CHAPTER 1

INTRODUCTION

1.1 WEB DATABASES

XML (eXtensible Markup Language), is a markup Language developed by the World Wide Web Consortium (W3C) that supports a richer set of features for effective representation of documents including user-defined tags that allow both data and descriptive information about data to be represented within a single document (Bradley 2000). It can be used mainly for two purposes namely as markup language and as an ideal structure for storing semi-structured data. Moreover, this XML language (Bray et al 2008) by virtue of its self-describing and textual nature has become extremely popular as a medium of data exchange and storage (Jiang et al 2002). As XML become a standard for data exchange (Natchetoï et al 2007) and communication over the Internet, the concept of XML databases and its manipulation including storing, retrieving and querying (Bogdan et al 2011, Rao and Moon 2004) XML document have become very significant.

1.2 ISSUES IN XML DATABASES

Web databases represented using XML documents are usually large in size as they often contain much redundant data such as repeated tags. Moreover, exporting data from proprietary formats to XML also increases its volume significantly. For example, Liefke and Suciu (2000) showed that specific format data such as weblog data and swissprot data when converted
to XML format had increased the storage size by about 40%. This type of increase in storage prevents the use of XML for storage and retrieval of web data as it substantially increases the cost of query processing, data storage and data exchanges over the web. Therefore, it is necessary to solve this problem by proposing effective storage and retrieval techniques for XML databases.

1.3 AIMS AND OBJECTIVES OF THE THESIS

The aim of this research work is to provide a compact storage structure for XML databases by developing a space efficient representation for the data. In this thesis, two different queriable compact storage structures for XML have been proposed, to overcome the existing storage problems. Among them, the first storage structure named RFX (Redundancy Free XML storage structure) is a non tree based storage structure, with a multilayered architecture, where the elements and data are stored as a separate layers and this facilitates the navigation and retrieval of data effectively. The second storage structure called QUICX (Query and Update support for Indexed and Compressed XML) is a compact storage structure that supports query and update efficiently. It minimizes the original XML file size considerable by usage of containers.

Objectives:

1. To propose new queriable compact storage structures for effective representation of XML data.
2. To reduce the storage space for XML data and processing time for XML queries.
3. To propose new compressors for data compression in XML databases.
4. To propose new query processing techniques for XML databases to reduce the processing time.

5. To propose new indexing techniques to further reduce the cost of query processing.

The main motivation for this research work is to support the storage and retrieval of XML data effectively using compression techniques in web databases.

1.4 REPRESENTATION TECHNIQUES FOR STORING XML DATA

Currently, two approaches are used normally to represent the XML tree space efficiently namely pointer-based and succinct approaches. In the pointer-based approach (Arion et al 2003), the XML tree is represented using pointers, which in turn increases the size of the XML storage. On the other hand in succinct approach (Munro et al 2001, Raman and Rao 2003, Zhang et al 2004, Lam et al 2007, Arroyuelo 2008), the tree is encoded as a bit array. Since succinct approach reduces the storage requirements, this method is the preferred structure when the metric is storage space. However, when performance is the major metric, the pointer-based approach is preferred because of the better performance provided by the pointer method. In this thesis, a succinct approach based, non tree structure is proposed for effective representation of XML documents, which is a unique storage structure that can accommodate both hierarchical and relational databases in a single structure. Moreover, this proposed structure is largely effective in saving storage space and thus achieves increase in compaction. Finally, this proposed storage structure maintains low access costs for all the desired primitive operations in data processing. The main advantage of this proposed method is its ability to store and retrieve large volume of XML data effectively.
1.5 STORING THE XML DATA

XML database management systems can be classified into XML-Enabled databases and Native XML databases. XML-Enabled databases such as conventional relational (Kappel et al 2004, Tatarinov et al 2002, Florescu and Kossmann 1999, Shanmugasundaram et al 1999), Object-Relational (Dobbie et al 2001, Klettke and Meyer 2000, Shimura et al 1999) and Object-Oriented databases (Yang and Rishe 2004, Martinez et al 2003) map XML data into XML Enabled database and XML queries into SQL (Structured Query Language) Queries. During this mapping, a heavy cost is incurred and some semantics are also lost. A Native database (Fiebig et al 2002) means that the database can store XML data itself in the database or it can be exported to the XML database, after making minimal changes. The native XML based data management (Meng et al 2003 and Jagadish et al 2002), is more feasible than the approaches that map XML onto other data models that suffer from severe drawbacks. In this thesis, an efficient, native database approach has been proposed for storing, querying and managing XML data. In this approach, an XML database management system has been designed and implemented from ground up. Therefore, in this proposed model, the XML data is stored directly, retaining its natural tree relationships between the elements and at the same time provides all the benefits of relational database management systems. The major advantages of this proposed native database approach are the provision of fast navigational operations, efficient insertions and deletions, support of efficient join operations and extra indexes.

1.6 XML COMPRESSOR

The main problem with the storage of XML documents is the huge document sizes made by the storage structures. Moreover, XML usage is growing constantly and since huge repositories of XML documents are currently pervasive there exists a great demand for efficient XML compressions techniques.
The use of compression techniques for data compression has many advantages such as reducing the disk space required for storage, reduction in the usage of network bandwidth required for data exchange, and minimization of the main memory requirements for processing and querying XML documents (Wang et al 2010). XML Compressors are classified with respect to two main characteristics based on their awareness of the structure of the XML documents and on their ability for supporting queries.

### 1.6.1 Structure Awareness Based Compressors

Based on the awareness of the structure of the XML documents, XML compressors can be divided into two main groups namely General text compressor and XML conscious compressor.

#### 1.6.1.1 General Text Compressor

In general text compressors, XML data are stored as text files. The text compressor XML-Blind (Cleary and Witten 1984), treats the documents as a plain text, and applies the traditional text compression techniques for data compression (David Solomon 2004). However, the existing text compressors such as Gzip (Deutsch1996) are not able to discover and utilize the redundancy present in the structure of XML, and therefore often yield only suboptimal results. As a consequence, the compressed XML documents can still remain larger than equivalent text.

#### 1.6.1.2 XML Conscious Compressor

The XML-conscious compressor has awareness about the XML document structure and hence achieves better compression ratios over the general text compressors. They are further classified into schema dependent and schema independent compressors. In schema dependent compressor, both the compressor and de-compressor must have access to the document schema
information in order to participate in the compression process (Girardot and Sundaresan 2000). In the schema independent compressor, the schema information is not required during the process of compression/decompression (Cheney et al 2000). In XMill, an efficient compressor for XML (Liefke and Suciu 2000) which is a schema independent compression scheme, both the structural and data parts of the XML document are collected and compressed separately. In XMill, the structure part, XML tags and attributes are encoded in a dictionary based fashion before passing it to a back-end for performing general text compression. The data values are grouped into homogenous and semantically related containers according to their path and data types. Each container is then compressed separately using a specialized compressor that is ideal for the data type of these containers.

In this thesis, a XML Conscious compressor has been proposed with schema Independent properties, since there is no guarantee that the schema information of the XML document is always available.

### 1.6.2 Query Support and Classification

The Second Classification of the XML databases is based on their ability for supporting queries. The Compressor that comes under this category is Non-Queriable XML Compressors and Queriable XML Compressors, (Wu et al 2009, Yang et al 2005 and Xu and Papakonstantinou 2005). In the Non-Queriable XML Compressor, the XML document is compressed effectively and hence it shows a good compression performance. However, it does not support querying over compressed data. Examples of compressor following this method are XMILL (Liefke and Suciu 2000), XML-conscious Prediction by Partial Matching compression scheme (XMLPPM) (Cheney 2000), Millau (Girardot and Sundaresan 2000) and Structure Compression Algorithm (SCA) by Levene, and Wood (2008). The queriable compressors are able to generate queriable compressed XML documents that support querying over compressed XML data. Some examples of the queriable
compressors are XGrind (Tolani and Haritsa 2002), Xpress (Min et al 2003), XQuec (Arion et al 2004) and XQZip (Cheng and Ng 2004).

1.6.2.1 Non-Queriable XML Compressors

The Non-Queriable XML Compressors do not allow queries to be processed over the compressed domain (Liefke and Suciu 2000, Cheney 2000). These compressors are classified as schema independent and schema dependent compression schemes.

There are many schema independent compressors that are found in the literature. Among them XMILL (Liefke and Suciu 2000) is the first implementation of XML conscious compressor where the structure is separated from data and grouping the data values into homogenous containers has been performed based on their relative paths in the tree where each container is compressed separately using specialized compressor. The Word Replacing Transform for eXtended Markup Language compressor (XWRT) (Skibinski and Swacha 2007a), uses a dictionary-based compression technique which replaces the frequently appearing words with reference to the dictionary obtained by a preliminary pass over the data. XWRT passes the encoded results to three alternative compression schemes for further processing. XMLPPM (Cheney 2000) a streaming XML compressor uses a multiplexed hierarchical PPM(Prediction by Partial Match) Model (MHM), and is considered as an adaptation of the general purpose PPM compression scheme (Cleary and Witten1984). The compression ratio of this model is relatively longer than other non queriable compressors. AXECHOP (Leighton et al 2005b), a grammar based compressor for XML uses Multilevel Pattern Matching (MPM) algorithm. Kieffer et al (2000) proposed a compression algorithm to generate a context-free grammar to represent the structure information and Burrows-Wheeler Transform (Burrows and Wheeler 1994) is applied to the contents to generate the compressed file.
Millau (Girardot and Sundaresan 2000) is the first XML schema dependant compressor that uses Differential Document Type Definition (DTD) Tree compression (DDT) techniques. However this technique results into huge memory consumption. In addition, when it is used for compressing large XML documents, it leads to the use of large amount of virtual memory that causes frequent thrashing of disk I/O. XML compression with AUtomata and a STack (XAUST) (Subramanian and Shankar 2005) converts the schema information of the DTD into a set of Deterministic Finite Automata, one for each element in the DTD. Then each element container is compressed incrementally using an arithmetic order-4 compressor (Witten et al 1987). The Structure Compression Algorithm (SCA) proposed by Levene and Wood (2008) uses DDT techniques and handles only valid XML documents. SCA creates a parse tree equivalent to DOM tree representations, uses breadth first traversal to navigate the parse tree, explore for the structural information. Due to usage of DDT technology, the memory consumption is large in SCA algorithm.

1.6.2.2 Queriable XML Compressors

Queriable XML compressors (Adiego et al 2009) are those that allow queries to be processed over the compressed formats. There are two types of Queriable XML compressors namely homomorphic and non-homomorphic Queriable XML compressors.

In Homomorphic queriable compressors, the original structure of the XML document is retained and the compressed format can be accessed and parsed in the same way that is used to access and parse the original format. The first XML-conscious compression technique called XGrind proposed by Haritsa and Tolani (2002), supports querying without the need for a full decompression of the compressed XML document. Since it is homomorphic in nature, it cannot perform any join, aggregation and nested queries. Another homomorphic compressor called Xpress (Min et al 2006)
adopts homomorphic transformation strategy to transform an XML document into a compressed form, which preserves the syntactic and semantic information of the original XML document. However, the compression ratio and compression time of XMill is better than that of XPress. Query-supporting XML Transform (QXT) proposed by Skibinski and Swacha (2007b), processes queries with partial decompression where the query execution starts with reading the dictionary from the compressed file and then the query processor resolves which container contains data matching the query. The required containers are decompressed and only the matching elements are decoded to the original XML form.

In Non-homomorphic queriable compressors, the encoding process of the XML document separates the structural part from the data part. Therefore, the structure of the compressed format is different from the structure of the original document (Arion et al 2007, Skibinski and Swacha 2007a, Leighton et al 2005a). XQzip (Cheng and Ng 2004) removes the duplicate structures in an XML document, separates the data values into sequence of blocks and applies gzip compression to each block. For querying, the XQzip tries to determine minimum number of blocks to be decompressed. It supports queries with multiple and nested predicates having mixed values and aggregations. However, XQzip does not support complex queries such as join queries and order based predicates. In tree based queriable compressor for XML, TREECHOP (Leighton et al 2005a), the parsed tokens of XML documents are written to a compression stream in depth-first order. In TREECHOP, exact-match and range queries are carried out with single scan through the compression stream. The XML Compression and Querying system (XCQ) compressor proposed by Wilfred et al (2006) exploits the information from the DTD associated with XML documents to achieve better compression as well as effective querying. The indexed Partitioned Path-Based Grouping (PPG) data streams support the XML query process that
involves aggregation and selection, without the need for full decompression. The limitations of XCQ are that it compresses only valid documents and it requires longer compression and decompression time. Bramandia et al (2009) proposed a non-homomorphic compressor in which the XML document is updated and the source relational database is also updated accordingly. XQuec (Arion et al 2007) separates the structure and content within an XML document. In order to perform the evaluation of a predicate within the compressed domain, it ensures that containers involved in the predicate belong to the same group and are compressed. XQueC supports a larger subset of XML query types than XGrind and XPRESS and achieves significant improvement in query performance than the two latter ones. However, XQueC makes insufficient use of commonality of XML data. Therefore, it achieves less compression ratio than XMill. Zhang et al (2004) provided a succinct approach using balanced parenthesis encoding to store blocks of data.

In this thesis, two different compressors namely RFX and QUICX have been proposed, which are Non-homomorphic queriable XML compressors. Similar to balanced parenthesis encoding, a new approach in RFX has been used in this work to encode the order of the XML document rather than nesting information, called as order encoding. The nesting information of the XML document is stored in the RFX tag layer. In QUICX, the both order and the nesting information are stored in the Meta data table. Thus in both approaches, XML is stored using succinct representation, thereby reducing the storage size with query support. The queriable approach supports direct node navigation operation, simple, complex, nested queries and also supports efficient node insertion and deletion mechanisms.
1.7 XML QUERY LANGUAGES

XPath (Clark and DeRose 1999) is a language that allows specific nodes or values to be selected from a source XML document for manipulation and presentation. An XPath query navigates within a XML document and retrieves information and is a language for selecting nodes from a XML documents. XPath 1.0 is widely implemented and used, either on its own (called via an API from languages such as Java , C# or JavaScript), or embedded in languages such as XSLT (Clark 1999) or XForms (Boyer 2009 ) was recommended. XPath 2.0 (Berglund et al 2010) was recommended in 2010. A number of implementations exist but are not widely used as XPath 1.0.

XPath selects nodes from given XML documents so that the selected path satisfies a specified pattern. The most simple form of an XPath query contains a sequence of tags alternated by axes. The XML queries are primarily divided into two main categories namely path queries and twig queries (Zhu et al 2008). Path queries contain simple path expressions, i.e. child axis “/” and descendant axis “//”. An example of a path query is “book/title”, that returns all books' titles. The twig query is complex than its counterpart. It contains selection predicates in the path expressions. One such example is /Publisher [@identity='Pearson'] //title” which finds the titles of all books, published by “Pearson”. Generally, there can be more than one predicate in a twig query (Zhu et al 2008). Classification of XPath queries (Balmin et al 2008) provides valuable scope for testing the performance of query processor in different workloads.

XQuery (Boag et al 2011) is also a well-known XML query language, which is a superset of XPATH. It is a functional language, comprised of the clauses ‘for’, ‘let’, ‘where’, and ‘return’. These clauses can
be nested or composed. For example, a simple XQuery to find the titles of all the books that costs more than $40 can given by:

\[
\text{for } z \text{ in } /\text{inventory/book} \\
\text{where } z/\text{cost}>40
\]

In this thesis, the XPath query has been used to query the proposed compact storage structure and a list of XPath functionalities have been selected for the implementation of this research work.

1.8 QUERY PROCESSING TECHNIQUES FOR COMPACT STORAGE STRUCTURE

During the past two decades, a lot of research works have been carried out in query processing (Kling et al 2011, Haw and Lee 2011, Jin and Veerappan 2010, Lee and Haw 2009) for XML storage structure as well as optimizing the XML path queries. The nature of the XML query processing and optimization techniques (Che et al 2006, Katajainen and Makinen 1990) varies based on the format in which the XML data are stored.

For XML data stored as flat files, the query processing is nothing more than the scanning the entire XML document sequentially one element after the other. The performance of such query processing systems is low because the sequential search technique employed in this process goes through elements which are irrelevant to the query. In XGrind (Haritsa and Tolani 2002), the query processing engine consists of a lexical analyzer that emits tokens for encoded tags, attributes and data values. The parser performs the matching and dumping of the matched records and makes a depth-first search traversal of the XML document. Queries can be carried out over the compressed document without fully decompressing it. The exact match and prefix match are carried out directly on compressed document. Moreover, the
range or partial-match queries require on-the-fly decompression of only the element/attributes values that are part of the query predicates. However, XGrind doesn’t support non-equality selection, join, aggregation, nested queries or construct operations. XQZip uses a Structure Index Tree (SIT) that tends to merge sub trees containing the exact set of paths, applies GZIP compression to value blocks, and entails decompression of entire blocks during query evaluation. The blocks have predefined length, say 1000 records each, at query processing. XQZip tries to determine the minimum number of blocks to be decompressed. The query belongs to extended version of XPath, with union and grouping operator. XQuec (Arion et al 2004) provides a random-access query evaluation strategy, as opposed to XPRESS, XGrind and XQZip where the queries are processed as much as possible in compressed domain. Each container tag or an item in the container is compressed. XQZip creates a high partitioned storage and uses path partitioning for selective data access. The XPRESS system is better over XQZip in terms of decompression times but XGrind is relatively lagging behind both Xpress and XQZip in terms of querying.

In this thesis, a data structure called ‘strategy list’ has been proposed, which uses divide and conquer approach to XPath query, to improve the query efficiency without any optimization. The strategy list optimizes the XPath query by breaking it into many independent queries which can be run independently of one another. Thus the strategy list avoids repeated traversal of the XML tree paths and provides a scope for parallel processing.

1.9 DATA INDEXING

Indexing is required on both document structure and the textual values in order to evaluate many queries efficiently and hence the performance would be enhanced. The semi structured nature of XML data and
the flexible mechanism of XML queries introduce challenges to the existing
database indexing methods (Tang et al 2008, Pal et al 2004 and Rizzolo and
Alberto Mendelzon 2001). The common indexing available for XML data are
Path indexing (Min et al 2005, Chen et al 2003 and Cooper et 2001), Node
indexing (Li and Moon 2001, Zhang et al 2001) and Sequence Based indexing (Rao and Moon 2004 and Wang et al 2003). The path indexing
greatly speeds up the evaluation of single path queries but needs expensive
join operation for processing queries with multiple branches. In the node
indexing, each data node is indexed by numbering schemes and the structural
relationship between a pair of nodes can be determined in constant time.
However, it is necessary to rely on the pair-wised comparison to answer a
query in many situations leading to performance degradation. Sequence based
indexing transforms both XML documents and queries into sequences and
evaluates queries based on sequence matching and supports queries without
join operation. Compact tree (Ctree) proposed by Zou et al (2004) provides
indexing structure based on path summary and detailed element level
relationships. Path summary works for single path queries but it is unable to
answer the branching queries. Virtual Suffix Tree proposed by Wang et al
(2003) is a dynamic indexing method that is classified as a sequence-based
indexing based on B+ trees. This Virtual Suffix Tree has structural and value
indexes in a single index. Multidimensional indexing structure proposed by
Jin et al (2007) for muti-attribute retrieval enables indexing of a collection of
string and/or numeric attributes.

XML indexing techniques (Elgedawy et al 2009, Chebotko et al
2007 and Chen et al 2003) can also be built on the compressed document in a
similar manner to those that can be built on regular XML documents. In
XGrind, elements and attribute names are encoded using a dictionary-based
encoding scheme, and character data is compressed using semi-adaptive
Huffman coding. The XCQ (Wilfred et al 2006) compressor generates more
usable compressed data to support querying. In this technique, the compressed XML documents are stored in Partitioned Path-based Grouping (PPG) data streams, which are equipped with a block statistics signature indexing scheme. The indexed PPG data streams support the processing of XML queries that involve selection and aggregation, without the need for full decompression. The limitations of XCQ are that it compresses only valid documents and it requires longer compression and decompression time. XQZip compressor removes the duplicate structures in XML documents to improve query performance by using an indexing structure called the Structure Index Tree. The SIT is constructed based on the partitioning of equivalent paths in an XML document. The data are separated from the structure and compressed into distinct data containers, which are further divided into smaller data blocks that can be compressed and decompressed as an individual unit. XQZip tries to determine the minimum number of blocks to be decompressed during the query evaluation. It also reduces the decompression overhead in query evaluation by applying the least recently used cache algorithm. XQZip does not support complex queries, join queries and order based predicates.

In this thesis, a new compression technique called QUICX has been proposed which is a custom mechanism for indexing in which each container is split into sub containers having only maximum of 50 records. This provides a support for faster update, i.e. rewriting a small file is much faster than rewriting a big file. This also provides a greater retrieval speed for querying. Further, new techniques called Indexing and Querying the Compressed XML data (IQCX) have been proposed for indexing the contents which have occurred in numerous places. Once a container is indexed, the original container is removed, retaining the indexed container, which leads to the deletion of huge containers thereby reducing the memory wasted in storing them and increasing the compression ratio. Thus, it is proved that the indexing technique proposed in this research work greatly reduces the time required to
locate the records of interest which in turn reduces the execution time and increases the querying efficiency.

1.10 CONTENT BASED COMPRESSION

Transmission applications use byte codes to transfer the encoded source. It can be either fixed length code (Girardot and Sundaresan 2000) or a variable length byte-code (Sundaresan and Moussa 2001). The most advanced encoding for transmission application was presented in (Sundaresan and Moussa 2001 and Girardot and Sundaresan 2000). Query enabled encoders separate the structural compression from content compression. XMLzip splits its content according to a certain depth of XML tree structure and uses LZW to compress each sub-tree. XQueC even separates between each path encoding. It uses Huffman coding for encoding the structure and arithmetic coding for encoding the content.

In QUICX proposed in this thesis, the structure and data are separated and the data are compressed using LZW compression. LZW compression is used in this work since it is the best technique for files containing more repetitive data and is not suitable for the file containing more non-repetitive data. Dictionary size occupies more space, if the file contains more non-repetitive data's. During decompression, the regeneration of dictionary is time consuming.

In this thesis, a content based Queryable Compressor is also proposed, where the data in container of QUICX are compressed using Bitmask based compression (Seong and Mishra 2008). The frame format and the decision vary for each type of the content in the container. Query processing time using Lempel-Ziv-Welch (LZW) is more, even though, the intended record is found quickly. It is because the dictionary has been regenerated at the decompression time. Thus the LZW compression is replaced by Bitmask based compression and hence the query is executed in much lesser time.
1.11 THESIS ORGANIZATION

The remainder of this thesis is organized as follows:

**Chapter 2** provides a survey of related works and compares them with the proposed work.

**Chapter 3** describes the architecture of compact storage structure proposed in this work for XML that provides facilities for querying and updating the compact structure.

**Chapter 4** explains the architecture of the RFX Storage structure and explains the various querying techniques for the RFX compact storage structure. This chapter also discusses the results obtained from this work and compares them with the existing works.

**Chapter 5** depicts the architecture of the QUICX system and explains the features provided by QUICX.

**Chapter 6** shows the architecture of the IQCX system proposed in this work and explains the various queries executed on IQCX. It also discusses the results obtained from this work and shows the performance improvements with respect to compression ratio and querying time.

**Chapter 7** discusses the architecture of the Content based compression subsystem for QUICX. It also explains the concept of node indexing and value indexing. Finally, it shows the results obtained from this work and depicts the improvements in compression ratio and querying time over the existing systems.

**Chapter 8** gives the conclusions on this work and presents some possible future enhancements.