CHAPTER 2
LITERATURE SURVEY

2.1 INTRODUCTION

Due to the rapid growth of Web, the usage of XML has exceedingly increased. As a result, more data are produced as XML data and the size of XML data also increases (Augeri et al 2007). In order to manage XML data, various types of researches have been carried out on issues such as XML Storage, XML indexing and XML query processing.

2.2 XML STORAGE

The XML being flexible, self describing and extensible in nature, the structure of XML may vary from a flat regular data-centric structure to a deep irregular document-centric structure. This creates various challenges to the document and database communities as the need for effective storage and query processing (Wichaiwong and Jaruskulchai 2011, Zhang et al 2011, Wang et al 2004) becomes critical. The XML is semi-structured data, there are two main approaches to storing XML documents i.e. using XML-Enabled Databases (XED) such as relational database (Zafari et al 2010, Shanmugasundaram et al 2001a and Zhang et al 2001), object-oriented database (Shimura et al 1999) and using a Native XML Database (NXD). Another storage called hybrid storage has also emerged, in the past which merely combines the XED and NXD solutions and it allows some parts of the structured XML to be mapped into relational data (Shanmugasundaram et al
2001b) while the other parts can be stored in XML data type itself. Further, to effectively manage large sized XML data, the compression on XML data was required. Each of these techniques has advantages and disadvantages, although both techniques are feasible for supporting XML queries.

2.3 COMPACT XML STORAGE

Data compression (Lelewer and Hirschberg 1987, Welch 1984) is a very popular topic in the research literature, and there is a copious amount of work on this subject. It is the process of encoding with a size smaller than that of the original data using a specific encoding method. Data compression (Chung et al 2011) provides several important advantages. First, the storage size for compressed data is reduced compared with that for original data. The second advantage is saving of the network bandwidth. Since the overall size of data is decreases, much more data can be transferred through the network within a given period of time. Lastly, the query performance (Skibinski and Swacha 2007a) can be improved since the memory is efficiently utilized and the required number of disk I/Os is reduced.

2.3.1 General Purpose Compression

According to the ability to recover data, compression methods are classified into two groups, lossy compression and lossless compression (Murthy and Mishra 2009a).

The lossy compression reduces a file by permanently eliminating certain information. The data compressed by the lossy compression method cannot be reconstructed into the original data after decompression. Interesting approaches to lossy compression are wavelet transformation (Sweldens 1996), histograms (Sacca et al 2001 and Vitter et al 1998) and methods for the extraction of significant part from free text (Witten et al 1994).
In the lossless compression/decompression, the original data are eventually restored. Lossless compression schemes refer to the work by Huffman (Knuth et al 1985), the algorithm by Lempel and Ziv (1977), Arroyuelo and Navarro (2005, 2011) and the Arithmetic coding (Lelewer et al 1987). Lossless compression is categorized into three groups: static, semi-adaptive and adaptive (Howard and Vitter 1991). The static compression uses either fixed statistics or do no statistics. The static compressions are dictionary encoding, binary encoding and differential encoding. Semi-adaptive compression scans the input data to gather statistics preliminarily and rescans the data to compress. Huffman encoding (Huffman et al 1952) and arithmetic encoding (Witten et al 1987) are examples of the semi-adaptive compression. Adaptive compression does not require any prior statistics. Instead, statistics are gathered dynamically and updated during the compression phase. Adaptive Huffman encoding, adaptive arithmetic encoding and LZ encoding are the representative’s compression methods.

2.3.2 Unqueriable XML Compression

The Unqueriable compressors are XMILL, XMLPPM, Millau and SCA. XMill (Liefke and Suciu 2000) is the first proposed XML conscious compressor. Both the structural and data value parts of the XML documents are collected and compressed separately. In the structure part, XML tags and attributes are encoded in a dictionary-based fashion before passing it to a back-end general text compressor. XML tags and attributes are encoded in a dictionary-based fashion before passing it to a back-end general text compression scheme. The data values are grouped into homogenous and semantically related containers according to the data type of the container. Each container is then compressed separately using specialized compressor. In XMill, the back-end general purpose compressor are gzip, bzip2 (Seward 2010) and PPM (Luoma and Teuhola 2007, Cleary and Teahan 1997). XMill achieves good compression ratio and does not require DTD to
compress the XML documents, its main drawback is that it needs the decompression of the whole document before querying and hence hinders its wide usage. It also results in a heavy burden on system resources such as CPU processing time and memory consumptions. XMLPPM (Cheney 2000) as a streaming XML compressor uses a Multiplexed Hierarchical PPM Model. It achieves better compression ratio than XMill but takes longer time to compress the document, since PPM is relatively compression technology. Millau the first XML schema dependent compressor, it takes advantages of the schema and data types to enable better compression. It consumes large memory when compressing large XML documents. SCA (Structure Compression Algorithm) aims at compressing the structure of an XML document, it only handles valid XML documents that conform to DTDs. SCA also suffers from same problems as Millau i.e huge memory consumption with the DDT technology.

XMLZip compresses XML documents are represented as DOM trees. It's built on the generic text compressor Gzip. It essentially parses and divides an XML DOM tree at a certain depth into multiple components, and then compresses the components separately. The node level grouping strategy does not improve compression efficiency. The compression ratio of XMLZip and Gzip are very close to each other for all the datasets, but these two compressors are worse than those of XMill and XMLPPM. Advantages of XMLZip is that it allows limited random access to partially decompressed XML documents, since XMLZip supports decompressing the portions of the compressed components that are needed in query evaluation.

Comparing with all these works, the system proposed in this thesis is different in many ways. First, it has been developed for XML that is independent of schema, since there is no guarantee that the schema information of the XML document is always available. Second, the
compressor developed has better compression ratio comparable to the existing compressor. Finally, it supports querying in the compressed modes also.

2.3.3 Queriable XML Compression

Diverse queriable compact storage for XML is XGrind, HuffWord, XPress, XMLZip, XQZip, CXQU (Alkhatib and Scholl 2008) and XQuec etc. XGrind is first Queriable Compressor, though has lower compression ratio than that of XMill and longer compression time. Moreover, it supports certain types of queries and retains the document structure. It requires two scans over the original document, which attributes to low compression speed. But it does not support set based query evaluation such as join queries. HuffWord is a variant of the classical HUFFMAN compressor in which the dictionary consists of the token extracted from the documents that is specialized in texts and uses a word-based alphabet. The authors of the HuffWord (Dvorsky and Martinovic 2007) claim a compression ratio of about 30%. XPress adopts reverse arithmetic encoding which maps the entire path expression to intervals for tags and diverse encoding methods for text according to the data types. The encoding technique enables XPress to achieve better compression ratios and higher query performance than XGrind. However, these methods do not support the evaluation of multi-conditional queries over compressed documents. XMill is in fact has better compression ratio than of XPress and the compression time of XPress is almost twice as long as that of XMill, since it requires parsing the input XML document twice in the compression process.

Tree parent storage (Zhang et al 2010) based on hash table, a dynamic encoding scheme. It has a great advantage in storage space and time of constructing the label path (Thoangi et al 2006). It has much performance in storage consumption and not on querying. TREECHOP XML compressor, the compression process is Simple Application Programming Interface (API) for XML (SAX) based, parsing of the XML document where the parsed
tokens are written out to the compression stream in depth-first order. The codeword is uniquely assigned based on the path of the tree node, it is possible for the decompressor to regenerate the original XML document using the adaptive encoding information incrementally. In TREECHOP, exact-match and range queries can be carried out via a single scan through the compression stream. In “Rank and Select for Succinct Data Structures (Farina et al 2009), the solutions for computing operations is a sequence of bits but however new applications require rank and select to be computed on sequence of bytes instead of bits. QUICX meets the requirement of the new application.

XQueC compresses each data item individually and this usually results in a lower compression ratio compared to XMill. It supports efficient evaluation of XQuery by using variety of structure information and other indexes. However, these structures together with pointers pointing to the individually compressed data items, incur huge space overhead. The query process evaluates XQuery queries over compressed documents. The complete set of operators of XQuery allows for efficient evaluation over the compressed repository.

XQZip supports a wide spectrum of XPath queries on compressed XML data by introducing a structural indexing tree scheme. It achieves a competitive querying time, especially when workload of queries is high and exhibits high degree of locality. But the characteristic of input XML documents may vary greatly, a suitable block size is difficult to find and hence a good balance between the compression ratio and the stability of the query performance may be difficult to maintain for some datasets. The SIT (Structural Indexing Tree) index does not support the evaluation of complex queries such as joins and order based predicates.

In this research work, two different efficient types of compact storage structures namely RFX (Redundancy Free XML Storage) and QUICX
(Query and Update support for Indexed and Compressed XML) have been proposed. RFX is based on layered approach, gathering structural information from XML documents and stores them in such a way that allows quick identification of structural relationships between the nodes. This identification plays a crucial role in efficient XML query processing. The QUICX strategy tries to reduce the size of the XML documents through succinct approach instead of pointer approach. In this method, the XML file stored in RFX can be navigated and queried in near constant time. RFX storage scheme, achieves an optimal balance between the storage as well as query efficiency. This has been possible through Strategy list, which is a data structure that results in a query plan with minimum overhead. Additionally, non-tree based layered structure augments the capability of our system with respect to performance.

The naive representation of the XML documents leads to excessive redundancy. The QUICX provides a good compression ratio by completely eliminating all redundant tags and data in an XML document. It also provides indexing to the data for querying efficiently, it also supports all common XPath Queries, and also supports structural level and data level update. It also formalizes the problem of compressed aware query optimization and presents an extensive experimental evaluation of the approaches proposed. This method shows that they yield considerable performance improvements for XML processing compared to other approaches in this field. In this work four data sets have been used for testing namely Shakespeare, Treebank, DBLP and Swissprot datasets (Miklau 2000).

2.4 INDEXING FOR QUERIABLE COMPRESSORS

Indexing (Arroyuelo et al 2010, Brito et al 2010 and Arroyuelo and Navarro 2005) is key factor in improving the performance of XML queries by reducing the search space. Thus indexing XML data to facilitate query
processing (Suei et al 2009 and Wang et al 2004) has been a popular subject of study in recent years. Most of the previous studies for indexing are path indexing, node indexing and sequence-based indexing for the XML document. Most of them cannot answer branching queries with multi value predicates efficiently. The indexing created by the recent work on XML file further increases the size of the storage and the compression ratio decreases. The goal of this research work is to create indexing that will occupy minimal storage without compromising the precision.

An indexing structure, extended inverted index technique proposed by Ozgur and Gundem (2006), used in information retrieval was proposed for processing queries efficiently. Four types of indexes are defined by them which are stored in relational database. However, more memory space is required by inverted indexing. To overcome this limitation, the data is simply stored in compressed form in the IQCX proposed in this thesis and thus reduces the memory required for storing the XML data.

Feragina and Manzini (2007) proposed two compressed data structures for the full-text indexing problem that support efficient substring searches using roughly the space required for storing the text in compressed form. The first compressed data structure retrieves the occurrences of the pattern within the text. The second compressed data structure provides optimal output-sensitive query time. This second data structure builds upon the first one. However, IQCX indexing technique proposed in this thesis produces an optimal algorithm for querying.

Two techniques have been introduced by Grimsmo (2008) for efficient memory resident path indexes. The first simple approach combines inverted lists, selectivity estimation, hit expansion, brute force search. The second uses suffix trees with additional statistics with multiple entry points into the query. The entry points are partially evaluated in an order based on
estimated cost until one of them is complete. But the brute force technique takes much time in finding the actual data required. In this work, index file is used in locating the records of interest and only those data are decompressed and displayed. A new methodology is proposed by Jin et al (2007) with multidimensional approach for retrieving attributes. The method proposed by them enables indexing of a collection of attributes to facilitate approximate retrieval using edit distance as an approximate match predicate for strings and numeric distance for numeric attributes. Frequently occurring strings are given minimal length code. Moreover, infrequent strings are given comparatively greater length code. Assigning code increases the memory requirement to store it. In IQCX, only the data in the XML file eliminating the tag names are stored and not the extra information about tags. Thus overall memory required to store the data is reduced. YFILTER (Diao et al 2002) aims to provide fast, matching of XML encoded data and transformation of the matched XML data based on specific requirements. Two extended query types are implemented. The first type of query considered in YFILTER is selection query where the predicates can be exact matching (equality comparison) or range matching (inequality comparison). The other type is aggregation type of queries where sum, average types of functions can be used for computation. This system has no compact storage system operation involved.

A Dynamic Relational XML Query Processor (DRXQP) (Alom et al 2010) technique stores the encoded value of element paths, attributes, contents of the element paths and attributes, and XML processing instructions in a dynamic relational structure termed as Multi-XML Data-Structure (MXDS). The encoded values are calculated based on the parent-child relationship. Encoding and query processing becomes complex when the size of the original XML document increase. IQCX does not generate any code from the input data and do not maintain tables with calculated values. Thus memory space as well as time to execute the query gets reduced.
Xlight proposed by Zafari et al (2010) is used to store all types of XML documents and is composed of five tables namely Document, Path, Data, Ancestor, Attribute. Querying becomes complex due to merging of tables for data retrieval. IQCX approach overcomes the problem by storing data in containers, the index file generated is used to retrieve only particular records required.

The means of effectively identifying the type of target node(s) that a keyword query intends to search for is suggested by Bao et al (2010). The types of condition nodes that a keyword query intends to search via is effectively inferred and each query result is ranked. A formula to compute the confidence level of a certain node type to be a desired search for node is designed by them. This formulae computes the confidence of each candidate node type as the desired search via node to model natural human intuitions. The pattern of keywords co occurrence in queries has also been considered in their work. However, some of the irrelevant keywords are also displayed in query answering. However, the IQCX algorithm executes the query after validating it against the Meta table contents. Moreover, IQCX selects only the desired records, decompress and print them and hence irrelevant details are not printed.

Navigation-Free Data Instance (Li et al 2008) is the combination of the runtime generated path tables and token buffers of a given XQuery Q over data stream D. Token Buffers store the offset of nodes related to the query. Considerable amount of memory is consumed in storing the offsets and more time is also needed for calculating them. A compressed index for XML, called XBzipIndex (Ferragina and Manzini 2007) is designed, and the XML document is maintained in a highly compressed format, and both navigation and searching is done uncompressing only a tiny fraction of the data. Compressing and indexing two arrays derived from the XML data are performed for the data retrieval.
XBzipIndex has compression ratio up to 35% which is better than the ones achievable by other tools (Jeuring and Hagg 2002). But then, IQCX indexing techniques, the first one using Meta data and the second one without using Meta data file contents give a better compression ratio compared to the compression ratio from XBzip. XQzip (Cheng and Ng 2004) supports a wide scope of XPath queries such as multiple, deeply nested predicates and aggregation. But IQCX technique proves that executing aggregate queries using the index takes lesser time when compared to executing such queries in XQzip. IQCX also support other queries. TREECHOP (Leighton et al 2005a) is a queriable XML compressor that employs a homomorphic compression scheme and implements a sequential query algorithm and lazy decompression. TREECHOP is not schema aware and implements top-down exact-match and range queries of the compressed document. TREECHOP's compression algorithm is highly efficient and the single pass scheme makes it ideal for data exchange across networks. The compression algorithm used in QUICKX compresses data efficiently and the queries executed using IQCX approach proves that the execution time is less compared to TREECHOP.

Chan et al (2007) proposed an O(n) - bit representation for suffix tree which can avoid pointer, and supports a dynamic test collections and can also be updated but it doesn’t support XPath Queries. XSAQCT (Muldner et al 2009) approach compresses data and stores it and decompression is performed while query processing. Simple queries executed using XSAQCT are compared with IQCX (Indexing and Querying Compressed XML data) technique. IQCX proves to be efficient than XSAQCT and TREECHOP techniques. A new ternary operator is developed, called Kappa-Join (Brantner et al 2006b), for efficiently evaluating queries with existential quantification. A correlation predicate which can occur conjunctively and disjunctively in XML queries are decorrelated efficiently in this work. Oracle accesses every row in the inner table for each and every row in the outer table. This
consumes much time in the case of a nested query and this problem is overcome in IQCX by executing the inner independent query first and then using its output, the outer query is executed. Correlation predicate query which is used in IQCX approach is based on the query given in reference (Brantner et al 2006a).

Thus in IQCX, indexing technique greatly reduces the overall storage size. The index file is later used for evaluating queries, it is also proved that the overall time required to index the file once and use them for evaluating queries is lesser when compared to using the original files without indexing.

2.5 BITMASK BASED COMPRESSION

Many research works have been carried out in the past on compression. However, XMILL (Liefke and Suciu 2000) was the first efficient compressor for XML, proposed to provide an effective storage. However, the main limitation of this compression is that it is not possible to provide queries on this compressed data. In order to facilitate querying over compressed XML, XGRIND (Tolani 2002) and XPRESS are query-friendly XML compressors which support querying over compressed XML data but when compared with XMILL, the compression ratio is less and also the compressing time is more. Murthy and Mishra (2009b) proposed an approach to improve both memory and communication bandwidth in embedded systems, provided a code compression technique using bitmasks without decompression penalty. It is most effective where small dictionaries are used for compression and most of the data patterns are compressed using one or more bitmasks. The Content based compression is designed, considering the dictionary size and the number of bitmasks. Larger dictionary sizes allow more dictionary entries which in turn increases the direct match from the dictionary, thereby increasing the performance of query.
QUICX provides a more compact structure for storing XML and querying XML data with the help of indexing with the data. QUICX consider the similar structure of XML for all record for better compression. First step in QUICX is to split up all features from XML, such as structure schema, metadata for tracking the occurrence of tag and content. For each tag, system maintains each container for data and compresses each container using dictionary based LZW compression. LZW Compression technique is one of the most standard and widely used compression techniques. LZW starts with initializing the dictionary with all the possible single character and corresponding code. Then Scan the input string, until it finds the longest string not in the dictionary. Once it found the longest string, insert it in the dictionary with the corresponding code and enter the code for the string (length one less than longest one) in the compressed code. LZW compression is the best technique for files containing more repetitive data and is not suite for the file containing more nonrepetitive data; dictionary size occupies more space, if the file contains more nonrepetitive data’s and at decompression, time taken for regenerating same dictionary generate at compression phase is time consuming. In proposed compression technique, this inefficiency is eliminated by recording minimum difference between the original and dictionary content in the compressed file. Further querying, efficiency over the compressed content is increased by keeping the semantic information about XML file after compression by applying node and content indexing.

Thus QUICX provides a flexible and compact framework for storage, retrieval and update mechanism of XML file dynamically. LZW Compression in QUICX system support better compression for the file containing more number of repeated words, but it fails to provide for the file containing more different words. The proposed Bitmask based compression yields better compression ratio for standard benchmark dataset such as Lineitem and Shakespeare.
XMill, a non queryable compressor achieved best compression ratio, on the average of 82%. The average compression ratio of XPress, queryable compressor is 45%. The average compression ratio of proposed work, RFX is 13%, QUICX is 58%, the IQCX is 64% and the average compression for QUICX with bitmask based compressor is 73%. Thus the compression ratio is improved by the proposed methods. XGrind, XPress, XQZip and XQuec are various existing queryable compressions techniques. Each compressor supports only restricted sets of queries. The proposed methods support wide range of queries and are efficient in executing queries efficiently than the existing approaches.