Designing a Resource–Efficient Data Structure for Mobile Data Systems

Richard Scott Gourlay

Department of Computer and Information Sciences
University of Strathclyde, Glasgow, UK
`Richard.Gourlay@cis.strath.ac.uk`

Abstract. Designing data structures for use in mobile devices requires attention on optimising data volumes with associated benefits for data transmission, storage space and battery use. For semi-structured data, tree summarisation techniques can be used to reduce the volume of structured elements while dictionary compression can efficiently deal with value-based predicates. This project seeks to investigate and evaluate an integration of the two approaches. The key strength of this technique is that both structural and value predicates could be resolved within one graph while further allowing for compression of the resulting data structure. As the current trend is towards the requirement for working with larger semi-structured data sets this work would allow for the utilisation of much larger data sets whilst reducing requirements on bandwidth and minimising the memory necessary both for the storage and querying of the data.

1 Introduction

Data intensive applications in mobile environments are increasingly dependent on processing semi-structured data. Such data is typically represented in a verbose format that is incompatible with the engineering constraints provided by small computing platforms. The development of mobile computing is associated with growth in the information available on the Internet and also in users expectations of the service that they will receive from mobile applications. The continuing development of miniaturised computing platforms that is necessary to provide for the expansion of mobile applications presents on-going engineering challenges in terms of minimising power use and at the same time maximising throughput. XML is a representation of semi-structured data that is widely used in distributed, Internet-based applications. The verbose native representation of semi-structured data needs to be replaced by an optimised physical structure for it to be a viable resource on mobile systems. This physical structure must combine the qualities of direct addressability with in-built indexing. Optimisations also need to be sought for query processing on such structures.

2 Background

Recently the memory, battery power and processor capabilities of mobile devices have increased greatly, but despite these advances the memory available to the system remains a critical resource for data-intensive mobile applications. Increasingly, mobile services rely on semi-structured data, for the storage and transmission of data. Care needs to be taken in the representation of such data to support optimal processing. Indexing can be achieved by external data structures although the original data needs to be retained to validate queries.

Conceptually semi-structured data can be represented as a graph with vertices used to indicate data items and structural interrelationships shown by arcs, arbitrary graphs can also be encoded in XML representation through the use of special ID:IDREF pairs. The structural elements can be compressed by retaining only a skeleton representation of the graph and dictionaries can be used to support non-redundant storage of leaf values. Although graphs are an accurate representation of semi-structured data, tree representations provide a useful simplification. A sequence of distinct arcs in a graph is called a *path*. Paths may be either *chains* (if the vertices in the path are distinct) or *circuits* (if the initial vertex and the final vertex are the same). Removing a single arc from each of the circuits in a graph produces a *spanning tree* that still connects all the data items in the graph. By removing different arcs, a number of different spanning trees can be produced from a single graph.

3 Related Work

Numbering schemes can be used to provide a basis for connecting structural indexes with value representations. Dietz [Die82] describes a data structure for efficient presentation of trees based on linked lists. An application of this structure is the determination of the ancestor-descendant relationship based on the pre- and post-order numbering schemes. *Pre-order numbering schemes* visit the root node first and then recursively traverse the rest of the tree. *Post-order numbering scheme* schemes visit parent nodes after all subtrees rooted by their children have been traversed recursively. Grust [Gru02] analysed the properties of Dietz's numbering scheme further and identified that the original pre- and post-order numbering scheme can also be used to answer queries along the previous/following axis of XPath. The scheme was extended to include direct references to parent nodes and type information (attribute node type or tag name and element) which allows for the storage of the complete XML document in a single relation. All XPath axis operations can be performed on the resulting relation with the help of only relational indices.

Work on integrating different kinds of index structures for semi-structured data was carried out by McHugh [MW99], who investigated heuristics to determine when to use the four specific forms of indices (value, text, link and path index). Halverson et al. [HB^{+}03] identify the need to combine pattern matching techniques based on inverted lists with the navigational approach typical for

XML tree traversal algorithms pointing out the lack of integration between these two lines of research. Vectorization of XML, developed by Choi and Buneman [CB03,BCF$^+$05], combines the XMill [LD00] approach for compact representation of atomic data with the approach for skeleton compression by sharing subtrees [BGK03]. The skeleton of typical XML documents is small and can be kept in memory while the data is only used in the later stages thereby avoiding unnecessary I/O operations. Kaushik et al [KK$^+$04] extend their original work [KB$^+$02,KS$^+$02] on structural indices for path expressions to include keyword constraints on the contained atomic data. They propose a general strategy to combine structural indices with inverted lists in order to address this class of queries efficiently and test their approach using the Niagara system [ND$^+$01]. As their value indices are based on techniques developed in the context of information retrieval systems, their resulting query system includes support for finding the k most relevant results. Structural and index values are combined by Amato et al [ADZR03] by extending the structure to incorporate the values for some elements or by incorporating B^+-Tree value indexes within the structure.

In a mobile data context the caching of query results allows for an improvement in response time and throughput data in the system. Gupta and Srimani [GS02] introduced a data caching system which was predisposed to repeated disconnections often associated with such systems. Semantic caching, the caching of semantic descriptions of previous queries and results, allows for future queries to be partially evaluated and trimmed to request only the data required that is not held in the current cache. Dar et al [DFJ$^+$96] proposed this scheme and showed its advantages over both page caching and tuple caching, Ren et al [RDK03] extended this work especially in the forming of the remainder queries. Lui et al [LLL05] further extend semantic caching into distributed environments and multi-dimensional queries.

4 Motivation

Queries over semi-structured data typically include both structural and value predicates. Query 1 provides an example that seeks to return the authors of books with the title ‘Databases’. Figure 1 shows the data graph of the example source with vertex identifiers added to each node.

Query 1 *//book[/author & /title/DATA=‘Databases’]*

Since Query 1 contains only forward facing query axes, it can always be answered by a single traversal of the data structure. However, this technique becomes impractical for very large data instances. For that reason, index structures are employed by most DBMS.

The structural index approach illustrated by the work of Kaushik et al can be used to resolve Query 1. Bisimilarity (i.e. the sharing of common subtrees) allows resolution of path location steps in linear time [BGK03]. A family of indexes ((j,k)-F+B-index) can be constructed using a range of values for forward or backward bisimilarity. Excluding the path from the root vertex, the longest

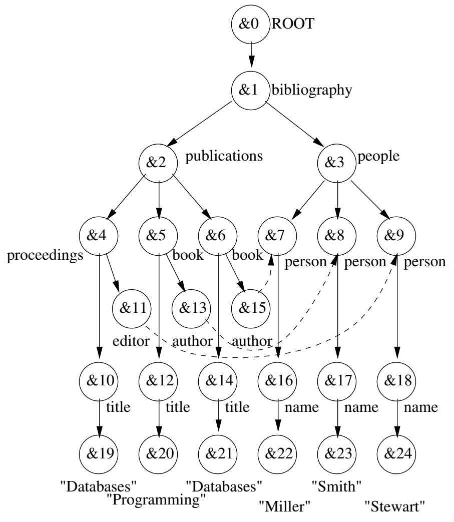

Fig. 1. Data graph of an example source

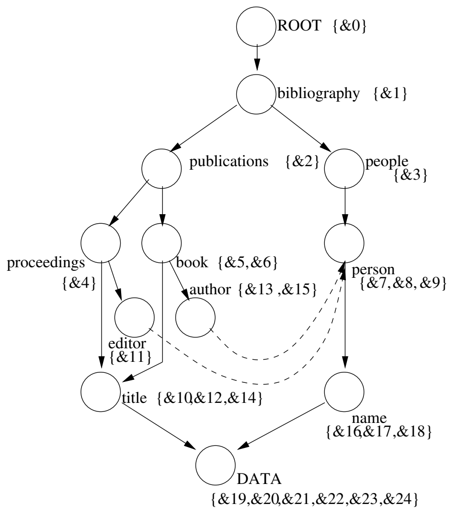

Fig. 2. (2,0)-F+B-index graph of Figure 1

forward facing path in the query graph has length two (`book/title/DATA`). There are no backward directed paths so the (2,0)-F+B-index shown in Figure 2 is the smallest covering index for the structural part of the example query, i.e. the query graph in which all value predicates have been replaced by a structural leaf predicate that select vertices with the special tag label `DATA`. This family of indexes does not incorporate atomic data, so no member of this index family is covering for this query.

Embedding the structural part of Query 1 into the index graph of Figure 2 can be done using the same algorithm that could be used to embed it into the data graph (since the graphs are bisimilar). The complexity of the embedding process remains unchanged but the size of the graph has been reduced from 25 vertices in Figure 1 to twelve vertices in Figure 2. Such a reduction in size can be expected for most semi-structured sources, as most practical data graphs contain only very few structural building blocks [BGK03]. The structural elements of the query can be resolved against the index graph but the original data graph needs to be maintained in order to resolve the value predicate.

The query evaluation process, starting with the atomic value predicate can be presented in terms a dictionary-based structure such as DDOM [NW02]. Figure 3 shows the fully indexed dictionaries and the structural array of a part of the example source. Text values are stored in dictionaries and references to these are preserved in the structure array. Recurring instances of the same text in a particular XML tag are stored once in each domain dictionary. Using this approach on Query 1, the existence of `title` vertices containing the atomic value 'Databases' can be quickly verified. These are the entries at the addresses **8** and **24** in the structural array, corresponding to the vertices **&19** and **&21** of Figure 1. The `book` vertices are represented by entries starting at addresses **11** and **19** in the structural array, the entries for the `author` vertices start at the addresses **12** and **20**. A linear scan through the structural array is needed to verify ancestor-descendant relationships between entries. In the example given, a scan for the first `book` entry starting at position **11** leads to a `title` entry at address **15**, but none of the identified atomic value entries is encountered before the closing tags of the `title` and `book` entries are found at positions **17** and **18** respectively. Thus this entry, corresponding to vertex **&5** of the data graph, does not represent a valid result. The similar scan starting at the `book` entry at position **19** matches all the required entries from the list of potential descendants, thus the result is valid.

Storing the complete range using the start and end addresses of the subtree rooted by a node, as shown in Figure 3, allows derivation of the ancestor-descendant relationship using this information alone. The fact that identifiers can be used to indirectly encode structural relationships between nodes of a tree will be used by the hybrid representation presented here. Although this allows the validation of the structural constraints between individual nodes, it still does not allow the selection of a set of nodes based on their structural properties as can be achieved using the index graphs.

The F+B-index and DDOM approaches originate from different perspectives and lack a common element that could be used for their combination. Index graphs allow set-at-a-time operation and maintain structural relationships between vertex-sets whilst abstracting away from the individual vertices of the data graph. Dictionary coding organises data into homogeneous domains and maintains the identity of individual vertices of the data graph but their structural relationships are not exposed directly.

5 Hypothesis and research questions

The purpose of this experimental work is to compare memory and processor performance on a prototype of a hybrid data structure with a similar system incorporating minimal bit string compression of the data. Several related research questions are also to be investigated.

5.1 Is it better to use a minimal bit string representation of semi-structured data rather than the native representation especially in the context of mobile applications?

With increasing bandwidth and wireless network availability is it still necessary to compress all but the largest of data sets before transmitting them to a mobile device? The main inconveniences of large text based data structures include the need to correctly transmit large amounts of data and providing storage for these files on the receiving device.

The traditional solution to this problem is to utilise a client/server architecture, keeping the data on a server and allowing the mobile device to form and transmit queries. The server can then processes the query before transmitting

#	Type	↗
0	Document	–
1	Element	1
2	Element	2
3	Element	3
4	Attribute	1
5	Text	1
6	/Attribute	1
7	Element	5
8	Text	1
9	/Element	5
10	/Element	3
11	Element	4
12	Attribute	2
13	Text	1
14	/Attribute	2
15	Element	5
16	Text	2
17	/Element	5
17	/Element	4
19	Element	4
20	Attribute	2
21	Text	2
22	/Attribute	2
23	Element	5
24	Text	1
25	/Element	5
26	/Element	4
27	/Element	2

↘	Element	Index Entries
1	bibliography	(1:54)
2	publications	(2:27)
3	proceedings	(3:10)
4	book	(11:18),(19:25)
5	title	(7:9),(15:17),(23:25)
6	people	...
7	person	...
8	name	...

↘	Attribute	Index Entries
1	editor	(4:6)
2	author	(12:14),(20,22)
3	id	...

↘	Text:title	Index Entries
1	Databases	8,24
2	Programming	16

↘	Text:name	Index Entries
1	Miller	...
2	Smith	...
3	Stewart	...

↘	Text:author	Index Entries
1	p2	12
2	p1	21

↘	Text:editor	Index Entries
1	p3	5

Fig. 3. Structure array and indexed domain dictionaries used by DDOM

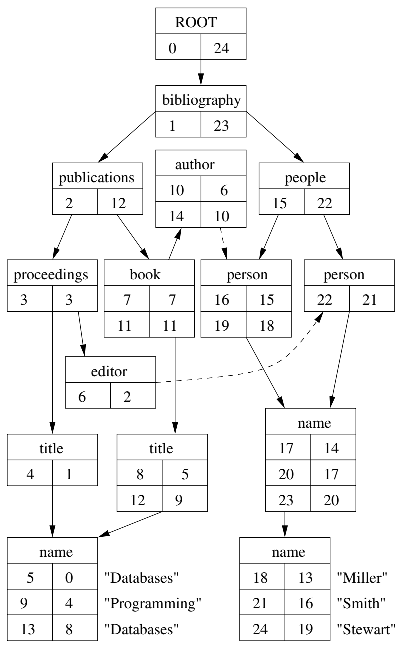

Fig. 4. NSGraph – combining structural and signature information

the results back to the mobile client. The weakness of this system is that there need to be a reliable data communication between the client and server whenever a query or response is required. Several optimisations have been implemented to this system including both semantic and data caching schemes.

A simple alternative solution is to transfer a copy of the data set onto the mobile device and then allow the device to the query data locally. This system however has the disadvantages of needing a reliable high bandwidth connection in order to transfer the data set and have sufficient memory available on the mobile device to allocate to storing and querying the data.

In order to reduce the overhead of transmitting large datasets the original data can be compressed, normally to less than 5–10% of its original size, this compressed file is then transferred to the mobile device. This system has the advantage of reducing the time and bandwidth required to transfer the data set at the processor cost of compressing and decompressing the data, sufficient storage memory is still required to contain and query the uncompressed data.

A fourth possible solution would be to encode a compressed representation of the dataset that can then be transferred and kept in a compressed format that can be queried directly. This reduces both the initial transfer overhead as well as the memory required to store and query the dataset.

5.2 Does a representation of semi-structured data that combines structural indexing and minimal bit strings outperform a similar non-optimised representation in terms of memory usage and processing overheads?

Dictionary coding allows for efficient compression of text giving a minimal bit string representation for common sub strings, there is however an overhead when decoding the compressed representation. The potential benefit of this approach is that the memory footprint of the minimal bit string system will be considerably smaller than the non-minimal system while the processor utilisation may well be increased as a trade-off.

An existing system has been developed that allows for the structural indexing of semi-structured documents via the F+B-index scheme. The objective of the current research is to implement dictionary coding into this system and therefore determine whether the memory requirements and processor usage involved in a minimal bit representation perform favourably with existing solutions as well as the non-minimal system.

5.3 What are the battery usage trade-offs when using a combined structural indexing and minimal bit strings representation of semi-structured data compared to existing representations?

In mobile systems not only is there a restriction on processor power and memory, both main and secondary, but also on the battery power consumption cost of processing. The aim of this research is to expand the previous evaluation of processor and memory usage to address the costs and benefits of minimal bit string representation of semi-structured data in the context of power consumption in a mobile device and therefore the expected life of the battery .

5.4 What benefits do 64bit architectures bring to minimal bit string compression and therefore to representations of semi-structured data which combine structural indexing with minimal bit strings?

One of the most anticipated recent developments in computing is the cost-effective introduction of 64bit architectures onto various computing platforms. 64bit architectures are already becoming common place in large scale server systems and stand-alone desktop systems, future mobile systems may well also be based on 64bit architectures. While these processors will inevitably be different to the existing 64bit processors, similar costs and benefits can be expected in terms of minimal bit string compression. The purpose of this research is to identify the possible cost and benefit trade-offs of using minimal bit representations on 64bit architectures in terms of memory usage and processor overheads.

6 Research strategy and milestones

Whilst the F+B-index approach provides a fast and compact representation for resolving queries on semi-structured data, supporting the structural part of query resolution, validation ultimately requires access to the native data representation. The DDOM approach replaces the native representation with a more compact structure that exploits the redundancy often occurring in large data structures. The research we are undertaking aims to combine the F+B-index and DDOM approaches to produce an efficient representation of the data graph that will allow querying of and access to a compressed representation of the data set.

This hybrid data structure is based on combining the F+B-index graph with the DDOM indexed dictionaries. The vertex identifiers used in both the index graph and the dictionaries are then replaced with the entries based on the numbering schemes creating a unique address space for validation purposes, this structure is referred to as an NSGraph(*N*umbering *S*cheme *Graph*). Figure 4 illustrates this scheme using the (1,1)-F+B-index graph of the original data graph shown in Figure 1. In this illustration the incorporated atomic value dictionaries are suppressed in order to simplify the diagram.

The purpose of this system is to allow the determination of the memory requirements and processor usage involved in a system utilising minimal bit representation and therefore determine how such a system performs in relation to both the traditional solutions and an existing non-minimal prototype previously developed. An existing prototype system developed in Java gives a base for this comparison. This system has already given some promising results in terms of memory utilisation and visit cardinality when compared to the data graph. Figure 5 shows the memory utilisation of a number of XML benchmark files by both the data graph and the NSGraph data structure, it can be seen that there is around a 50% reduction when using the NSGraph representation. Figure 6 gives the visit cardinalities for a number of classes of queries when run over the NSGraph and Data graph representations, shown in Figure 7, these results show an encouraging reduction in the number of visits required to resolve the sample queries, some even showing orders of magnitude reductions.

We anticipate that the final developed system will at least match the results shown in the prototype as well as adding improvements in the compression of the data and management of system memory. Future improvements may be based on the results of the related research issues including efficient dictionary coding in 64bit architecture environments.

7 Conclusion

We are currently developing a system that combines the benefits of an F+B-index with the additional benefits of dictionary coding. The intention is to produce a data structure that can utilise the fast response characteristics of structural indexing whilst removing the need to refer to the original uncompressed data

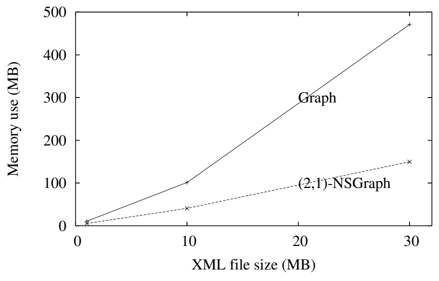

Fig. 5. Memory use in Data graph vs. NS-Graph

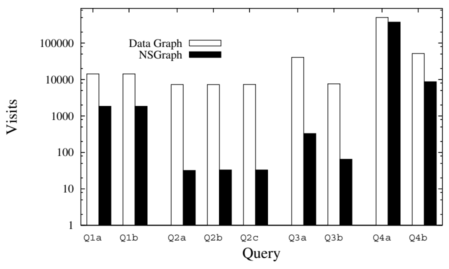

Fig. 6. Visit cardinality in Data graph vs. NSGraph

Class	Query	Tree pattern	Results
Q1	Q1a	//person[/name/DATA="Klemens Pelz" & /watches & /emailaddress& /creditcard]	0
	Q1b	//person[/name/DATA="Klemens Pelz" & /watches & /emailaddress & /phone]	0
Q2	Q2a	//profile[/income & /education & /gender]	972
	Q2b	//profile[/income & /education & /gender/DATA="male"]	477
	Q2c	//profile[/income & /education & /gender/DATA="female"]	495
Q3	Q3a	//item[/location/DATA="United States" & /payment/DATA="*Creditcard*"& /quantity/DATA="1"]	2044
	Q3b	//item[/payment[/DATA="*Creditcard*" & /DATA="*Cash*"]]	1540
Q4	Q4a	//description[//text[//keyword & //bold]]	4579
	Q4b	//description[//text/*[/keyword & /bold]]	109

Fig. 7. Sample queries used in Figure 6

structure. In the context of mobile systems this optimises use of data caching whilst at the same time minimising the data representation and consequently saving on memory, battery and processor utilisation.

The purpose of the prototype system was to explore the effect of variations in bisimulation on the processing of queries as well as to measure the effect that compressing and indexing semi-structured data has on the size of its in-memory representation. The objective of the current research is to build on the promising results gathered in initial experiments and to determine whether the memory, battery and processor usage involved in a combined structurally indexed-minimal bit representation will perform favourably with both traditional solutions and similar structurally indexed non-minimal systems.

References

[ADZR03] G. Amato, F. Debole, P. Zezula, and F. Rabitti. YAPI: Yet another path index for xml searching. In *Proc of ECDL*, pages 176 – 187, 2003.

[BCF$^+$05] Peter Buneman, Byron Choi, Wenfei Fan, Robert Hutchison, Robert Mann, and Stratis D. Viglas. Vectorizing and querying large xml repositories. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 261–272, Washington, DC, USA, 2005. IEEE Computer Society.

[BGK03] P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *Proc. of VLDB*, pages 141–152, 2003.

[CB03] B. Choi and P. Buneman. XML vectorization: A column-based XML storage model. Technical Report MS-CIS-03-17, University of Pennsylvania, 2003.

[DFJ$^+$96] S. Dar, M. J. Franklin, B. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 330–341, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[Die82] P. F. Dietz. Maintaining order in a linked list. In *Proc. of STOCS*, pages 122–127, 1982.

[Gru02] T. Grust. Accelerating XPath location steps. In *Proc. of SIGMOD*, pages 109–120, 2002.

[GS02] Sandeep K. S. Gupta and Pradip K. Srimani. Data management in wireless mobile environments. pages 553–579, 2002.

[HB$^+$03] A. Halverson, J. Burger, et al. Mixed mode XML query processing. In *Proc of VLDB*, pages 225–236, 2003.

[KB$^+$02] R. Kaushik, P Bohannon, et al. Covering indexes for branching path queries. In *Proc. of SIGMOD*, pages 133–144, 2002.

[KK$^+$04] R. Kaushik, R. Krishnamurthy, et al. On the integration of structure indexes and inverted lists. In *Proc. of SIGMOD*, pages 779–790, 2004.

[KS$^+$02] R. Kaushik, P. Shenoy, et al. Exploiting local similarity for indexing paths in graph-structured data. In *Proc. of ICDE*, pages 129–140, 2002.

[LD00] H. Liefke and D.Suciu. XMill: An efficient compressor for XML data. In *Proc. of SIGMOD*, pages 153–164, 2000.

[LLL05] B. Liu, W. Lee, and D. L. Lee. Distributed caching of multi-dimensional data in mobile environments. In *MDM '05: Proceedings of the 6th international conference on Mobile data management*, pages 229–233, New York, NY, USA, 2005. ACM Press.

[MW99] J. McHugh and J. Widom. Query optimization for XML. In *Proc. of VLDB*, pages 315–326, 1999.

[ND$^+$01] J F. Naughton, D. J. DeWitt, et al. The Niagara internet query system. *IEEE Data Eng. Bull.*, 24(2):27–33, 2001.

[NW02] M. Neumüller and J N. Wilson. Improving XML processing using adapted data structures. In *Proc. of WEBDB*, pages 63–77, 2002.

[RDK03] Q. Ren, M. H. Dunham, and V. Kumar. Semantic caching and query processing. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):192–210, 2003.