Sharing large data collections between mobile peers

Brian Tripney, Christopher Foley, Richard Gourlay, John Wilson
Department of Computer and Information Sciences
University of Strathclyde
Glasgow, G1 1XH. UK
{brian, chris, rsg, jnw}@cis.strath.ac.uk

ABSTRACT

New directions in the provision of end-user computing experiences mean that we need to determine the best way to share data between small mobile computing devices. Partitioning large structures so that they can be shared efficiently provides a basis for data-intensive applications on such platforms. In conjunction with such an approach, dictionary-based compression techniques provide additional benefits and help to prolong battery life.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems, Distributed Databases; E.4 [**Coding and Information Theory**]: Data compaction and compression

General Terms

Experimentation, Performance

Keywords

Peer-to-peer, data sharing

1. INTRODUCTION

Growth in personal computer sales dropped almost to zero in 2008. Meanwhile two hundred million smartphones were sold, an increase of 13% on the previous year[1]. Smartphone users are becoming accustomed to their needs for data being satisfied on demand, thereby presenting a wealth of fresh challenges for computer science. In situations where a plethora of small devices are operating in an infrastructure-light environment it is more effective to share information between local mobile peers and only involve a central server as necessary. Location awareness can be used to ensure that the right information is available at the right time.

[1] `www.zdnetasia.com/news/communications/0,39044192,62052144,00.htm`

This paper describes an architecture for sharing large collections of semistructured data between many small mobile devices with the aim of reducing their dependence on fixed server infrastructure. It also reports the initial steps we have taken to implement and evaluate this architecture. We expect that our work will contribute to an understanding of query processing and data management in XML data models particularly in the framework of location and context-aware applications and services.

We start with a definition of the problem addressed and show how research in peer-to-peer database systems (PDBS), mobile and *ad hoc* networks (MANETs) and semistructured data provides a basis for our approach. We present the methodology of our design and the preliminary results. Finally we characterise the current stage of the project and the work that is planned to enable an overall evaluation of our ideas.

2. PROBLEM STATEMENT

For many years, client-server models have dominated the sharing of data in Internet contexts. More recently, peer-to-peer (P2P) data exchange has become a widely used method of resolving the issues presented by server overload. Whilst this helps to solve the problem of inadequate server infrastructure, typical conceptions of P2P topologies are based on the dynamic creation of *ad hoc* networks in a wired world. Performance of such systems is always a concern but given a balance between the number of uploading and downloading peers and the available bandwidth, adequate response times can usually be supported. Since peers in this scenario are typically desktop or laptop computers, the location of the user is not significant in the context of the data required. We have already noted the remarkable growth in the number and power of small mobile computing devices and the way that the expectations of the user population have similarly expanded. Tourism, advertising and entertainment provide large-scale application areas where users need access to location sensitive data. Users increasingly expect instant access to information on the move, however storage restrictions imposed by limited capability mobile devices prevent the preloading of large data collections. On the other hand, partial disconnection intermittently prevents loading the data from a server 'on the fly'. In this context, neither the client-server model nor the conventional P2P sharing are efficient enough to support data intensive applications where large numbers of users demand access to large data collections.

We address this dilemma by using P2P distribution schemes to share compressed, partitioned XML. We arrange that queries over this data can be resolved without first fully decompressing the complete structure. This extends knowledge in the area of PDBS in the context of small mobile devices. We plan to investigate the most efficient way of propagating the physical representation of such data to peer devices that do not already possess it, an issue that has been neglected in the past. Without establishing this, succeeding generations of smartphones will continue to struggle with data intensive applications.

We assume a partial disconnection model dominated by disconnection periods and a class of applications in which localisation defines the parts of a data collection of greatest interest. Shoppers wanting to see product promotions whilst on a busy shopping street represent such a scenario. Fragmented localised data is propagated between shoppers' phones or pulled if the shopper requested data not already present. Fragments of the data are locally autonomous and the assembly is managed by federation, thereby avoiding the need for communication with a centralised directory structure. During busy periods, the database would flow within the phone population but never be wholly resident on a single device. If there are few shoppers on the street, those who want the data would have to wait for a server connection or pay for data via GSM. Although available bandwidth will grow in future, an approach such as ours will be necessary to limit the effects of network congestion. Compression will also save CPU cycles and consequently battery power; a significant future limiting factor for small mobile devices. Our approach to partitioning is an essential component of this scenario since small partitions and their associated dictionaries can be propagated efficiently between phones and queried without complete decompression. The decompression load then becomes proportional to the size of the result set. The partitions are logically homogeneous so there is a strong likelihood that user queries over data relevant to a particular location can be satisfied without recourse to pulling additional data from other phones, making this approach the most effective way of servicing shoppers' needs during disconnection periods. Queries that span partitions would be more expensive but processing the incoming partition as a stream will improve the effective performance of the system. The technique integrates with raw textual and multimedia data. Query results over partitioned and compressed data can be provided directly where predicate values are contained in the partitioned structures or indirectly where reference needs to be made to the underlying multimedia files.

3. OVERVIEW OF RELATED WORK

Buneman and co-workers combine the XMill[9] approach for compact representation of atomic data with the approach for skeleton compression by sharing sub-trees[2] to address XML join queries. Kaushik et al.[6] include keyword constraints on the contained atomic data. In earlier work, we investigated the use of dictionary compression techniques for representing both tags and values in XML structures[4, 12] and other investigators have also examined the decomposition of XML into array-based structures[3]. Dictionary compression has been used with mobile client systems that refresh themselves from a central server[10]. PDBS[1] have attracted significant attention and have typically focused on the use of distributed hash tables. Algebraic optimisations for managing distributed XML data structures have also been proposed[7]. There is recognition that a variety of approaches may be necessary to exploit various communication architectures[5]. Simulation of client-server and P2P networks suggest that scaling in client-server systems can only be addressed by the additional expense of adding more servers and providing appropriate management for these systems[8].

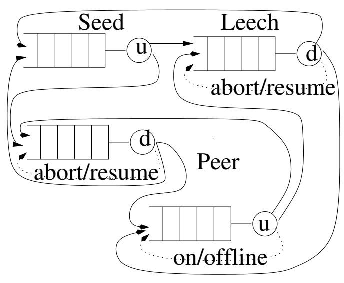
A simplified analytical model[11] can be used to assess the impact of data compression in peer-based architectures. In such system, there will inevitably be users who will share the data elements they possess and those for whom the technical or economic benefits that are available as a consequence of sharing are insufficient to persuade them to take part (leeches). The BitTorrrent model of incentivization promotes sharing by down-grading the service to those who don't share and improving service to those who are willing to host software that will perform P2P sharing. Incentivization can also be provided by using micropayment models to offset the potential for increased call charges. However, as with all P2P-based systems, leeches are likely to be a persistent feature of the environment. The scenario involves devices that upload only (seeds), download only (leeches) and both upload and download (peers). The relationships between these kinds of devices are shown in Figure 1. Each queue consists of a list of tasks and a server (u - upload, d - download). The queues form a closed queuing network i.e. the system has no source or sink. The model is simplified by assuming that all peers have equal capacity both to upload and download and that in a steady state, the number of uploads and downloads is independent of time. The model incorporates random churn of the peer pool via the off line rate (κ) and the abort rate (ρ). These are expressed as a rate correction to the upload rate (μ) and download rate (τ) respectively. The model is represented by the expressions shown in Figure 1. Little's Law is used to produce the service time (t) experienced by users waiting for the completion of downloads (3). Compressing data provides leverage for both upload and download and is incorporated into the model by the coefficient δ.

Assuming a nominal upload bandwidth (μ) of 12 Mbits/sec, off line rate (κ) of 50 Mbits/sec, individual peer download bandwidth usage (τ) of 20 Mbits/sec and an abort rate (ρ) of 10 Mbits/sec, the analytical model produces estimates of service time shown in Figure 2. It can be seen from that an increasing selfish rate increases the expected download time for other peers but that this is considerably mitigated by the effect of compression. Similarly, at a fixed selfish rate, the effective download bandwidth is improved by increasing compression.

This model predicts that compressing data improves service to users irrespective of the number of leeches in the system. In addition to these beneficial effects of compression, the effect of partitioning data structures so that only limited relevant data is sent between peers, results in additional savings. This further reduction will provide more efficient use of bandwidth and better resilience to high selfish rates than is predicted by the analytical model. Since the number of available peers varies in inverse proportion to the number of leeches, the model also suggests that peer pool variations will be mitigated by compressing data.

4. RESEARCH METHODOLOGY

Compressing data provides a useful step in empowering



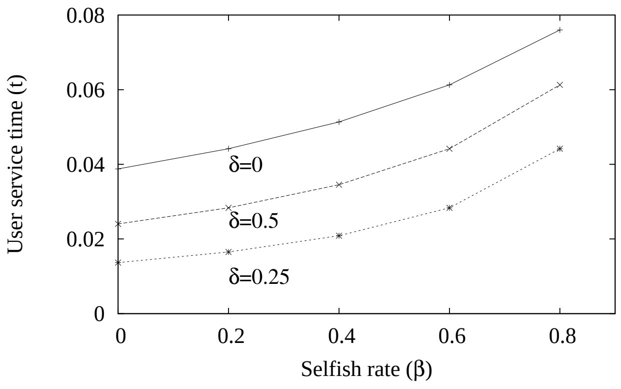
$$\frac{1}{\iota} = \frac{1}{1-\beta}\left(\frac{1}{\mu} - \frac{1}{\kappa}\right) \quad (1)$$

$$\frac{1}{\theta} = \max\left(\frac{1}{\tau}, \frac{1}{\iota}\right)\delta \quad (2)$$

$$t = \frac{1}{\theta + \rho} \quad (3)$$

ι	effective upload rate
β	selfish rate
μ	nominal upload rate
κ	off line rate
θ	bottleneck bandwidth
τ	download bandwidth usage by each peer
ρ	abort rate
δ	compression ratio
t	service time

Figure 1: Queueing model for P2P data sharing



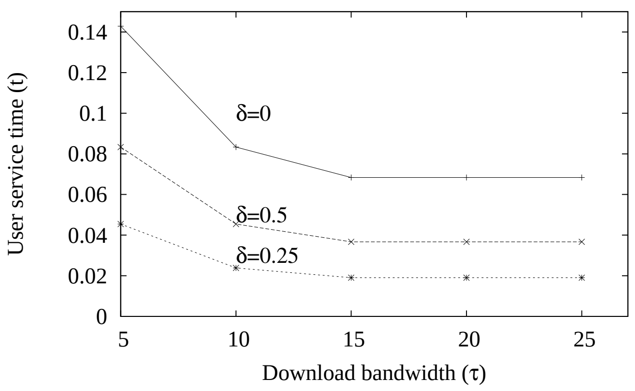


Figure 2: Performance characteristics of the queueing model

applications running on phones to rapidly receive large volumes of data in response to requests from users. Efficient processing of XML is helped by choosing the right physical data structures and supporting them with appropriate indexing techniques. Bisimilarity (i.e. the sharing of common subtrees) allows resolution of path location steps in linear time [6]. A family of indexes ((j,k)-F+B-index) can be constructed using a range of values for forward or backward bisimilarity. Embedding the structural elements of queries into such an index graph can be done using the same algorithms that could be used to embed them into a data graph. 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 value predicates. The approach we have developed (NSGraph) [12] constructs atomic data dictionaries according to the structural groupings. Consequently the part of the dictionary corresponding to a structural grouping can be incorporated into the node of the index graph representing it. The vertex identifiers used in both the dictionaries and the index graph can be replaced with the entries based on a numbering scheme, creating a unique address space for validation purposes. This approach produces a hybrid that represents the cross-product of an index graph with a signature with its leaf nodes being replaced by domain dictionaries. Figure 4 illustrates this using the (1,1)-F+B-index graph of the example data graph shown in Figure 3. This approach allows queries to be resolved directly on the compressed data structure with only the returned values being decompressed.

Memory-boundness is a significant limitation of NSGraph. Our initial implementation requires the complete graph and the associated leaf values to be present in memory. Dictionary compression and fragmentation of the compressed XML tree reduces the size of the data structure that needs to be held in memory. The NSIndex extension builds on the NSGraph model by supporting non-volatile storage of the resulting structure and providing for its direct interrogation. This approach gives the advantage of not having to parse and store the entire data structure in memory in order to evaluate a query, as well as giving opportunities for optimisations of the data structure. The architecture for this extension is split into three major modules shown in Figure 5. In the NSGraph module the underlying source data structure is read into the structural summarisation and the block-based dictionaries are generated. This model of the source is fed into NSStore where the memory-based structure is mapped to the non-volatile structure seen in the lower part of Figure 5. The NSQuery system can be used to directly read the non-volatile structure and allow for the evaluation of arbitrary queries. The compiled representation is separated into three components, shown in Figure 5. Partitioning, based on bisimilarity, is performed as the source data structure is processed by the NSGraph module. Blocks are produced by grouping bisimilar data together and these blocks, along with the structural summarisation, form the complete NSGraph data structure. The NSIndex structural summarisation is a mapping of the blocks that make up the compressed XML structure. Blocks provide a container for elements holding the numbering scheme data (pre-order, post-order, level & size). A DataBlock is a container for data elements, each being represented as pre, post, level and size, together with a minimal-bit token to indicate the raw data value contained in the appropriate dictionary. All data elements in one DataBlock use the same dictionary. The dictionaries

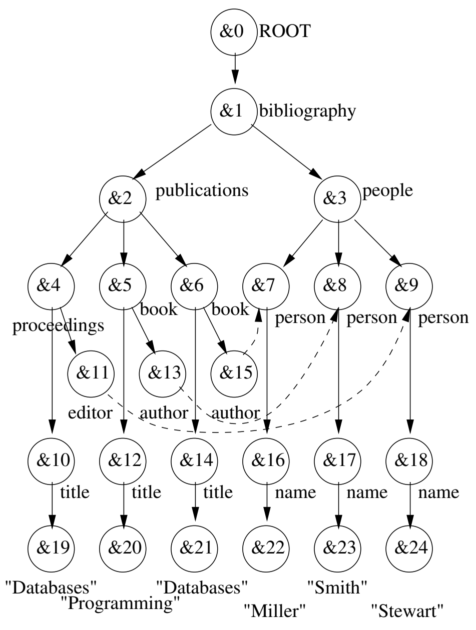


Figure 3: Data graph

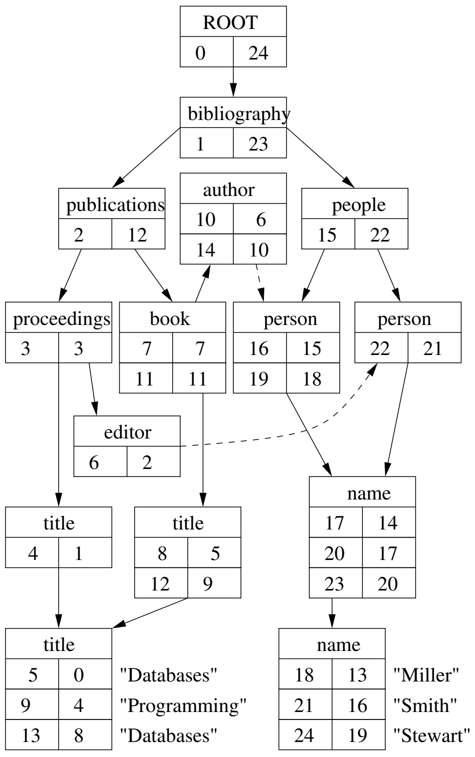


Figure 4: NSGraph

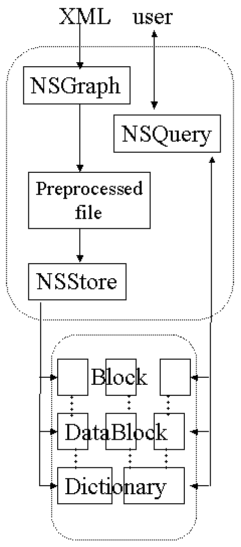


Figure 5: NSIndex component architecture

may be shared by a number of DataBlocks and contain the mapping between the token and the raw data value.

5. PRELIMINARY RESULTS

Initial work on NSGraph [12] suggests that useful compression can be obtained. More recently, we have established the effects of different levels of forward and backward partitioning on the data and associated dictionaries. A number of different datasets (XMark10, NASA, Orders-E15 and Legal-13) with sizes of 10-15Mbytes were processed.

Each file was partitioned in sixteen ways using a version of the NSGraph program - these ranged from (0,1)-F+B (0-forward and 0-back) to (3,3)-F+B. This produced a set of uncompressed DataBlocks for each partitioning scheme. The unique values from each DataBlock were then extracted and used to calculate the size of data and dictionaries under the minimal-bit scheme. For each data file, the total size of uncompressed blocks is unaffected by the partitioning scheme used (as the complete set of blocks will contain all the data values regardless of how these are distributed across blocks). However, the compressed data size is affected by the distribution of data values, as the success of the minimal-bit scheme relies upon the repetition of values within individual blocks.

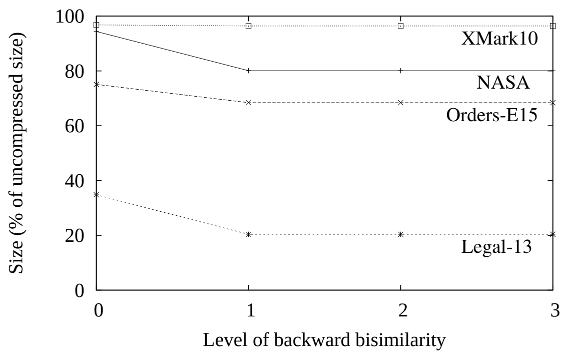
Across all files tested, compression was improved by moving from (0,0)-F+B to (0,1)-F+B. For most files further increases had no effect upon compression (one file showed a <1% compressed size increase between (0,1)-F+B and (0,2)-F+B) (Figure 6). The move to (1,n)-F+B (where $n < 1$) caused at best a 1% improvement in compression and had an adverse effect of up to 3% in the worst case. Some files were completely unaffected by forward bisimilarity (Figure 7). With no great benefit shown by partitioning using forward bisimilarity, and noticeably improved compression shown only when moving up to one level of backward bisimilarity, it would appear that the (0,1)-F+B partitioning scheme allows for greatest compression with least effort.

However, with a view to sharing the data in individual blocks, the overall size is only one consideration. The number of blocks the data is split into is also a factor, as fewer blocks will necessarily be larger blocks. In all cases the move from (0,0)-F+B to (0,1)-F+B produces a slightly increased number of blocks. A move from (0,n)-F+B to (1,n)-F+B ($n > 1$) can greatly increase the number of blocks for some data sets. It follows that partitioning the data into different numbers of blocks will cause the overall compression level for that data to change. There is a level of forward and backward bisimilarity beyond which no significant changes are made to the data blocks caused by the fairly flat, regular schema of the test data files. Files with a deeper tree structure are expected to show further changes in partitioning as bisimilarity is increased.

5.1 Current stage of the project

We have completed a working implementation of the parser that constructs the NSGraph and NSStore (Figure 5) elements of our architecture. We are currently working on NSQuery. There are three basic approaches to query evaluation in decomposed tree structures. Queries can either be evaluated based on the root of the structure and extending through the tree in a top-down fashion and finally validating the results against the dictionary. The bottom-up alternative to this involves resolving predicates at the dictionary level and joining these results to the tree structure. A hybrid of these approaches is also possible.

We are also further investigating the potential for splitting



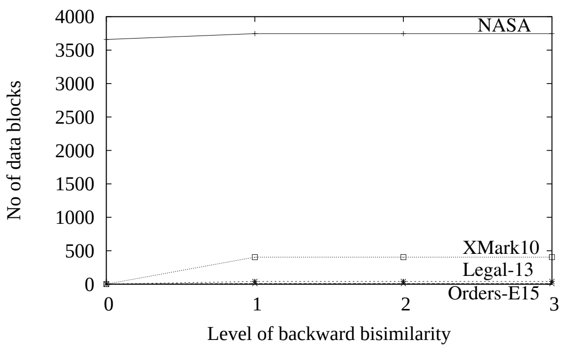
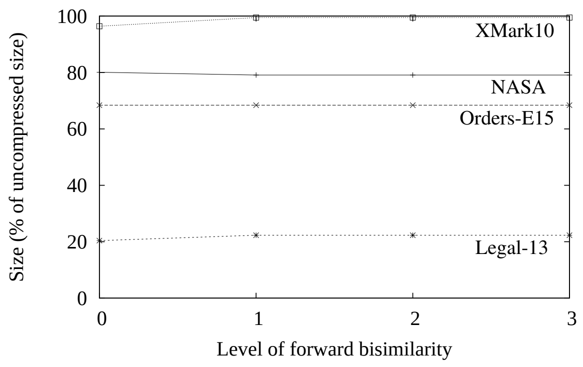


Figure 6: Effects of increasing backward bisimilarity (f=0)



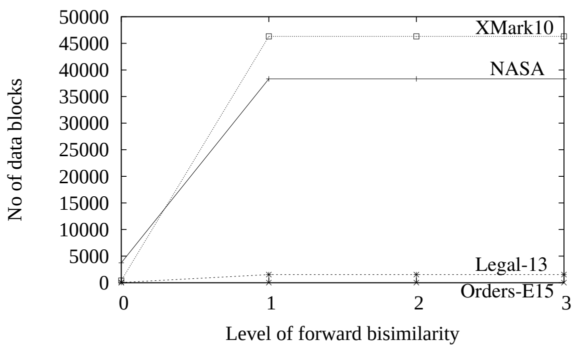


Figure 7: Effects of increasing forward bisimilarity (b=1)

dictionaries and re-combining small dictionaries to optimise the use of minimal bit tokens. We are also planning to evaluate the sharing of the partitioned elements between devices in the context of location-based data.

6. FUTURE PROSPECTS

Our initial results suggest that bisimilarity provides a basis for partitioning large data structures so that they can be compressed efficiently. The need to process large structures that may not fit into main memory has led to the development of interest in XML stream processing and the potential for compressed streams of XML to be made accessible to query processing. Stream processing will provide better response times for our target class of applications and we plan to explore the use of this approach in conjunction with sharing both dictionary and block information. Security and integrity of data in such an environment are important issues and we plan to address the implications of authentication, access control, privacy and encryption.

7. REFERENCES

[1] A. Bonifati et al. Distributed databases and peer-to-peer databases: past and present. *SIGMOD Rec.*, 37(1):5–11, 2008.

[2] P. Buneman et al. Path queries on compressed XML. In *Proc 29th VLDB*, pages 141–152, 2003.

[3] P. Buneman et al. Vectorizing and querying large XML repositories. In *ICDE*, pages 261–272, 2005.

[4] R. Gourlay, B. Tripney, and J. Wilson. Compressed materialised views of semi-structured data. In *Workshops of the 24th BNCOD*, pages 75–82, 2007.

[5] J. Kangasharju and S. Tarkoma. Benefits of alternate XML serialization formats in scientific computing. In *Proc SOCP'07*, pages 23–30, 2007.

[6] R. Kaushik, R. Krishnamurthy, et al. On the integration of structure indexes and inverted lists. In *ICDE 2004*, page 829. IEEE Computer Society, 2004.

[7] G. Koloniari and E. Pitoura. Peer-to-peer management of xml data: issues and research challenges. *SIGMOD Rec.*, 34(2):6–17, 2005.

[8] K. Leibnitz et al. Peer-to-peer vs. client/server: Reliability and efficiency of a content distribution service. In *Int. Teletraffic Cong.*, pages 1161–1172, 2007.

[9] H. Liefke and D. Suciu. XMILL: An efficient compressor for XML data. In *Proc SIGMOD*, pages 153 – 164, Dallas, Texas, USA, May 2000.

[10] Y. Natchetoi et al. EXEM: Efficient XML data exchange management for mobile applications. *Information Systems Frontiers*, 9(4):439–448, 2007.

[11] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *SIGCOMM '04*, pages 367–378, New York, NY, USA, 2004. ACM.

[12] J. N. Wilson et al. A resource efficient hybrid data structure for twig queries. In *Proc XSym'06)*, pages 77–91, 2006.