On Clustering Schemes for XML Databases

Damien K. Fisher William M. Shui Franky Lam Raymond K. Wong
School of Computer Science & Engineering
University of New South Wales
Sydney, NSW 2052, Australia
{damienf,wshui,flam,wong}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-0315
June 2003

Abstract

Although clustering problems are in general NP-hard, many research efforts have been put in the areas of OODB and RDBMS. With the increasing popularity of XML, researchers have been focusing on various XML data management including query processing and optimization. However, the clustering issues have been disregarded in all their work. This paper provides a preliminary study on data clustering for optimizing XML databases. Different clustering schemes are compared through a set of extensive experiments.

1 Introduction

In recent years, XML [2] has become one of the dominant data representation languages. While the reasons for its success are not immediately clear, it is likely that XML's ability to represent relatively unstructured data has played a key role in its success. Relational databases have dominated database research for the past thirty years, but the success of XML has highlighted that its restrictive data model does pose some real world problems. In spite of these difficulties, attempts have been made to merge RDBMS systems with the data model of XML, by overlaying XML upon existing relational databases, e.g. [7]. However, it has been shown that the performance of such systems is beaten by native XML databases in many cases [18].

Numerous native XML and semi-structured databases have been developed over the past decade, e.g., [11, 15]. As XML query languages standardize, we expect their popularity to increase. For such database systems to gain widespread acceptance, however, their performance must be notably better than relational databases. These databases share many similarities with the object oriented databases popular in the early 1990s, although their simpler data model should make their task considerably easier. In spite of this research, to the best of the authors' knowledge, the effect of clustering, a well-researched OODB topic, upon XML and semi-structured databases has never been investigated. Research into OODB clustering techniques has shown that effective clustering can improve the performance of some OODBs by up to several hundred percent [4]. Apart from the obvious effect on the number of disk accesses, clustering related objects together yields other performance improvements. For instance, better locality of reference can improve locking contention in highly concurrent systems. It could be expected that clustering would have a similar effect on XML databases.

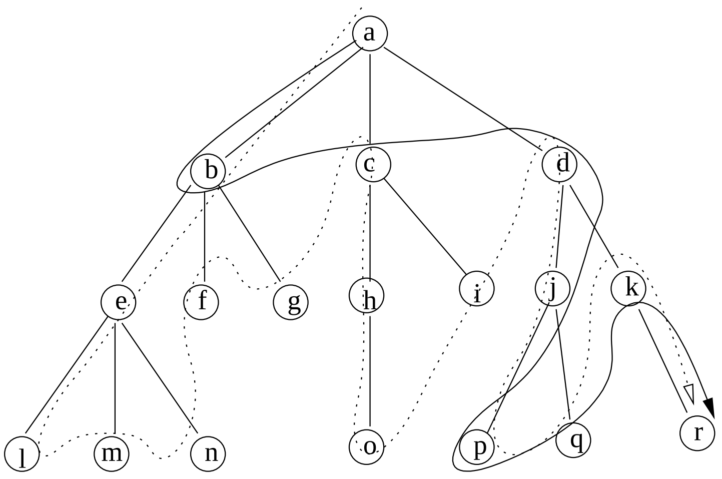


Figure 1: A sample XML database

As an example of the potential effect of clustering, consider the database shown in Figure 1. A typical XML database may load the data in using a SAX parser, and hence the nodes would be clustered in the order indicated by the dashed arrow. However, suppose we run the XPath query `/a/d//*` over this database. If the nodes were clustered in the order indicated by the solid arrow, it is clear that the average distance between consecutive nodes is smaller, because nodes which are not accessed by the query are stored in another part of the database. More dramatic improvements could be anticipated for real world database systems, which very often have large portions that are rarely queried.

While XML databases have many similarities to traditional object oriented databases, there are some differences which mean that OODB clustering strategies may not be utilized directly, or can be improved upon. The most important difference is that XML is ordered, whereas in object databases unordered sets of objects are common. This difference is particularly crucial as the most popular query languages for XML, XPath and XML Query [2, 6, 17], both deal with ordered queries explicitly. Thus, it is important that any clustering strategy takes into account the relative order between objects in the database.

A second significant difference is that XML databases store trees of objects, whereas object databases store general graphs. We note that the use of `ID` and `IDREF` types in a DTD allows the XML data model to represent general graphs; however, this is a second-class feature of XML in the sense that it is rarely seen in practice, and requires additional effort to use. Thus, ignoring the presence of `ID`s, we expect that many of the algorithms designed for object oriented databases could possibly be made considerably simpler, by restricting them to the case where the data is a tree.

In this paper, we present several alternative clustering strategies, and test their effectiveness in a simple native XML database system. We show that clustering does indeed have a positive effect on overall query time, and hence further research in this area would be useful. Section 2 covers the plethora of related work from OODB research. Section 3 lists five different proposed clustering strategies of varying degrees of complexity and characteristics, and some of which are adapted from object oriented clustering strategies. Section 4 presents our experimental results, and finally Section 5 concludes the paper.

2 Related Work

Clustering has been thoroughly investigated in both relational and object oriented databases. Relational clustering is relatively simple; generally, the clustering is performed along one or more attributes of the relation. If a single attribute is used, then the standard structure is a B-tree or one of its many variants. Multi-dimensional clustering is a little more involved, but there are well known data structures which perform quite well, such as the R-tree and its variants. A good survey of classic multi-dimensional access schemes is [8]. Relational clustering is not particularly relevant to XML database clustering, due to the fact that access patterns are considerably simpler.

OODB clustering is a far more difficult and well-researched problem. Given that XML databases can be thought of as fairly simple object oriented databases, it is clear that OODB clustering should be very applicable to XML databases. Generally speaking, the research into OODB clustering can be classified into *static* and *dynamic* strategies. While all static strategies are *offline*, dynamic strategies can be further partitioned into *offline* and *online* strategies.

2.1 Static Offline Clustering

Static schemes perform clustering using information available at database design time only, such as schema and type information. These schemes are typically very simple, and hence the clustering schemes of most commercial OODBs fall into this category. Unfortunately, due to their simplicity, there are many situations where they have only a negligible impact on performance, because they do not take advantage of any additional structure found in the database.

As mentioned above, many commercial databases employ simple static clustering schemes. GemStone [14], one of the first object oriented databases, provides the most trivial possible clustering algorithm, by allowing the user to specify which segment, which is simply a group of pages, of the database an object is to belong. Clearly, the fact that this algorithm is not automated imposes a significant burden on the developer of the database.

ObjectStore [13] provides another simple scheme, through the use of placement hints specified by the user. In practice, however, placement hints can sometimes have an only negligible impact on system performance. More importantly, placement hints can only be used at object creation time; this means that it is not possible to have a re-clustering tool for the database, because any information the user specified is meaningless after creation.

The O_2 database system [16] uses *placement trees*. When the database schema is unrolled, placement trees correspond to sub-trees in this schema. Objects whose type lie in the placement tree are clustered together. This is a powerful technique, but suffers from the significant drawback that human intervention is required to carefully specify the placement trees to be used.

The ORION database [12] provides another simple scheme: all objects of the same type are clustered together into the same database segment. The user may additionally specify whether two classes should be grouped into the same segment. Needless to say, this simplistic scheme also suffers from the same problems that the other commercial solutions suffer from.

2.2 Dynamic Offline Clustering

Dynamic offline clustering methods address the most serious problem with static clustering methods by taking advantage of patterns in the actual data to be clustered. Generally, this is done by maintaining some sort of statistical summary of object accesses. In practice, dynamic offline algorithms are particularly attractive, because whilst they adapt to the changing topology of the database, they do not generally impose a continuous overhead upon the normal operation of the database, as online algorithms do. Nevertheless, in some situations it is not possible to take the database offline for the significant periods of time the re-clustering process can take. Additionally, frequent updates to the database which occur between re-clusterings can negate any benefit that the re-clustering may have resulted in.

The first dynamic offline clustering scheme to be considered was that of Yue [22], who proved that the obvious clustering scheme based on decreasing probability of access was optimal under the assumption that accesses are independent and identically distributed. Of course, it has been made patently clear over the years that in real world databases systems accesses are very much correlated, and hence this scheme is generally sub-optimal in practice, sometimes by a significant degree.

So far, all of the commercial solutions have been static. The Cactis database [10] instead used a dynamic algorithm. Like most dynamic algorithms, the Cactis algorithm maintains a count of the references to each object, and the coreferences between every two objects. It then uses this information to periodically perform an offline re-clustering of the database, using a simple greedy algorithm. The advantages of a dynamic strategy are obvious; unfortunately, the storage overhead required for the statistical information is relatively high.

Cheng and Hurston proposed the leveled clustering algorithm [5]. Again, this algorithm takes as input the references and coreferences maintained as in previous algorithms. The algorithm is different from other clustering algorithms in that it considers a leveled clustering scheme, which takes advantage of the hierarchical nature of many OODBs. The algorithm also has the ability to work on segments of contiguous data.

The most effective dynamic clustering algorithm is stochastic clustering, proposed by Tsangaris and Naughton [19]. They model the clustering problem as a graph partitioning problem (where the graph is based on the reference and coreference information), which is NP-complete. They employ a heuristic method to find a good partitioning of this database. Unfortunately, the best heuristics have quadratic or worse time complexities; given that clustering can be performed on huge databases, this is unacceptable.

To alleviate this problem, a simpler partition-based approach was proposed by Gerlhof [3]. The idea is very similar to that of stochastic clustering, but a much simpler heuristic algorithm is employed. The experimental results of [3] show that this clustering strategy produces excellent results with relatively little time spent. Gerlhof's work also statically analyzes queries to infer possible clustering strategies.

Another clustering technique of significant interest to XML clustering is that of Banerjee et al [1], which studies the problem of clustering directed acyclic graphs, a category into which XML data clearly falls. The paper describes three traversal methods of DAGs, including the two traditional depth-first and breadth-first traversals, and examines the effect of clustering using these traversals.

2.3 Online Clustering

Online clustering algorithms generally build upon offline algorithms, by adding an incremental maintenance scheme. Wang [20] studies the problem of determining how to perform dynamic clustering, by considering various simple models of database activity. He offers several practical improvements which may mitigate the general performance hit of dynamic clustering. Nevertheless, even with this improvements dynamic clustering remains expensive.

McIver and King [21] propose a framework for online clustering, which divides the process into three components: the Statistics Collector, the Cluster Analyzer, and the Reorganizer, the functions of which should be obvious. Their clustering mechanism keeps track of the number of references to objects and coreferences between objects. Additionally, a distinction is made between *navigational* accesses (i.e., scanning a set of objects) and *materialization* accesses (i.e., accessing an object and all of its sub-objects). In the first case, the objects are sorted in a breadth-first traversal, in the second case, in a depth-first traversal. We will in fact make use of a similar scheme in the algorithm of Section 3.3. In addition to this basic clustering scheme (which is in fact offline), they use a simple analysis to determine whether it is worthwhile to re-cluster using the new scheme. The dynamic clustering is performed using triggers.

In practice, online clustering schemes are extremely difficult to implement in such a way that they actually provide a positive benefit. As there has been no work in offline clustering algorithms for XML, we instead choose to focus on these, and leave online clustering algorithms as an open, but difficult, research problem.

3 Proposed Clustering Strategies

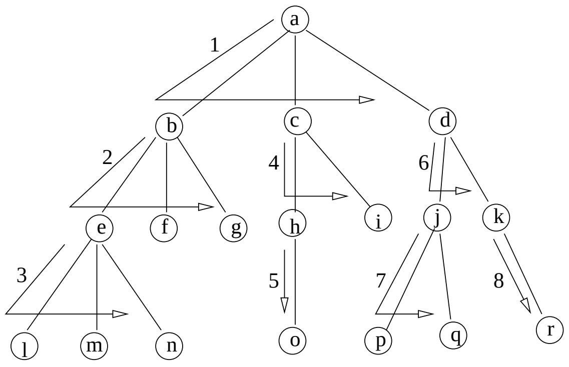
This section proposes several clustering strategies that are based on the availability of different database information, e.g., query statistics or schema information.

3.1 Scheme A: Hybrid Breadth-Search Clustering

Our first static clustering scheme is based on observations of real world XML documents such as MEDLINE and DBLP. The vast majority of large XML documents tend to have a very structured model, with a very broad breadth and shallow depth. In fact, many XML documents can easily be visualized as a large relational database; the key advantage to using XML in these cases is that each record has a high degree of irregularity. As an example of this phenomenon, DBLP is only five levels deep, but contains more than 300000 records (in fact, the vast majority of these records are only two levels deep). Therefore, we can expect that the number of children of each node is generally quite high.

As a result of the pseudo-relational nature of many XML documents, a possibly effective clustering scheme is to cluster in breadth-first order. The effectiveness of this strategy is predicated upon the assumption that the data is accessed like a relational database, with many horizontal scans of the records. However, we expect breadth-first clustering will tend to slow down materialization queries, such as `//*`, which will typically be executed when printing out portions of the database.

Figure 2 shows the order of traversal used by our clustering scheme. The arrows indicate the order in which the nodes are rearranged. Note that we do not use an actual breadth-first based ordering of nodes, but instead a hybrid between pre-order (document order) and breadth-first based ordering of nodes. We believe this strategy can perform well for fetching nodes that are closer to bottom of the tree,



```
Order of nodes after clustering:
  [a, b, c, d, e, f, g, l, m, n, h, i, o, j, k, p, q, r]
```

Figure 2: Static hybrid breadth-search based clustering of XML an database

without sacrificing too much for depth-based traversals. We also generally expect that when horizontal scans are performed, they will only be performed on the set of all children of a node, instead of the set of all nodes at a particular level. Hence, by grouping children together, instead of entire levels, we strike a balance between navigational and materialization accesses. The pseudo code for re-clustering the database using this strategy is described in Algorithm 1.

We suggest that there are two types of XML databases: *document*-centric and *data*-centric databases. This scheme is designed to work well with some data-centric databases, in particular, databases which have some properties of relational data.

Algorithm 1 Re-clustering of a database $\mathcal{D}_1$ into a new ordered set $\mathcal{D}_2$ based on hybrid breadth-depth based ordering

```
HYBRID-ORDER-RECURSE($n$)
1 $\mathcal{C} \leftarrow$ CHILDREN($n$) (in document order)
2 $\mathcal{D} \leftarrow \mathcal{C}$
3 for $c \in \mathcal{C}$ do (in document order)
4    APPEND($\mathcal{D}$, HYBRID-ORDER-RECURSE($c$))
5 end for
6 return $\mathcal{D}$

HYBRID-ORDER-RECLUSTER($\mathcal{D}_1$)
1 $\mathcal{D}_2 \leftarrow \emptyset$
2 APPEND($\mathcal{D}_2$, ROOT($\mathcal{D}_1$))
3 APPEND($\mathcal{D}_2$, HYBRID-ORDER-RECURSE(ROOT($\mathcal{D}_1$)))
```

3.2 Scheme B: Document-order Based Clustering

This clustering scheme is designed based on the requirements of current XML query languages such as XPath 2.0 and XML Query, where the result node set must be in document order. Thus, we might expect that a sensible approach for the query processor to return node sets in document order is to actually search through the document in document order.

Of course, if there were indexing support for document order built in to the database system, or the queries executed rely on other properties of the data (for instance, searching for records with an element whose numerical value lies within some range), then clustering in document order may not be the most efficient way of ordering. As mentioned earlier, we categorize XML databases into *document*-centric and *data*-centric databases. While in the latter case many queries will have access patterns not corresponding to document order, we expect that in the former case document order will be heavily relied upon. Even in the event that queries are executed on document-centric databases which do not follow document order, it is likely that the results of these queries will have to be materialized in document order. We propose that, in lieu of any further information, clustering an XML database in document ordering is a good default choice.

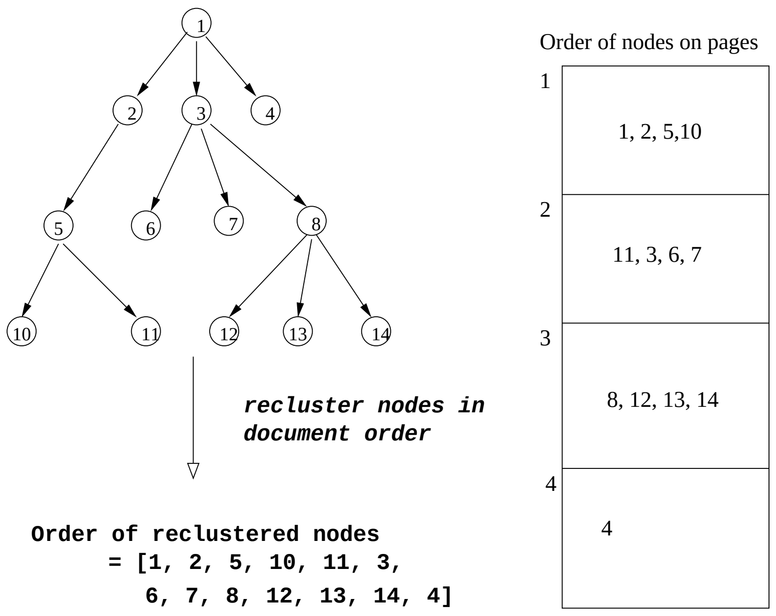

1
2
3
4
5
6
7
8
10
11
12
13
14
recluster nodes in document order
Order of reclustered nodes = [1, 2, 5, 10, 11, 3, 6, 7, 8, 12, 13, 14, 4]
Order of nodes on pages
1
1, 2, 5,10
2
11, 3, 6, 7
3
8, 12, 13, 14
4
4

Figure 3: Static document order based clustering of XML database

Figure 3 shows how a small database is rearranged in document order and how they are loaded on pages, assuming each page stores four objects. This clustering scheme is particularly effective at speeding up queries which contain the descendant operator (`//`). The pseudo code for the database into document order is given by Algorithm 3.2.

3.3 Scheme C: Schema-based Dynamic Clustering

The last two clustering schemes were both static, in the sense that they did not adapt to the query workload of the database. Our next algorithm is inspired by McIver and King's dynamic use of breadth-first and depth-first traversals [21]. Their algorithm keeps track of two quantities, *heat* and *tension*. The heat of an object is the number of references made to it, and hence is a measure of that object's popularity. The tension is the number of co-references made navigating from one object to another, and measures the correlation between accesses to different objects.

The novel insight of McIver and King was to split the heat metric into two values, the *navigational* heat and the *materialization* heat. Navigational heat measured the references that were made to an object during a navigation-like query, such as a scan over a set of objects. Materialization heat measured the references that were made to an object during a query which accessed an object and all of its sub-

Algorithm 2 Re-clustering of a database $\mathcal{D}_1$ into a new ordered set $\mathcal{D}_2$ based on document ordering

```
DOC-ORDER-RECURSE($n$)
 1 $\mathcal{D} \leftarrow \emptyset$
 2 APPEND($\mathcal{D}, n$)
 3 **for** $c \in$ CHILDREN($\mathcal{D}$) **do** (in document order)
 4    APPEND($\mathcal{D}$, DOC-ORDER-RECURSE($c$))

DOC-ORDER-RECLUSTER($\mathcal{D}_1$)
 5 **end for**
 6 **return** $\mathcal{D}$
```

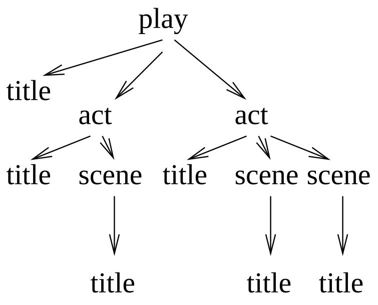


Figure 4: A small XML database

objects (for instance, to print the object on screen). Using these two different quantities, their algorithm then clustered the children of each object as follows: if the navigational heat was greater than the materialization heat, a breadth-first traversal of the children was used, otherwise, a depth-first traversal was used.

It could be expected that this OODB clustering algorithm will work quite well on XML databases, due to the nature of XML query languages. Most XML query languages, such as XPath, are built upon the concept of root-to-leaf path navigation. Hence, one expects that at each level of the data tree, depending on the type of query, either breadth-first or depth-first traversals will dominate. For instance, consider the sample XML database of Figure 4. If we executed the query `/play/act//title` over this database, then a traditional top-down XPath processor would perform a breadth-first traversal of the second level, and a depth-first traversal of all the sub-children of the `act` node.

Of course, McIver's algorithm has one particularly significant disadvantage: we must store a reference count for each node and edge. This overhead is typical of virtually all OODB dynamic clustering algorithms. We can alleviate this to a significant degree in an XML database due to the properties of real world XML data. We make the assumption that elements of an XML document which share the same "type" will most likely be accessed in similar ways. Hence, if we only store the relevant information per type, instead of per node, then we should be able to render the storage overhead of the clustering algorithm insignificant, whilst maintaining a reasonable degree of optimality.

While DTDs provide a standard method of defining types, much of the interest in XML data involves untyped documents, in which case type information must be inferred. To handle this case, we will make use of DataGuides [9], which provide a structural summary that can be efficiently computed and stored. A particularly appealing property of a DataGuide is that they mesh well with simple regular path expressions, whose navigation axes correspond to traversal of the DataGuide. Thus, we expect that if the statistical information is stored per DataGuide node instead of per database node, the loss in accuracy should be minimal, due to the fact that access patterns should remain relatively constant across the target set of a DataGuide node.

For each node of the DataGuide, our algorithm maintains a reference count, which is incremented every time an object in the target set of that node is accessed. We do *not* store coreference information, for several reasons. Firstly, in an XML database there are typically only a few access paths that can be taken from a node; this contrasts with OODBs where access paths can be far richer. As we are restricting our statistics to DataGuide nodes only, this is made even worse by the fact that sibling information is essentially lost. Hence, we do not expect that maintaining coreference information would add much accuracy to the algorithm.

To perform our re-clustering pass, we traverse the database in a single pass, beginning with the root node. We maintain a queue of nodes to visit; for each node, we examine its direct children, through whichever access paths the database implementation supports. We sort these nodes by the reference count of their DataGuide node, with ties broken by favoring depth-first traversals (such as accessing child nodes) over breadth-first traversals (such as accessing sibling nodes). We favor depth-first traversals because we expect these to occur more frequently in practice, however, we also do not believe that this choice substantially affects the performance of the algorithm.

Algorithm 3 Re-clustering a database $\mathcal{D}_1$ into an ordered set $\mathcal{D}_2$ using the schema based dynamic clustering algorithm.

```
DYNAMIC-SCHEMA-RECLUSTER($\mathcal{D}_1$)
 1 $\mathcal{D}_2 \leftarrow \emptyset$
 2 $\mathcal{Q} \leftarrow \emptyset$
 3 APPEND($\mathcal{Q}$, ROOT($\mathcal{D}_1$))
 4 while $\mathcal{Q} \neq \emptyset$ do
 5    $n \leftarrow$ REMOVE-FIRST($\mathcal{Q}$)
 6    $\mathcal{C} \leftarrow$ CHILDREN($n$)
 7    Sort $\mathcal{C}$ by decreasing DataGuide node
      reference count
 8    $c_1 \leftarrow$ REMOVE-FIRST($\mathcal{C}$)
 9    APPEND($\mathcal{D}_2$, $c_1$)
10    APPEND($\mathcal{Q}$, $\mathcal{C}$)
11 end while
12 return $\mathcal{D}_2$
```

Our algorithm is given in pseudo-code in Algorithm 3. At each step of the algorithm, the most expensive step is the sort. However, as we expect the number of nodes to be sorted to be very small, this step can be regarded as constant time. Hence, the entire algorithm runs in approximately one linear pass over the database, which is the best that can be hoped for.

3.4 Scheme D: Schema-based Static Clustering

The previous scheme was a dynamic clustering strategy which took advantage of schema information such as a DataGuide. In this scheme, we consider a static clustering strategy which takes advantage of schema information. While we cannot expect it to be as accurate as the previous scheme, we can expect the scheme to be considerably easier to implement, and faster to run. The scheme clusters the database by clustering by DataGuide type.

We assume each DataGuide node keeps a count of the number of nodes in its target set (this information is trivial to maintain). The algorithm proceeds by picking the DataGuide node that has the highest count of corresponding document nodes. For each DataGuide node, we cluster its target set by document order. Hence, the nodes are clustered in a hierarchy of clusters: at the top level, they are clustered by type, and then clustered by document order. Thus, this algorithm is an adaptation of the ORION database's clustering strategy [12] to ordered XML. This approach differs from the previous two static clustering strategies in that it assumes, like the previous schema algorithm, that access patterns for XML nodes remain approximately the same for all nodes of the same type. The algorithm is described by Algorithm 4.

Algorithm 4 Re-clustering of a database $\mathcal{D}_1$ into a new ordered set $\mathcal{D}_2$ based on static schema-based clustering

```
CLUSTER-DATAGUIDE-NODES()
 1 $\mathcal{D} \leftarrow \emptyset$
 2 for $n \in$ DATAGUIDE-NODES() do
   (sorted in decreasing order of access frequency)
 3    $\mathcal{S} \leftarrow$ TARGET-SET($n$)
 4    APPEND($\mathcal{D}$, DOC-ORDER-RECLUSTER($\mathcal{S}$))
 5 end for
 6 return $\mathcal{D}$

CLUSTER-OTHER-NODES($\mathcal{S}$)
 1 $\mathcal{V} \leftarrow$ ALL-NODES-IN-DATABASE()
 2 $\mathcal{D} \leftarrow \mathcal{V} - \mathcal{S}$
 3 $\mathcal{S} \leftarrow$ DOC-ORDER-RECLUSTER($\mathcal{D}$)
 4 return $\mathcal{S}$

RECLUSTER()
 1 $\mathcal{D} \leftarrow$ CLUSTER-DATAGUIDE-NODES()
 1 $\mathcal{S} \leftarrow$ CLUSTER-OTHER-NODES()
 1 APPEND($\mathcal{D}, \mathcal{S}$)
 2 return $\mathcal{D}$
```

3.5 Scheme E: Clustering based on Access Patterns

The previous dynamic clustering scheme in section 3.3 made approximations based on assumptions about standard XML query access patterns. In this approach, we remove these assumptions, and instead process data about access patterns directly. We store information from the most popular database queries and recluster the database system periodically, in order to optimize it for these popular queries. In between reclustering runs, we store the access paths of all queries. An example of this information follows:

```
firstchild 2 -> 3
firstchild 3 -> 4
firstchild 4 -> 7
firstchild 7 -> 9
next 9 -> 28
next 28 -> 42
```

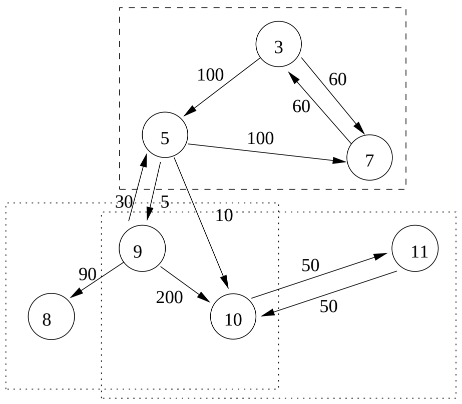


Figure 5: Clustering based on the node access paths

```
next 42 -> 72
...
parent 79298 -> 7
parent 7 -> 4
```

Clustering is then performed based on the frequencies of the access paths appearing in the *access log* (above), which can be visualized as a graph as shown in Figure 5, with the nodes of the graph being the nodes in the database, and the edges of the graph annotated with their access frequencies. In the figure, we assume each page can hold at most three nodes; if this is the case, then for the access patterns given, the following clustering is optimal according to most of the OODB clustering schemes [3]: $\{\{1, 3, 5\}, \{8\}, \{9, 10, 11\}\}$. Although this has proved to be effective on object databases, this clustering scheme is extremely expensive unless heuristics are employed.

In order to cluster vast amounts of XML data in a very short time, we simply partition the nodes by their access frequencies. Note that this has been proved to be optimal for database with independent and identically distributed accesses [22]. To compute the frequencies from the access log file efficiently, we combine the following steps and perform them at the same. First, let us assume the links in the access log denoted as $(x, y)_i$ (i.e., the i-th entry is *operation x* `->` *y*) and each page can hold P nodes.

1. Swap x and y if necessary so that $x \leq y$.

2. Sort the access log by (x, y), so that $(x_i, y_i) < (x_j, y_j)$ if and only if either $x_i < x_j$ or $x_i = x_j$ and $y_i < y_j$.

3. Scan the access log, merging duplicates by computing the frequencies of their occurrences (note that this can be performed at the same time with step 1 and 2 above).

4. Partition the nodes into pages of P nodes each based on their frequencies, using a simple greedy algorithm.

One aspect of this method is that we treated the graph as an undirected graph, even though the access paths are directed. Our reason for doing so is that, in terms of clustering, it makes little difference whether node A accesses node B or vice-versa; we are instead interested in the total number of coreferences between A and B.

We demonstrate our technique on the example of Figure 5. After the first two steps, the frequency table looks like:

```
x y  freq       x y  freq
3 7  60         3 7  60
7 3  60         3 7  60
3 5  100        3 5  100
5 7  100        5 7  100
9 5  30    ==>  5 9  30
5 9  5          5 9  5
5 10 10         5 10 10
9 10 200        9 10 200
...             ...
```

After the log is sorted and frequencies are calculated, the summarized log will look like this:

```
x  y  freq         9
9  10 200          10
3  7  120          3
3  5  100  ==>     7
5  7  100          5
10 11 100          10
...                ...
```

As described above, once we have this data we use a simple greedy algorithm to partition the nodes into pages. We note that, as proved in [22] under the assumption of i.i.d. accesses, this scheme is optimal under an LRU buffer management scheme. Since most operating systems and database systems use algorithms very similar to LRU for their buffer management, grouping nodes with high access frequencies together is likely to increase the effectiveness of LRU.

4 Experimental Results

We performed our experiments on a dual Pentium III 750MHz box with 256MB memory and a 50 GB 7200 rpm SCSI hard-drive. In order to increase the number of page swaps performed in the database kernel (so as to simulate very large databases, where we expect the effect of clustering to be greatest), we set the cache size to 1000 pages, where each page holds 8 kilobytes of data. Therefore, the experiment was effectively performed on a machine with 8 megabytes of memory.

For our experimental database, we used a cut-down version of the DBLP database, which had 50000 bibliographic records. The experiment was performed using the native XML database SODA4 from Soda Technologies (http://www.sodatech.com). However, we stress that none of our clustering approaches are tied to any particular implementation of query processors or database systems, and hence they will work equally well on other XML database implementations. To ensure that we were only measuring the effect of clustering, we turned off all database indices. We expect that our clustering schemes will work for databases with various indexing schemes, and even XML persistent repositories based on memory-mapped approaches which do not have any fast indexing, such as Java serialization of XML objects.

The data was first loaded into an empty database with a SAX-based XML parser, such that the XML data was stored according to the order of how the SAX-based parser performed the parsing, that is, as a pre-order traversal. However, most database repositories will generate and store some auxiliary information such as statistics, indexes or schema information. Therefore, the data nodes may not be physically stored adjacent to each other according to the document order. The query results for Raw in Table 1 and also in Figure 6 are obtained using this typical way of storing XML data.

Different queries, as described below, were then sent to and processed by the database system. The following shows some of the queries and their processing time according to the different clustering schemes. In particular, queries two through six were among the popular queries and were cached for use by two of our proposed clustering schemes (C and E), while the other schemes (A, B and D) clustered the data based on the schema and data distribution of the database.

Query 1 Find all the publications (Size of the answer: 50,000 records).

```
/dblp/*
```

Query 2 Find all the articles (Size of the answer: 533 records).

```
/dblp/article
```

Query 3 Find the first 100 publications (Size of the answer: 100 records).

```
/dblp/*[position() >= 1 and
        position() <= 100]
```

Query 4 Find all publications published by Springer between 1998 and 2002 (Size of the answer: 250 records).

```
/dblp/*[publisher='Springer'
  ][year >= 1998 and year <= 2002]
```

Query 5 Find all publications authored by Hector Garcia-Molina (Size of the answer: 14 records).

```
/dblp/*[author='Hector Garcia-Molina']
```

Query 6 Find all the co-authors (with duplicates) published with Hector Garcia-Molina (Size of the answer: 27 records).

```
/dblp/*[author='Hector Garcia-Molina'
  ]/author[.!='Hector Garcia-Molina']
```

Query 7 Find any school/institution information (if available) that is recorded in the database (Size of the answer: 77 records).

```
//school
```

Processing of any descendant queries such as the one shown above without any database indexes will naturally involve traversing all descendant nodes of the context node. Since the context node of the above query is the root node, all nodes would basically need to be visited. This explains why this is the most expensive query among the others as shown in Figure 6. Due to the fact that all objects were needed to be visited in this query, our proposed clustering schemes are relatively less effective (compared to that in the other 6 queries). However, the overall number of pages can still be reduced by clustering all related objects into the same page, and hence it is still better to cluster instead of storing the data in an arbitrary order (including the SAX-based style of storage as mentioned earlier).

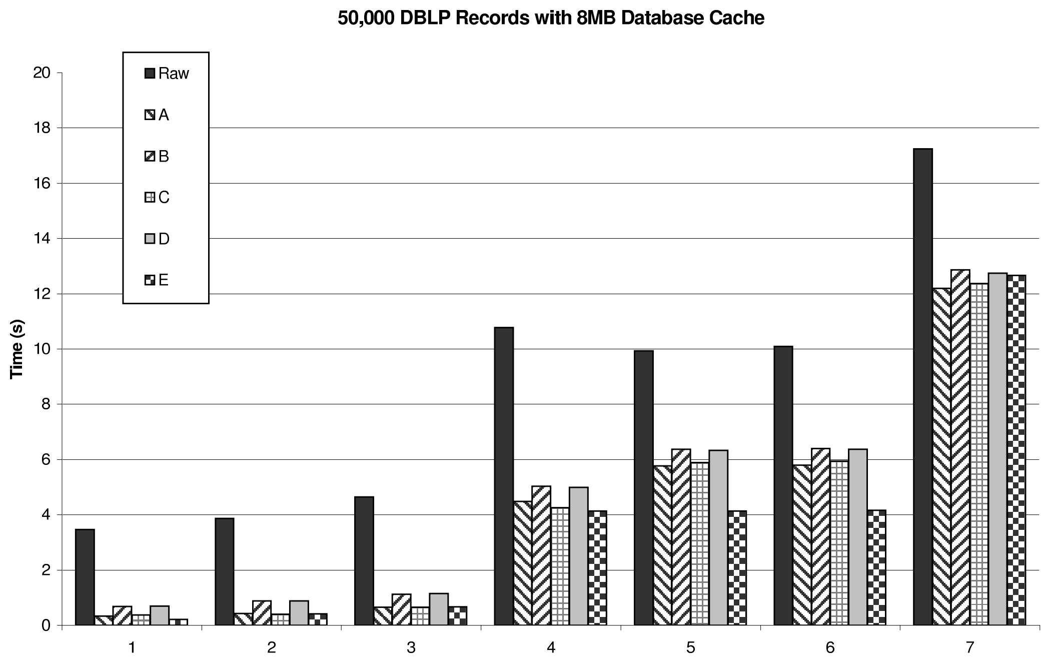


Figure 6: Results (in secs)

	Raw		A		B		C		D		E	
	Time	Δ	Time	Δ	Time	Δ	Time	Δ	Time	Δ	Time	Δ
Q1	3.47	1.00	0.34	10.21	0.68	5.10	0.37	9.38	0.70	4.96	0.21	16.5
Q2	3.87	1.00	0.43	9.00	0.89	4.35	0.40	9.68	0.89	4.35	0.41	9.43
Q3	4.65	1.00	0.66	7.04	1.13	4.12	0.66	7.05	1.15	4.04	0.67	6.94
Q4	10.77	1.00	4.48	2.40	5.04	2.14	4.26	2.53	5.00	2.15	4.13	2.61
Q5	9.93	1.00	5.77	1.72	6.37	1.56	5.89	1.69	6.33	1.57	4.13	2.40
Q6	10.10	1.00	5.80	1.74	6.40	1.58	5.94	1.70	6.37	1.59	4.16	2.43
Q7	17.24	1.00	12.19	1.41	12.87	1.34	12.37	1.39	12.74	1.35	12.66	1.36

Table 1: Detailed results

Figure 6 and Table 1 shows the performance of different clustering schemes. As shown in our experimental results, all of our clustering algorithms had improved the overall performance of each query over the typical data arrangement scheme (i.e., Raw). In particular, Scheme E performed significantly better than the other clustering schemes on some queries. This is due to the fact that it uses substantially more data than the other schemes to perform its clustering, and hence can produce considerably more accurate results. However, on simple queries, it is interesting to note that the relative difference between the clustering schemes is much less, which indicates that the assumptions made by the simpler static clustering schemes are appropriate for primitive XML queries.

Apart from Scheme E, the most consistent performer was Scheme C. Recall that the main assumption that this clustering scheme made was that access patterns remained the same for nodes which had the same DataGuide node. It is clear from our experimental results that this assumption seems to hold fairly well in practical queries. It is particularly interesting to note that Queries 4 and 5, where Scheme E comfortably beat Scheme C, are queries which do not just access structural information, but also data values (the author's name). This is exactly the kind of access path which Scheme C's assumption implicitly ignores.

The other three schemes used had relatively similar performance on all queries. This is somewhat surprising, as we expected the document ordering scheme (Scheme B) to outperform the other two simple static clustering strategies. Nevertheless, as can be seen from our results, the other two static schemes both consistently out-performed document ordering.

5 Conclusion

While clustering has been a well-researched topic in object oriented databases, there has not as yet been a transfer of knowledge into the relatively simpler domain of XML databases. In this paper, we have adapted some of the standard OODB clustering techniques to XML databases. We have investigated the applicability of various assumptions about standard access paths in XML.

We have shown that clustering has a dramatic effect on essentially all queries. Just as with OODB clustering, partitioning-based clustering (Scheme E) can perform dramatically better than other, simpler schemes, particularly as query complexity increases. On the other hand, as in OODBs, partitioning-based algorithms require substantially more data to achieve accurate results.

If the storage overhead required to run partitioning-based strategies is excessive , we have shown that by making some simple assumptions about XML access methods, we can adapt these schemes to achieve very respectable performance (e.g., Scheme C). We suspect that in practice it will be these schemes that are used, as the amount of data required for the re-clustering pass is an order of magnitude less than for the full partitioning-based strategies.

There are still many open questions left in this research area. Of most pressing interest is the effect of clustering on very complex queries, written in languages such as XQuery. Gathering empirical evidence in this case will be significantly more challenging, due to the fact that evaluating such queries will generally have to involve sophisticated query optimization plans. Another research area we intend to investigate is the use of dynamic online clustering algorithms to continuously adapt the database to its query workload.

References

[1] J. Banerjee, W. Kim, S-J. Kim, and J. F. Garza. Clustering a DAG for CAD databases. *IEEE Transactions on Software Engineering (SE)*, 14(11), November 1988.

[2] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (second edition). http://www.w3.org/TR/2000/REC-xml-20001006, 2000.

[3] Christoph Kilger Carsten A. Gerlhof, Alfons Kemper and Guido Moerkotte. Clustering in object bases. Technical Report 6, Universitität Karlsruhe, 1992.

[4] E. E. Chang and R. H. Katz. Exploiting inheritance and structure semantics for effective clustering and buffering in an object-oriented dbms. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 348–357. ACM Press, 1989.

[5] Jia-Bing R. Cheng and A. R. Hurson. Effective clustering of complex objects in object-oriented databases. In *Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, pages 22–31. ACM Press, 1991.

[6] W3C Working Draft. Xquery 1.0: An xml query language. *http://www.w3.org/TR/2002/WD-xquery-20021115*, November 2002.

[7] Mary Fernández, Yana Kadiyska, Dan Suciu, Atsuyuki Morishima, and Wang-Chiew Tan. SilkRoute: A framework for publishing relational data in XML. *ACM Transactions on Database Systems*, 27(4):438–493, December 2002.

[8] Volker Gaede and Oliver Gnther. Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231, 1998.

[9] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of VLDB*, pages 436–445, East Sussex - San Francisco, August 1998. Morgan Kaufmann.

[10] Scott E. Hudson and Roger King. Cactis: a self-adaptive, concurrent implementation of an object-oriented database management system. *ACM Transactions on Database Systems (TODS)*, 14(3):291–321, 1989.

[11] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A native XML database. *VLDB Journal: Very Large Data Bases*, 11(4):274–291, ???? 2002.

[12] W. Kim, J. F. Garza, N. Ballou, and D. Woelk. Architecture of the ORION next-generation database system. *tkde*, 2(1):109–124, March 1990.

[13] Charles Lamb, Gordon Landis, Jack Orenstein, and Dan Weinreb. The objectstore database system. *Communications of the ACM*, 34(10):50–63, 1991.

[14] David Maier, Jacob Stein, Allen Otis, and Alan Purdy. Development of an object-oriented dbms. In *Conference proceedings on Object-oriented programming systems, languages and applications*, pages 472–482. ACM Press, 1986.

[15] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Wid om. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[16] O. Deux et al. The story of O_2. *IEEE Trans. on Knowledge and Data Eng.*, 2(1):91, March 1990.

[17] W3C Recommendation. XML Path Language (XPath) Version 1.0. *http://www.w3.org/TR/xpath*, November 1999.

[18] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 302–314. Morgan Kaufmann, 1999.

[19] Manolis M. Tsangaris and Jeffrey F. Naughton. On the performance of object clustering techniques. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 144–153. ACM Press, 1992.

[20] Chao-Ming Wang. *Dynamic Online Data Clustering for Object-Oriented Databases*. PhD thesis, University of Illinois at Urbana-Champaign, 2000.

[21] Jr. William J. McIver and Roger King. Self-adaptive, on-line reclustering of complex object data. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM Press, 1994.

[22] P. C. Yue and C. K. Wong. On the optimality of the probability ranking scheme in storage applications. *Journal of the ACM (JACM)*, 20(4):624–633, 1973.