Schema Independent Queryable XML Compression Method

Otlapile Dinakenyane¹,* and Siobhán North²

Department of Computer Science, University of Sheffield

Abstract

The Extensible Markup Language (XML) is a self-describing extensible language that is widely used in the web. Its self-describing nature makes it notoriously verbose [1-2] resulting in large file sizes. Limited devices like smartphones cannot cope with very large files because of memory limitations. To address this problem, wireless connections are normally used to only load the necessary module of the database needed for evaluating a query at a given time. However, using wireless is costly in terms of energy given that Smartphones have a finite power source. Alternatively, compression methods can be used to address the memory problem. The current compression methods with high compression ratio like XMill do not allow query evaluation over compressed data. Other methods like XGrind and XPRESS allow query evaluation but need to be decompressed to evaluate range queries. This work seeks to design and develop an XML compression method that is schema-independent, has a compression ratio close to that of XMill and allows query evaluation over compressed data. This can be achieved by using labelling schemes to encode data and then adopting existing compressing methods to compress the data. Labelling schemes have always been used to reduce the size of the XML tree to optimize query evaluation. This research aims to investigate how the labelling schemes and existing compression methods can be brought together to achieve high compression ratio and allow query evaluation on compressed data so that users are more likely to be able to carry XML databases in their Smartphones.

Keywords: XML, XML Compression, Smartphones, native XML databases

1 Introduction

XML has been described as a de facto standard for storage and exchange of data over the web [1-2]. This is because XML has many attractive features which include: it is readable by both machines and people, its syntax is fairly simple and it has an extensible vocabulary where tags are user-defined. These tags mostly hold semantics about the data hence XML is said to be self-describing. The self-describing nature of XML makes it verbose which is a challenge for storage in limited devices. Between the tags is data which is referred to as Parsed Character Data (PCDATA) or Character Data (CDATA). In addition to the above mentioned attractive features XML is also platform independent, which is always desirable for application developers because it provides portability. For these reasons there are currently a large number of XML documents on the web. These documents can be stored as a set and manipulated together; querying and updating. When stored as a set, they are referred to as an XML database.

2 Background and Motivation

2.1 Smartphones

The increased processing capabilities of smartphones have led users to become dependent on them for carrying out more and more tasks. Smartphones are now used for creating documents, accessing emails, playing multimedia and retrieving information from servers via wireless connections [5]. They are classified as pervasive and ubiquitous devices [14] because they allow users to access information wherever they are. However, the use of smartphones raises many challenges because they have limited resources. Smartphones have limited memory, rely on finite power from a battery and are equipped with processors that are slower than those of high-end
computing devices like laptops. This is of interest in database research. The challenge is to design and build databases that can be run in these devices. Overcoming the problems caused by the constraints just listed is the basic challenge that a developer should keep in mind when developing an application or database for smartphones.

2.2 The Use of Communication

Databases developed for limited devices should have a small footprint [5-6] because of the memory limitation. Currently, this is achieved by loading only the necessary modules as and when needed. Such an approach implies that most of the data resides in the server and the smartphone acts as a comparatively thin client. The necessary modules are loaded using wireless connections. However, wireless connections are not ubiquitous [7], [8], [3]. Users experience many disconnections while carrying out transactions. Disconnections may be a result of lack of coverage, radio interference or hand-off between cellular base stations [3] causing the phone to acquire a new IP address and delaying executions of queries. These delays waste time and may render a good system inefficient because the time taken to retrieve information is longer than anticipated. For that reason, the database has to be aware of such disconnections so that it may rollback unfinished updates or pause and resume when the connection is regained.

The use of wireless interface is also costly on the lithium battery of a smartphone, more especially during data transmission [14] [3] and this is a concern because the battery can only provide finite energy. Therefore, though wireless seems to be a solution for limited memory it creates energy problems by consuming a lot of power. As stated in Lindholm [3], the battery problem for smartphones is likely not going to disappear. If the battery runs out, the database should be able to self-maintain.

Wireless disconnections and the battery depletion may result in inconsistencies in the data that is held in the phone and the server thereby compromising data integrity. Therefore some form of synchronization is required. Synchronization can be achieved by having a simple log with timestamps but it is more complex when dealing with interrupted transactions. Choi et al [15] suggested Synchronization Algorithms based on Message Digest (SAMD) which uses message digest to synchronize data.

Connecting a smartphone wirelessly to a server to cater for memory limitation works but has many weaknesses. In conclusion, an attempt to reduce memory footprint that relies heavily on wireless connection is not ideal because wireless communication is not always available, it consumes significant amounts of power and it exacerbates data integrity issues. Wireless can also be expensive because, in general, smartphones users are charged according to the amount of data used and are charged disproportionately more when they are roaming. Security can also be an issue because data is transmitted along a network.

2.3 Compression Methods

Compression methods can be used to provide a small footprint of a database locally thereby all but removing the need to rely on intensive wireless communication to address this particular goal. Many XML compression methods have been presented. XMill [9] has a very good compression ratio even compared to the classic Gzip compression method it is based on. However, it does not allow for data to be queried in the compressed state. This is not desirable because a database is expected to allow users to query the data on demand, without restrictions. To address this problem, queryable methods like XGrind [13], XPRESS [10] and TREECHOP [11] were proposed.

XGrind is a homomorphic compression method based on dictionary encoding and adaptive Huffman encoding. It compresses an XML file by scanning it twice. The first scan is to collect statistics about the data to create coding
models for the Huffman coding system. Scanning the input file twice creates a time overhead [10]. XGrind is also dependent on a schema to build up a dictionary and, hence, requires parsing and validation of the document. Though query evaluation is possible on compressed data, partial decompression is required when evaluating range queries. By allowing query evaluation on compressed data, the compression ratio is compromised in the case of this particular technique. To provide a comparison, an 89 MB file can be compressed to 38MB using this method whereas the same file can be compressed to 2.3 MB using XMill [1].

It is claimed that XPRESS has a better compression ratio than XGrind [10] and allows query evaluation on compressed data. XPRESS is not dependent on a schema but it also incurs a time overhead because it scans the input file twice. This method is based on Reverse Arithmetic Encoding. In this method, real number intervals are used to label elements paths. Like XGrind, XPRESS requires partial decompression to evaluate complex queries. For example, queries with textual range predicates sometimes require total decompression and this detracts from the efficiency of this particular method [10].

TREECHOP is based on SAX and is used for systems that use TCP/IP networks. In this method, query evaluation is done by scanning the compression stream. XMLZip [1] and XCQ [12] both require partial decompression to evaluate queries, which means that more space is required for query evaluation. It is clear that a more effective compression method is required. This compression method should have a compression ratio close to that of XMill, be schema independent and also allow query evaluation without decompression.

A non-compressed XML database allows query evaluation but the size that can be put in a limited device is severely limited due to the amount of memory that can be allocated to an application in a device like a smartphone. It is generally very small as compared to the compressed database that can fit in the same memory. Therefore a large gap exists between the size of the compressed database and the uncompressed database that can be put in the same memory. This research therefore seeks to bridge this gap by gradually compressing data to fit in a specified memory but retaining the ability to do query evaluation.

3 Plan

In the initial stage of this research, an empirical study will be carried out to set some parameters. Since the motivation for this research is based on smartphones, the memory limit will be based on them. The latest smartphones like iPhone 4, Blackberry Bold and Galaxy S all have up to 512 MB of RAM. In 2008, Riva and Kangasharju stated that, for a smartphone with 128 MB of RAM, only a fraction (between 10MB and 20MB) of this memory can be used by programs [8]. Given that in 128 MB of RAM a program is limited to between 10 MB and 20 MB then, for 512 MB of RAM, a program will be limited to 64MB. 64MB in 512 MB is equivalent to 16MB in 128 MB by ratio. This will be used for experimental test initially. Future hardware developments may allow a bigger limit but some starting point must be established.

XMill will be used to compress DBLP data set gradually to make it fit in 64MB of memory. This is to establish the actual size of the database that can fit in this memory. A subset of the uncompressed data will also be used to establish how much memory is needed for evaluating queries. The memory needed for query evaluation and the size of the database should 64MB altogether.

Having established the limits, subsequent experiments will seek to find out how much of that database can be queried when compressed taking into consideration the memory that is needed to evaluate the queries.
Optimization techniques such as labelling schemes may be used to reduce the size of the database along with compression methods to achieve the best possible compression ratio and have a compressed XML database that allows query evaluation. This research will investigate how they can be used to achieve this.

Compressing the database means the actual size of the compressed queryable database is expected to be bigger than that of the uncompressed database but it is likely going to be smaller than the non-queryable compressed database. The results will be evaluated based on how big the queryable compressed database is in actual size as compared to the uncompressed database. In respect to query evaluation, the experiments will be based on simple path queries first and then extended to handle much more complex queries. Performance will be compared with that of the uncompressed database.

References


