figure 5.9: Level 1 - Test Scenario - Adaptive Scheme 2 - Part (a)



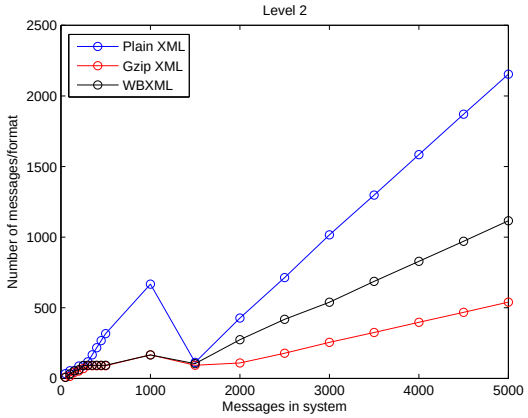
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

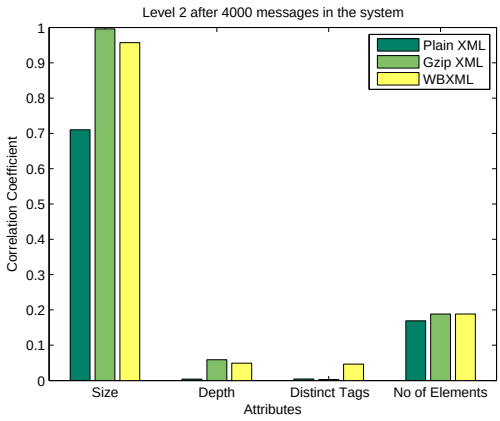


(c) Number of messages for each format as per the size range

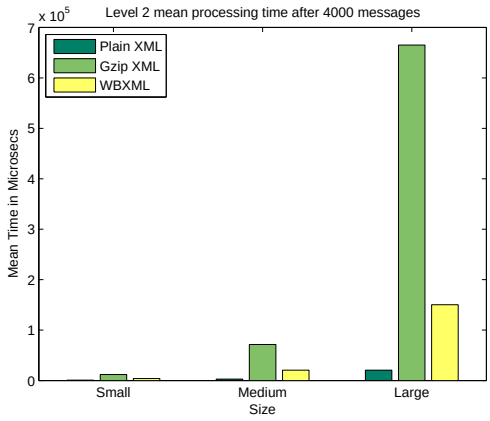


(d) Number of messages in the system for each format

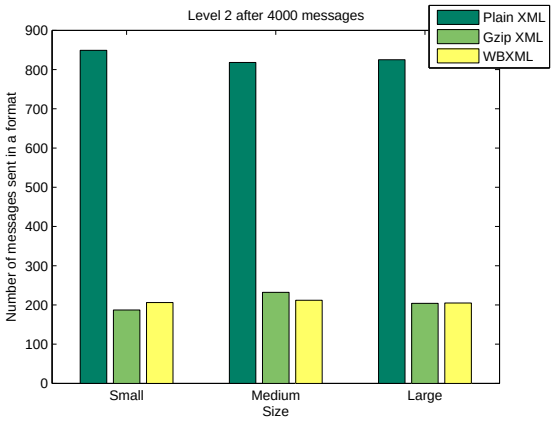
Figure 5.10: Level 1 - Test Scenario - Adaptive Scheme 2 - Part (b)



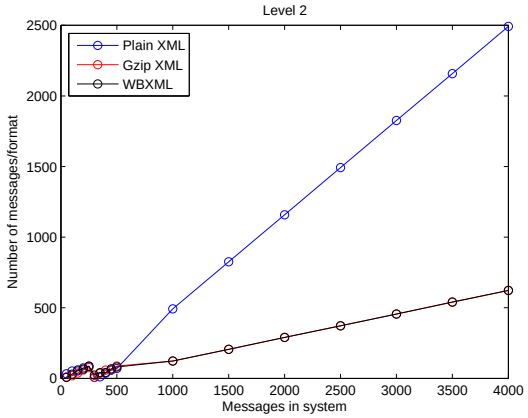
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

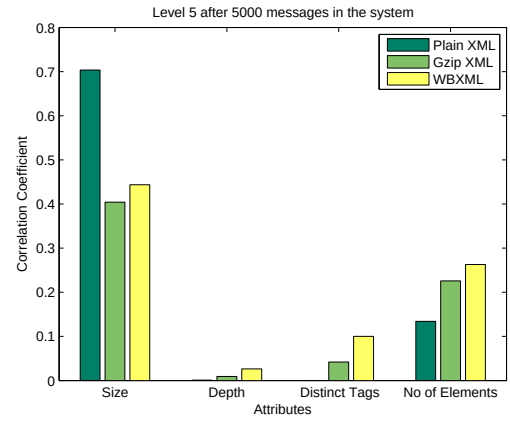


(c) Number of messages for each format as per the size range

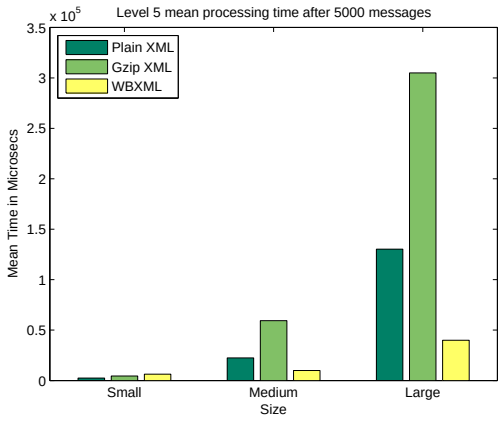


(d) Number of messages in the system for each format

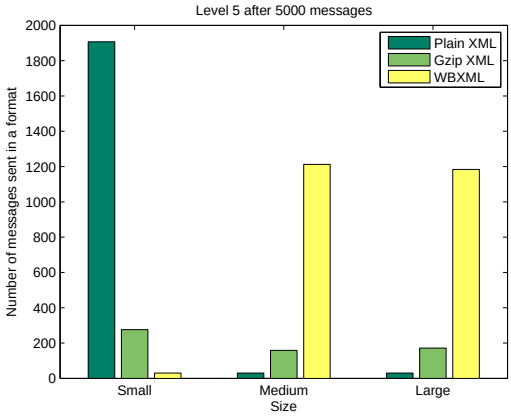
Figure 5.11: Level 2 - Test Scenario - Adaptive Scheme 2



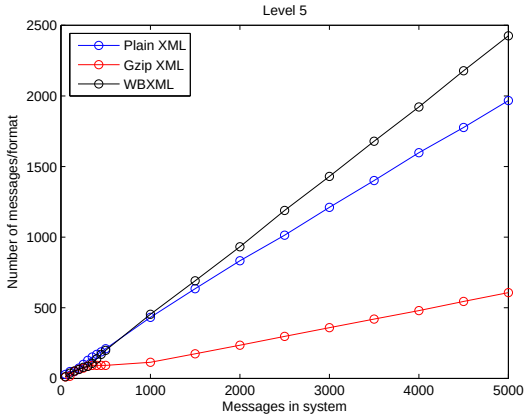
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute



(c) Number of messages for each format as per the size range



(d) Number of messages in the system for each format

Figure 5.12: Level 5 - Test Scenario - Adaptive Scheme 2

Chapter 6

Conclusion and Future Work

We looked at the problem of improving the processing speed in the middleware, given a set of different XML formats. Though a lot of work has been done on finding different formats of XML optimized for one operation or another and there are lot of applications which measure the performance of the middleware, the application of these formats to measure and optimize the processing speed in middleware has not been addressed before. Thus to the best of our knowledge, this is a novel contribution in the area of middleware and XML.

We designed and implemented a feedback based monitoring software which measures the performance of the system, analyzes it and automatically improves or fine tunes it, based on the given XML formats. This software was tested on a simulated middleware environment. The numerical results proved that our software identifies the best format from a given set for XMLs for a particular component and sends the message in that format. It also showed that the software is adaptive and detects and reacts to the changes in the processing speeds of any component. Finally the performance of the system as a whole improves by the use of this software.

While our primary focus was on designing a working system to enable performance monitoring and tuning, we note that as a secondary function, this software can be used as a profiling tool for a middleware system; that is provide an administrator with a tool to understand what XML variants are most suited to different parts of a middleware system at a given time. The specific results we produced in the simulation demonstration provide

an example of this capability.

The software can be improved if there are tools or packages available for accurately measuring the processing, overhead and transmission times in the system. Also considering the correlation of more than one attribute at a time, with the processing time, to make the decision can be investigated in the future. The XML generator can be improved to cover all well formed XMLs and more attributes so that the input set covers all the possible types of XMLs. The software can also be tested in a real world middleware to further refine the design.

In conclusion, we initiated investigation into a new problem area of practical importance. We believe a lot of useful and practical research can be undertaken in this area in the near future.

Bibliography

[1] BluePrints. http://java.sun.com/reference/blueprints/.

[2] The Apache Project. Xerces c++ version 2.3.0. http://xml.apache.org/xerces-c/.

[3] J. Clark and S. DeRose. XML Path Language (XPATH) Version 1.0. http://www.w3.org/tr/xquery/. November 1999.

[4] J. Robie J. Simeon D. Chamberlin, D. Florescu and M. Stefanascu. Xquery: A query language for xml. In *Technical report, World Wide Web Consortium*, February 2001.

[5] The World Wide Web Consortium. Document Object Model (DOM). http://www.w3.org/dom.

[6] Clark Cooper. Using expat. http://www.xml.com/pub/a/1999/09/expat/index.html. In *xml.com*, 1999.

[7] Tivoli Composite Application Manager for Response Time Tracking. http://www–306.ibm.com/software/tivoli/products/composite-application-mgr-rtt/.

[8] IBM Tivoli Composite Application Manager for WebSphere. http://www–306.ibm.com/software/tivoli/products/composite-application-mgr-websphere/.

[9] Java implementation of the Wireless Application Protocol (WAP). http://jwap.sourceforge.net/.

[10] Philip Bohannon J. F. N. Raghav Kaushik and H. F. Korth. Covering indexes for branching path queries. In *SIGMOD 2002*, 2002.

[11] M. Park J. Min and C. Chung. A compressor for effective archiving, retrieval, and update of xml documents. In *ACM Transactions on Internet Technologies*, August 2006.

[12] Java Message Service (JMS). http://java.sun.com/products/jms/.

[13] Sun Java System Message Queue Java Message Service (JMS). http://www.sun.com/software/products/message_queue/index.xml.

[14] Q. Li and B. Moon. Indexing and querying xml data for regular path expressions. In *VLDB 2001*, 2001.

[15] H. Liefke and D. Suciu. Xmill: An efficient compressor for xml data. In *Proceedings of ACM SIGMOD Intl. Conf. on Management of Data*, May 2000.

[16] B. Martin and w3c recommendation B. Jano. Wap binary xml content format. http://www.w3c.org/tr/wbxml/. June 1999.

[17] D. Megginson and D. Brownell. Sax. http://www.saxproject.org/.

[18] BEA MessageQ. http://www.bea.com/framework.jsp?cnt=index.htm&fp=/content/products/more/messageq.

[19] WebSphere MQ. http://www-306.ibm.com/software/integration/wmq/.

[20] University of Washington XML Data Repository. http://www.cs.washington.edu/research/xmldatasets/www/repository.html.

[21] D. Gruhl R. Agrawal, R. Bayardo and S. Papdimitriou. Vinci: A service-oriented architecture for rapid development of web applications. In *WWW10, Hongkong*, May 2001.

[22] A. M. Flavio Rizzolo. Indexing xml data with toxin. In *WebDB-2001*, 2001.

[23] N. Sundaresan and R. Moussa. Algorithms and programming models for efficient representation of xml for internet applications. In *Proceedings of the 10th International World Wide Web Conference*, pages 366–375, 2001.

[24] Microsoft Message Queuing (MSMQ) technology. http://www.microsoft.com/windowsserver2003/technologies/msmq/default.mspx.

[25] Pankaj M. Tolani and Jayant R. Haritsa. Xgrind: A query-friendly xml compressor. In *18th International Conference on Data Engineering (ICDE'02)*, 2002.

[26] Yannis Viniotis. Probability and random processes for electrical engineers. pages 258, 350–352, 1998.

[27] P. Wood W. Lam, W. Ng and M. Levene. Xcq: Xml compression and querying system. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[28] The World Wide Web Consortium. Extensible Markup Language (XML). http://www.w3.org/xml.

Chapter 7

Appendix

7.1 XML Generator

Various XML generators were examined to provide the input to our software. However none of the XML Generators provided the flexibility of making the XML based on certain attributes like size, depth etc. Hence a XML Generator had to be designed and developed to provide an input set to our software.

The XML generator developed generates the types of XMLs most commonly used in transfer of data across the middleware. The structure of the XMLs generated was based on the xml files available at the University of Washington XML Data Repository [20]. Our XML generator used the same structure as the files available in the repository.

The XMLs are generated based on four attributes size, depth, number of distinct tags, number of elements. They are provided as input to the XML Generator. The XML Generator first determines the distribution of the distinct tags according to the depth of the XML. The distribution is done randomly with the condition that each depth has at least one distinct tag assigned to it. Then the number of repetitions of the first element under the root element is determined based on number of distinct tags and the number of elements available.

Then the first node under the root element is generated. A node has how many children, is determined randomly for each node at each depth till the number of distinct tags available at that depth is exhausted. Each node and its child has distinct tags assigned to them. The number of end nodes (which can have text under them) are counted while generating the first node.

After the creation of first node, the size of the text to be added at each end node is determined by taking into consideration the size of the first node without text, the number of such nodes and the number of end nodes. The text to be added in the XML is read from an input file containing content copied from news sites and encyclopedia available on the internet.

The first node is cloned and added to the document till the number of repetitions calculated earlier are satisfied. The text is then added to the end nodes of the whole XML to complete the whole document. The generated XML is then serialized to a byte buffer and returned. The algorithm is as shown in Algorithm 9 and 10.

Algorithm 9 XML Generator

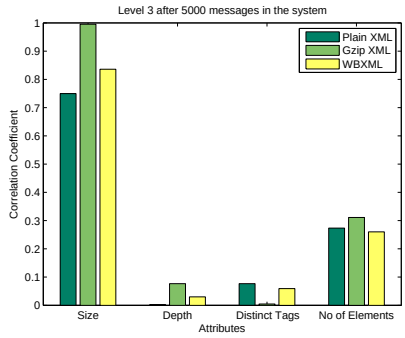
{*Input* : Size i_size, Depth i_depth, Distinct Tags i_disttags, Number of Elements i_noelements}
{*Output* : The XML generated in a byte buffer b_xml}
i_disttagsleft ← i_disttags
for i_count ← 1 to i_depth **do**
  d_random ← random number between 0 and 1
  data structure containing the distribution for each depth ds_distribution ← 1 + (i_disttagsleft (i_depth-i_count)) * d_random
  i_disttagsleft ← i_disttagsleft – ds_distribution
end for
i_repetition ← i_noelements / i_disttags
d_root ← new root element of a new dom d_xml
r_firstnode ← return value of recursive algorithm to create a node with ds_distribution, current depth 1, depth of xml i_depth as input
r_firstnode ← update the n_clone with text values
append n_firstnode to d_root
for i_count ← 1 to i_repetition – 1 **do**
  r_clone ← clone of r_firstnode
  r_clone ← update the r_clone with text values
  append r_clone to d_root
end for
b_xml ← serialization of d_xml
return b_xml

Algorithm 10 Recursive algorithm to create a node

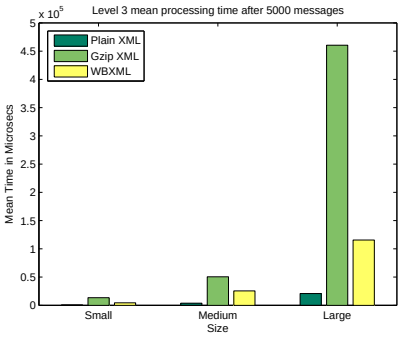
```
{Input : Data structure ds_distribution containing the distribution of distinct tags at a
particular depth, current depth i_currdepth, depth of xml i_depth}
{Output : node r_node}
r_node ← new node
d_random ← Random number between 0 and 1
i_elem ← d_random * value in ds_distribution corresponding to the current depth
ds_distribution ← ds_distribution - i_elem for this depth
if i_elem = 0 then
  its a end node. set text value
end if
for i_count ← 1 to i_elem do
  if i_currdepth <> i_depth then
    n_child ← return value of recursive algorithm to create a node with ds_distribution
    and i_currdepth + 1 as input
    append r_child to r_node
  else
    its a end node. set text value
  end if
end for
return r_node
```

7.2 Supplementary Graphs For Test Scenario

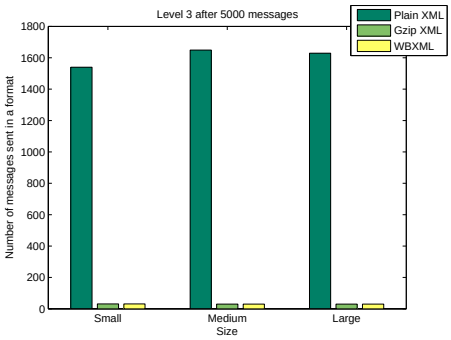
7.2.1 Non Adaptive



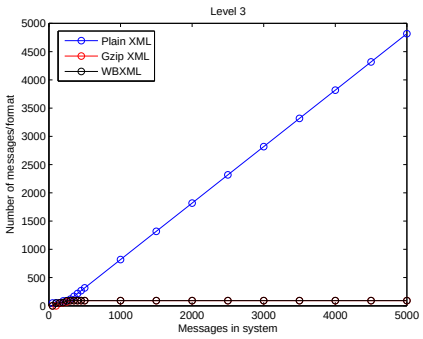
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

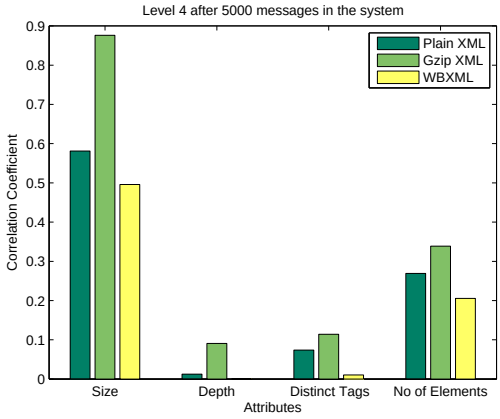


(c) Number of messages for each format as per the size range

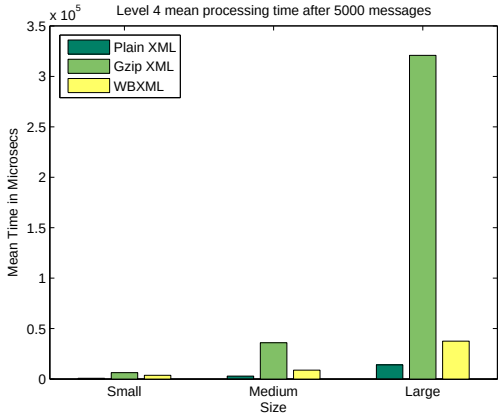


(d) Number of messages in the system for each format

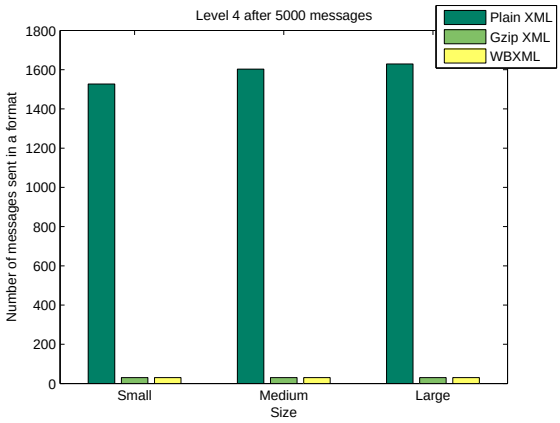
Figure 7.1: Level 3 - Test Scenario - Non Adaptive



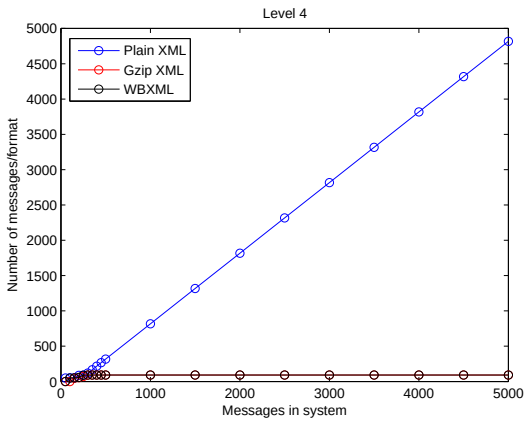
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute



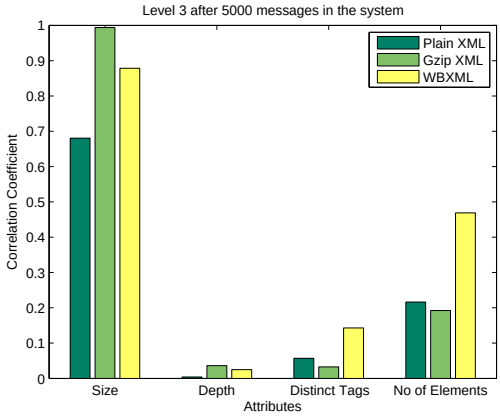
(c) Number of messages for each format as per the size range



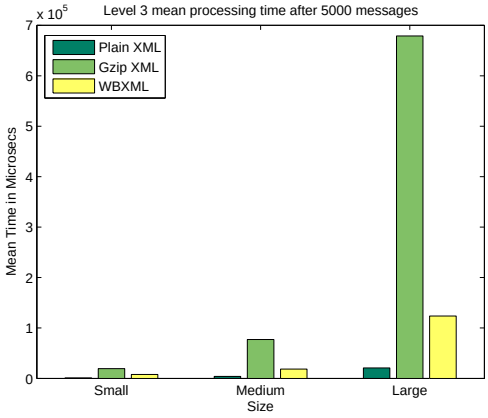
(d) Number of messages in the system for each format

Figure 7.2: Level 4 - Test Scenario - Non Adaptive

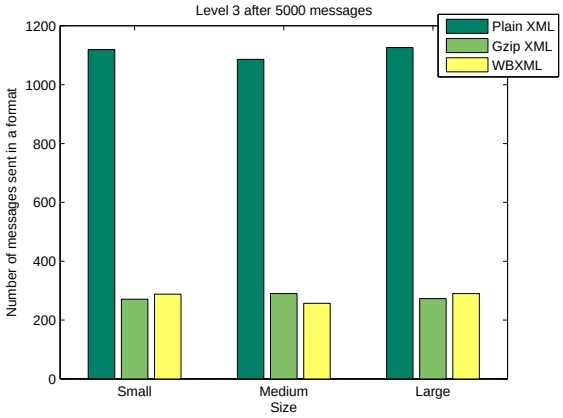
7.2.2 Adaptive Scheme 1



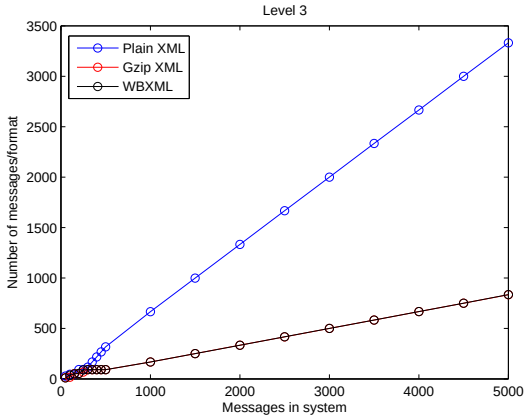
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

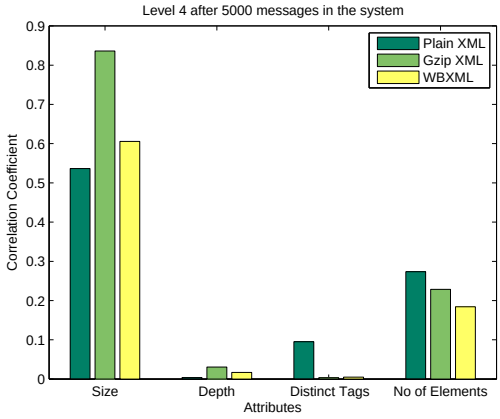


(c) Number of messages for each format as per the size range

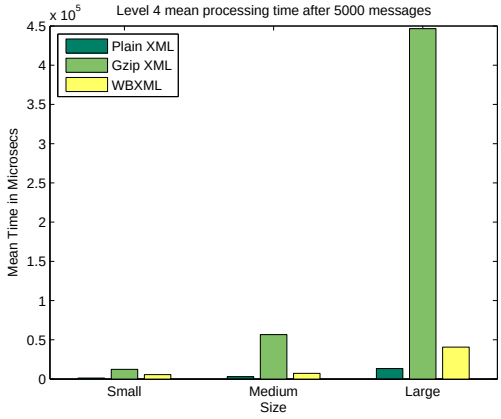


(d) Number of messages in the system for each format

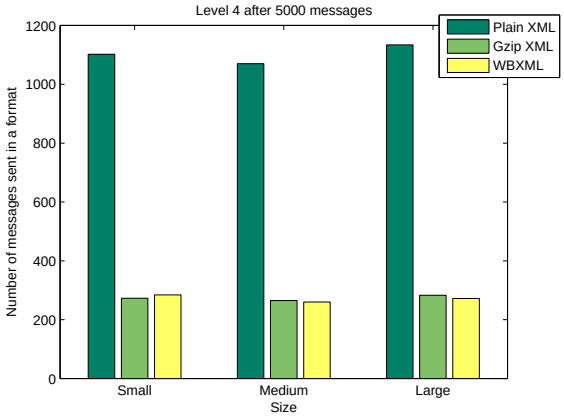
Figure 7.3: Level 3 - Test Scenario - Adaptive Scheme 1



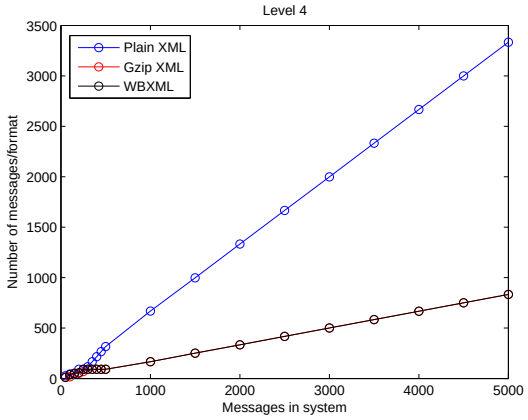
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute



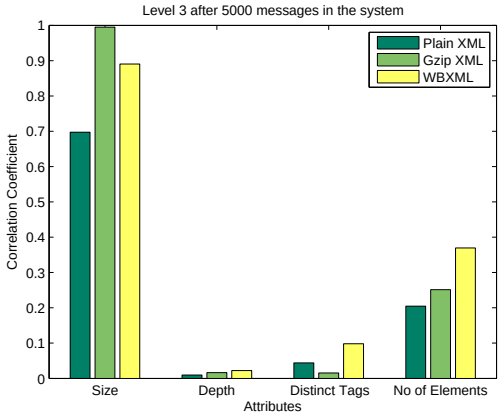
(c) Number of messages for each format as per the size range



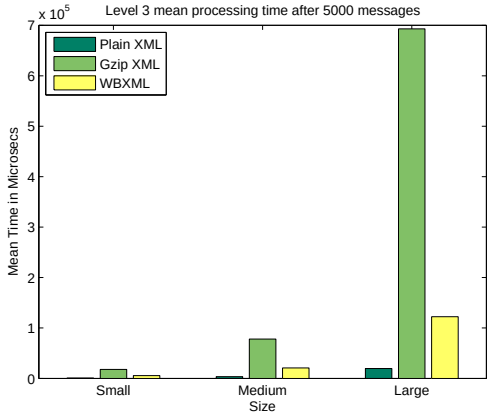
(d) Number of messages in the system for each format

Figure 7.4: Level 4 - Test Scenario - Adaptive Scheme 1

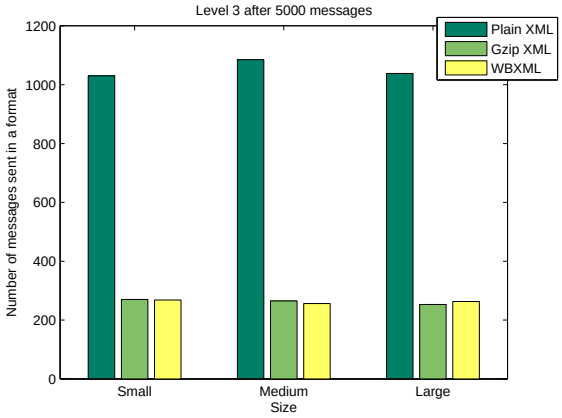
7.2.3 Adaptive Scheme 2



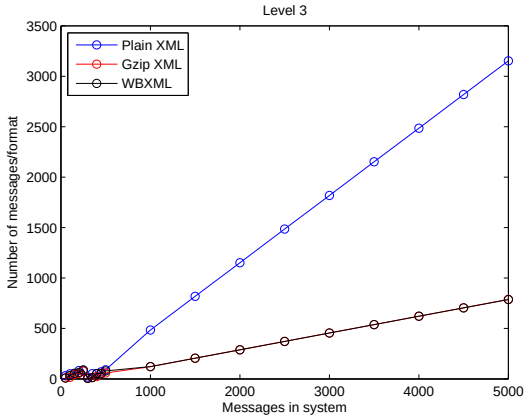
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute

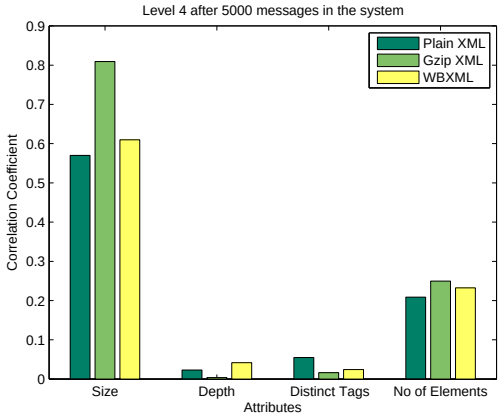


(c) Number of messages for each format as per the size range

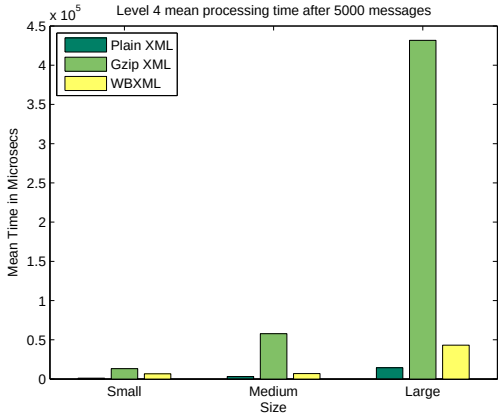


(d) Number of messages in the system for each format

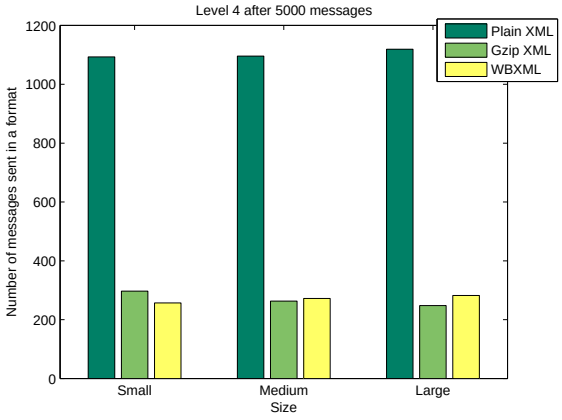
Figure 7.5: Level 3 - Test Scenario - Adaptive Scheme 2



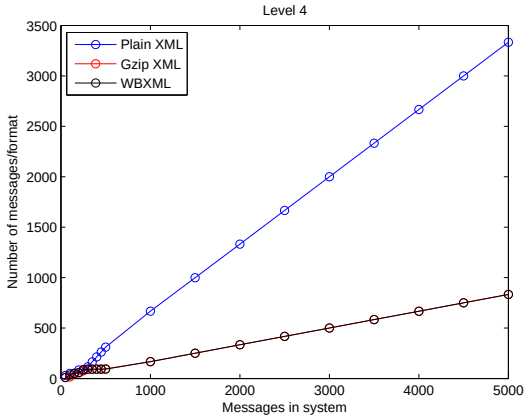
(a) Correlation coefficients



(b) Mean processing time for the three ranges of size attribute



(c) Number of messages for each format as per the size range



(d) Number of messages in the system for each format

Figure 7.6: Level 4 - Test Scenario - Adaptive Scheme 2

7.3 Data Tables For Test Scenario

The data for the graphs presented in chapter 5 was obtained from the Correlation and Attribute Tables at each components in the test scenario. The following sections lists the two tables and their values for all the three schemes.

7.3.1 Non Adaptive

Table 7.1: Correlation Table - Level 1 - Non Adaptive

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.89	3308
1	Depth	0	0.02	3308
1	DistinctTags	0	0.05	3308
1	NoofElements	0	0.1	3308
1	Size	1	0.99	91
1	Depth	1	0.06	91
1	DistinctTags	1	0.01	91
1	NoofElements	1	0.32	91
1	Size	2	0.91	1601
1	Depth	2	0.02	1601
1	DistinctTags	2	0.01	1601
1	NoofElements	2	0.35	1601

Table 7.2: Attribute Table - Level 1 - Non Adaptive

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	30	10416	31	11088	1541	8517
35000	65000	1649	16544	30	60937	30	43750
350000	650000	1629	111513	30	558333	30	240625

Table 7.3: Correlation Table - Level 2 - Non Adaptive

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.77	4818
1	Depth	0	0.03	4818
1	DistinctTags	0	0	4818
1	NoofElements	0	0.17	4818
1	Size	1	0.99	91
1	Depth	1	0.07	91
1	DistinctTags	1	-0.01	91
1	NoofElements	1	0.3	91
1	Size	2	0.91	91
1	Depth	2	0.12	91
1	DistinctTags	2	-0.12	91
1	NoofElements	2	0.11	91

Table 7.4: Attribute Table - Level 2 - Non Adaptive

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1540	872	31	9072	31	4032
35000	65000	1649	2349	30	48958	30	25520
350000	650000	1629	22828	30	459375	30	152083

Table 7.5: Correlation Table - Level 3 - Non Adaptive

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.75	4818
1	Depth	0	0	4818
1	DistinctTags	0	0.08	4818
1	NoofElements	0	0.27	4818
1	Size	1	1	91
1	Depth	1	0.08	91
1	DistinctTags	1	0	91
1	NoofElements	1	0.31	91
1	Size	2	0.84	91
1	Depth	2	0.03	91
1	DistinctTags	2	-0.06	91
1	NoofElements	2	0.26	91

Table 7.6: Attribute Table - Level 3 - Non Adaptive

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1540	710	31	13608	31	4536
35000	65000	1649	3742	30	50520	30	25520
350000	650000	1629	20804	30	460416	30	115625

Table 7.7: Correlation Table - Level 4 - Non Adaptive

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.58	4817
1	Depth	0	0.01	4817
1	DistinctTags	0	0.07	4817
1	NoofElements	0	0.27	4817
1	Size	1	0.88	92
1	Depth	1	0.09	92
1	DistinctTags	1	0.11	92
1	NoofElements	1	0.34	92
1	Size	2	0.5	91
1	Depth	2	-5.00E-004	91
1	DistinctTags	2	0.01	91
1	NoofElements	2	0.21	91

Table 7.8: Attribute Table - Level 4 - Non Adaptive

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1527	736	30	6250	30	3645
35000	65000	1603	2904	30	35937	30	8854
350000	650000	1629	14080	30	320833	30	37500

Table 7.9: Correlation Table - Level 5 - Non Adaptive

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.53	1415
1	Depth	0	-0.02	1415
1	DistinctTags	0	0.15	1415
1	NoofElements	0	0.27	1415
1	Size	1	0.76	2181
1	Depth	1	-0.01	2181
1	DistinctTags	1	0.04	2181
1	NoofElements	1	0.18	2181
1	Size	2	0.43	1404
1	Depth	2	0.01	1404
1	DistinctTags	2	0.03	1404
1	NoofElements	2	0.26	1404

Table 7.10: Attribute Table - Level 5 - Non Adaptive

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	31	6552	2121	3712	30	5208
35000	65000	1354	5527	30	83333	30	23958
350000	650000	30	60416	30	663020	1344	55884

7.3.2 Adaptive Scheme 1

Table 7.11: Correlation Table - Level 1 - Adaptive Scheme 1

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.89	2821
1	Depth	0	0	2821
1	DistinctTags	0	0.09	2821
1	NoofElements	0	0.13	2821
1	Size	1	0.99	705
1	Depth	1	0	705
1	DistinctTags	1	0.02	705
1	NoofElements	1	0.25	705
1	Size	2	0.91	1474
1	Depth	2	-0.02	1474
1	DistinctTags	2	0.06	1474
1	NoofElements	2	0.48	1474

Table 7.12: Attribute Table - Level 1 - Adaptive Scheme 1

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	33	14204	231	19751	1414	8972
35000	65000	1361	16796	242	84129	30	44270
350000	650000	1427	109528	232	785088	30	253125

Table 7.13: Correlation Table - Level 2 - Adaptive Scheme 1

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.7	3333
1	Depth	0	-0.01	3333
1	DistinctTags	0	0.01	3333
1	NoofElements	0	0.17	3333
1	Size	1	0.99	834
1	Depth	1	0.01	834
1	DistinctTags	1	-0.03	834
1	NoofElements	1	0.22	834
1	Size	2	0.94	833
1	Depth	2	-0.03	833
1	DistinctTags	2	0.1	833
1	NoofElements	2	0.26	833

Table 7.14: Attribute Table - Level 2 - Adaptive Scheme 1

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1123	1043	289	12489	266	4170
35000	65000	1083	2553	283	70395	267	21769
350000	650000	1127	21545	262	687738	300	152656

Table 7.15: Correlation Table - Level 3 - Adaptive Scheme 1

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.68	3331
1	Depth	0	0	3331
1	DistinctTags	0	0.06	3331
1	NoofElements	0	0.22	3331
1	Size	1	0.99	834
1	Depth	1	-0.04	834
1	DistinctTags	1	0.03	834
1	NoofElements	1	0.19	834
1	Size	2	0.88	835
1	Depth	2	-0.02	835
1	DistinctTags	2	0.14	835
1	NoofElements	2	0.47	835

Table 7.16: Attribute Table - Level 3 - Adaptive Scheme 1

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1119	1019	271	19372	288	7758
35000	65000	1086	3985	290	77209	257	18543
350000	650000	1126	20800	273	679029	290	123760

Table 7.17: Correlation Table - Level 4 - Adaptive Scheme 1

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.54	3334
1	Depth	0	0	3334
1	DistinctTags	0	0.1	3334
1	NoofElements	0	0.27	3334
1	Size	1	0.84	833
1	Depth	1	-0.03	833
1	DistinctTags	1	0	833
1	NoofElements	1	0.23	833
1	Size	2	0.61	833
1	Depth	2	-0.02	833
1	DistinctTags	2	0	833
1	NoofElements	2	0.18	833

Table 7.18: Attribute Table - Level 4 - Adaptive Scheme 1

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1102	1233	273	12362	284	5666
35000	65000	1070	3037	265	56603	260	7271
350000	650000	1134	13434	283	446499	272	40728

Table 7.19: Correlation Table - Level 5 - Adaptive Scheme 1

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.66	1534
1	Depth	0	0.02	1534
1	DistinctTags	0	0.08	1534
1	NoofElements	0	0.38	1534
1	Size	1	0.67	694
1	Depth	1	0	694
1	DistinctTags	1	0.11	694
1	NoofElements	1	0.41	694
1	Size	2	0.61	2772
1	Depth	2	-0.01	2772
1	DistinctTags	2	0.08	2772
1	NoofElements	2	0.3	2772

Table 7.20: Attribute Table after 1000 messages - Level 5 - Adaptive Scheme 1

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	30	10416	54	9837	392	4583
35000	65000	54	19097	46	105298	153	13480
350000	650000	115	106521	60	853645	96	82194

Table 7.21: Attribute Table after 5000 messages - Level 5 - Adaptive Scheme 1

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1294	2306	284	8912	689	5283
35000	65000	90	18750	190	141858	1063	13611
350000	650000	150	103020	220	862784	1020	87408

7.3.3 Adaptive Scheme 2

Table 7.22: Correlation Table - Level 1 - Adaptive Scheme 2

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.98	2154
1	Depth	0	0.02	2154
1	DistinctTags	0	0.03	2154
1	NoofElements	0	0.25	2154
1	Size	1	1	540
1	Depth	1	0.03	540
1	DistinctTags	1	-0.02	540
1	NoofElements	1	0.16	540
1	Size	2	0.81	1116
1	Depth	2	-8.62E-006	1116
1	DistinctTags	2	0.23	1116
1	NoofElements	2	0.38	1116

Table 7.23: Attribute Table after 1000 messages - Level 1 - Adaptive Scheme 2

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	215	8357	59	20656	51	11948
35000	65000	228	16995	62	87449	60	42187
350000	650000	224	108119	46	785326	55	235511

Table 7.24: Attribute Table after 5000 messages - Level 1 - Adaptive Scheme 2

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1054	8968	184	100883	31	18145
35000	65000	1070	98919	180	672656	31	118447
350000	650000	30	875000	176	6398615	1054	769879

Table 7.25: Correlation Table - Level 2 - Adaptive Scheme 2

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.71	2492
1	Depth	0	0	2492
1	DistinctTags	0	0	2492
1	NoofElements	0	0.17	2492
1	Size	1	1	623
1	Depth	1	0.06	623
1	DistinctTags	1	0	623
1	NoofElements	1	0.19	623
1	Size	2	0.96	623
1	Depth	2	0.05	623
1	DistinctTags	2	0.05	623
1	NoofElements	2	0.19	623

Table 7.26: Attribute Table - Level 2 - Adaptive Scheme 2

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	849	754	187	12032	206	3944
35000	65000	818	2884	232	71390	212	20636
350000	650000	825	20473	204	665058	205	150381

Table 7.27: Correlation Table - Level 3 - Adaptive Scheme 2

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.7	3153
1	Depth	0	0.01	3153
1	DistinctTags	0	0.04	3153
1	NoofElements	0	0.2	3153
1	Size	1	0.99	788
1	Depth	1	0.02	788
1	DistinctTags	1	0.02	788
1	NoofElements	1	0.25	788
1	Size	2	0.89	787
1	Depth	2	0.02	787
1	DistinctTags	2	0.1	787
1	NoofElements	2	0.37	787

Table 7.28: Attribute Table - Level 3 - Adaptive Scheme 2

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1030	879	270	17939	268	5538
35000	65000	1085	3629	265	78066	256	20751
350000	650000	1038	19764	253	693120	263	122326

Table 7.29: Correlation Table - Level 4 - Adaptive Scheme 2

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.57	3334
1	Depth	0	0.02	3334
1	DistinctTags	0	0.05	3334
1	NoofElements	0	0.21	3334
1	Size	1	0.81	833
1	Depth	1	0	833
1	DistinctTags	1	0.02	833
1	NoofElements	1	0.25	833
1	Size	2	0.61	833
1	Depth	2	0.04	833
1	DistinctTags	2	0.02	833
1	NoofElements	2	0.23	833

Table 7.30: Attribute Table - Level 4 - Adaptive Scheme 2

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1093	1157	297	13152	257	6687
35000	65000	1096	3079	263	57806	272	7008
350000	650000	1119	14368	248	431640	282	43218

Table 7.31: Correlation Table - Level 5 - Adaptive Scheme 2

Component No	Attribute	Format Type	Correlation Coefficient	N
1	Size	0	0.7	1967
1	Depth	0	9.16E-004	1967
1	DistinctTags	0	-1.70E-004	1967
1	NoofElements	0	0.13	1967
1	Size	1	0.4	607
1	Depth	1	0.01	607
1	DistinctTags	1	0.04	607
1	NoofElements	1	0.23	607
1	Size	2	0.44	2426
1	Depth	2	-0.03	2426
1	DistinctTags	2	0.1	2426
1	NoofElements	2	0.26	2426

Table 7.32: Attribute Table - Level 5 - Adaptive Scheme 2

Size Range Min	Size Range Max	N (Plain XML)	Mean (Plain XML)	N (GZip XML)	Mean (GZip XML)	N (WBXML)	Mean (WBXML)
0	15000	1907	2449	276	4528	30	6250
35000	65000	30	22395	159	59355	1212	9991
350000	650000	30	130208	172	304960	1184	39933