Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 60 (2007) 361–376
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

A clustering method based on path similarities of XML data ☆

Ilhwan Choi [a,*], Bongki Moon [b], Hyoung-Joo Kim [a]

[a] *School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Republic of Korea*
[b] *Department of Computer Science, University of Arizona, Tucson, AZ 85721, United States*

Received 31 October 2005; received in revised form 31 October 2005; accepted 8 February 2006
Available online 13 March 2006

Abstract

Current studies on the storage of XML data are focused on either the efficient mapping of XML data onto an existing RDBMS or the development of a native XML storage. Some native XML storages store each XML node in a parsed object form. Clustering, which means the physical arrangement of objects, can be an important factor in improving the performance in this storage model. In this paper, we propose a clustering method that stores data nodes in an XML document into the native XML storage. The proposed clustering method uses path similarities between data nodes, which can reduce page I/Os required for query processing. In addition, we propose a query processing method using signatures that facilitate the cluster-level access on the stored data to benefit from the proposed clustering method. This method can process a path query by accessing only a small number of clusters and thus need not use all of the clusters, hence enabling the path query to be processed efficiently by skipping unnecessary data. Finally, we compare the performance of the proposed method with that of the existing ones. Our results show that the performance of XML storage can be improved by using a proper clustering method.
© 2006 Elsevier B.V. All rights reserved.

Keywords: XML storage; Clustering; Path query

1. Introduction

The eXtensible Markup Language (XML) has become the de facto standard for data exchange on Internet. Recently, XML has become more widely used in various Internet applications, and the importance of XML repository has increased. XML is based on a new data model that includes structural information in the document itself. However, this new data model makes it difficult to store XML data in the existing relational or object-relational database systems that are widely used for data management. Accordingly, we need to modify the existing database systems or create a new XML database to manage XML data efficiently, and thus, many studies have been done on these issues [2,6–8,11,12,14,17,18].

☆ This work was supported in part by the Brain Korea 21 project and the Ministry of Information and Communications, Korea, under the Information Technology Research Center (ITRC) Support Program.
* Corresponding author.
E-mail addresses: ihchoi@oopsla.snu.ac.kr (I. Choi), bkmoon@cs.arizona.edu (B. Moon), hjk@oopsla.snu.ac.kr (H.-J. Kim).

0169-023X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2006.02.004

In previous studies, XML storages have managed XML data with various types of storages including file systems, relational database systems, object-oriented database systems and native XML database systems [6,8,12,14,17]. In contrast, current studies are more focused on either the storage of XML data using a relational database system or the creation of a native XML database system [2,11,18]. One of the storage techniques in the native XML database systems is to store XML data as a collection of objects. It parses an XML document to generate parsed objects from the nodes such as element and attribute, and then stores them as objects. In this storage model, the physical arrangement of objects can be an important factor in improving the performance of the system. In other words, the physical arrangement of XML data can affect the numbers of page I/Os required for query processing. The technique that considers the physical arrangement of the object is called *clustering*. In the object-oriented database system, many studies had been done on the clustering methods [4]. But few studies have been done on storing XML data. Most XML storages make little account of the clustering method when they store XML data. They just store XML data in a document order (Fig. 1). Therefore, in this paper, we studied various clustering methods for storing XML data more efficiently for the query processing purpose.

The contributions of this paper are as follows. First, we propose a clustering method called PSim clustering with which to cluster data nodes in a document in the native XML storage. Using this method, we can reduce page I/Os required for query processing. PSim clustering compares absolute paths of nodes so that the nodes on the similar paths are stored in the same cluster. But, first, we will introduce two simple clustering methods: one based on path and the other based on label. These two are simple intuitive methods, and each of them limits the minimum or the maximum size of the cluster for PSim clustering. Second, we propose a query processing method using signatures to benefit from the clustering methods for the processing of path queries. Using this method, we can process a path query by accessing as small number of clusters as possible. This method enables us to process a path query more efficiently because we can reduce the search space by skipping unnecessary data during query processing. Finally, we compare the performance of our proposed method with that of the existing ones. Our results show that the performance of XML storage can be improved by using a proper clustering method.

The rest of the paper is organized as follows: Section 2 discusses related work. In this section, we will discuss about the clustering issues, both on the XML storages and the object oriented database systems. Section 3 defines the data model and explains other background concepts, and then introduces two intuitive clustering methods. In Section 4, we propose PSim clustering, and Section 5 explains how it is applied to the query pro-

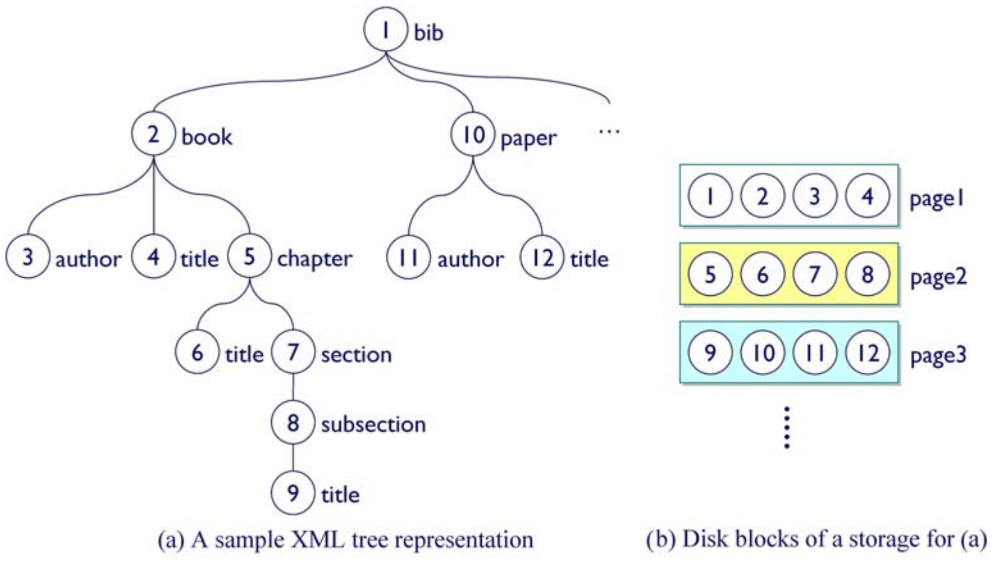


Fig. 1. Storing an XML document with a document order.

cessing. The results of our experiments are summarized in Section 6 and then conclusions are presented in Section 7.

2. Related work

Studies on clustering issues can be broadly classified into two approaches: one based on the structure of a document and another based on the content of a document. Ref. [1] proposed the well-known structure based clustering methods. It proposed simple clustering methods, such as a depth-first, a breadth-first and a hybrid method, based on the structure of a document. Among these methods, the depth-first method is similar to the document order method in XML. This method stores elements using the same order as in the original text XML files and many XML storages that make no account of clustering use this method to store an XML document. However, these approaches have a limitation considering the irregularity of XML data because they cluster data based on some fixed structural information.

Few studies had been done on clustering issues in the XML storage, but they mostly had content based approaches to support XML path queries more efficiently. A path in XML consists of the element labels called 'tag' and the content based approaches cluster XML data based on their tags. Ref. [19] compared the performance of five strategies for storing XML documents. They analyzed the difference in terms of performance of the five strategies from a clustering viewpoint, and presented three desirable clustering types when storing XML files: The first type is the clustering of elements corresponding to the same real world object. This was used in [17]. The second type is the clustering of the same kind of elements together. This type clusters elements that have common characteristics such as sharing the same tag name, having the parent–child relationship or the sibling relationship, etc. [8,17] clustered elements using this type. The third type is the document order method that is the structure based method as described in the earlier part of this section. Ref. [15] proposed two clustering methods based on DTD: The first method is storing elements whose node types are the same. This is similar to the second type reported in [19]. The second method in [15] is storing elements that are partitioned by their semantic blocks from the schema graph. Semantic blocks are obtained using the heuristic. But the heuristic is so simple that it does not perform outstandingly compared with the other clustering methods. These approaches can be considered as coarse-grained clustering methods that are simply based on tags.

On the other hand, [5] proposed a finer-grained clustering method. It proposed PathGuide, a clustered index based on the path suffix pattern. PathGuide can efficiently process complex path queries including ancestor–descendant relationship as well as simple queries by clustering elements with the same path pattern together. Unfortunately, PathGuide cannot benefit from its clustering method when the query is too complex, especially with a short suffix pattern. Since PathGuide is built based on the suffix of element's path, it has the advantage only when processing queries with a long suffix pattern that is a sequential parent–child relationship.

Other studies on clustering issues are approaches based on the user query patterns that had been studied in object-oriented database systems [9,20]. They found the relationship of objects for clustering from the user query patterns and clustered the frequently used nodes first. These methods showed better performance than the previous ones especially in processing frequently used queries [4]. However, these studies are not easy to apply to XML storage because while the information about the frequently used nodes is easily gathered (i.e. from the member function of an object) in object oriented database systems, it is not easy to obtain these informations in XML. And, it requires too much overhead to store every access information in nodes whenever processing a query.

3. Basic concept

In this section, we present a data model for XML documents and some basic clustering methods that will be used throughout in this paper.

3.1. Data model and concept definition

In this paper, an XML document is modeled as a rooted tree $T_d = (V, E, root, \lambda)$, $V = V_i \cup V_t$, where V_i is the set of nodes (both element and attribute) in the document and V_t is the set of text nodes in the document. Nodes from V are connected by edges from $E \subseteq V_i \times V$ to form a tree with a root node $root \in V$. The function λ assigns a label to each node in V_i. Fig. 1(a) shows an sample XML tree representation.

A *path* is a sequence of labels separated by /s. An *absolute path* of a node x is a path starting at the root of a given tree, $/l_1/l_2/\ldots/l_n$, such that we can traverse a path of $(n-1)$ edges from x to *root* where $\lambda(root) = l_1$ and $\lambda(x) = l_n$. Note that the terms 'path' and 'absolute path' are used interchangeably throughout this paper.

A *cluster* is a logical partition created by a clustering method. Let $C_1, C_2, \ldots, C_n$ be clusters created by applying a clustering method to an XML document T_d, $N(C_i)$ is the set of all nodes stored in a cluster C_i. Then all nodes V in T_d, $V = \sum_{i=1}^{n} N(C_i)$.

3.2. Basic clustering methods

In this section, we describe two intuitive clustering methods. These two methods play the role of minimum and maximum limitation of the cluster size in our proposed clustering method.

The first method is SL (same label) clustering where all nodes with the same tag name are stored in the same cluster. Data nodes in a cluster are stored according to their document order and each cluster has a label as a cluster identifier, which is the label of data nodes stored in it. As new clusters are allocated for each label, the number of clusters created after clustering is the same as the number of distinct labels in the document. This method is a simple intuitive method and thus can be easily implemented; for example, an inverted index from structural join can be used. Fig. 2(a) shows the result of SL clustering method that is applied to the XML document as shown in Fig. 1(a). In this figure, only two clusters with the labels 'author' and 'title', respectively are shown for simplicity.

The second method is SP (same path) clustering where all nodes with the same absolute path are stored in the same cluster. Data nodes in a cluster are also stored according to their document order as in the SL clustering method and each cluster has an absolute path, which is the absolute path of data nodes stored in it, as a cluster identifier. This method can be easily implemented using path index such as DataGuide [10] where nodes with the same absolute path are managed as one target set. As new clusters are allocated for each of absolute paths, the number of clusters created after clustering is the same as the number of DataGuide nodes, that is, the number of distinct target sets. Fig. 2(b) shows the result of SP clustering method that is applied to

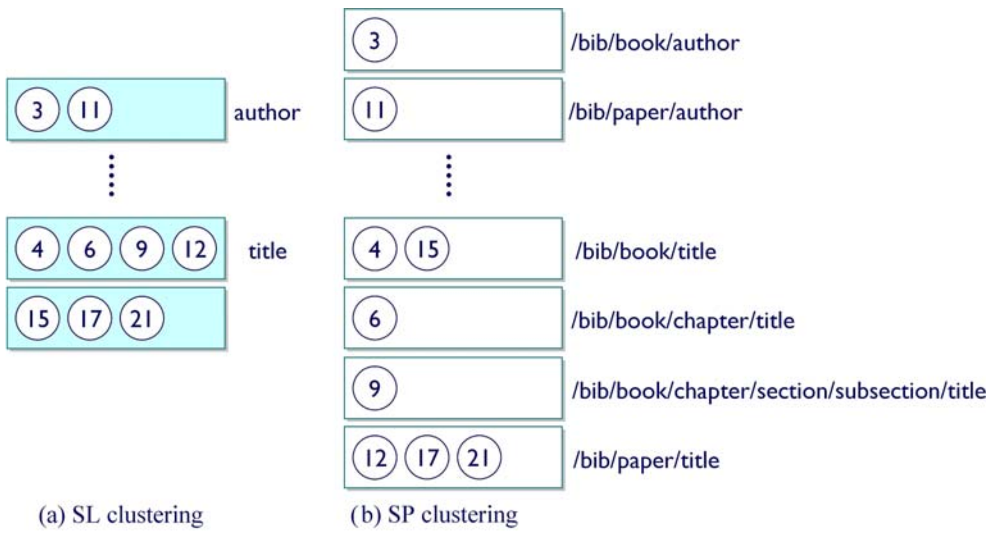


Fig. 2. SL clustering and SP clustering.

the XML document as shown in Fig. 1. Unlike SL clustering in Fig. 2(a), *node 3* and *node 11* are stored in different clusters because they have different absolute paths. Similarly, nodes with the label 'title' are stored in separate clusters according to their own paths.

3.3. Limitations of basic clustering methods

The difference between SL clustering and SP clustering is the granularity of clustering. SP clustering can perform finer grain clustering than SL clustering, because SP clustering stores nodes into several clusters according to their absolute path even if they have the same label while SL clustering stores them into one cluster. The difference in the granularity like this can affect query processing. For example, SP clustering is more efficient for processing a query like '*/A/D/X*' than SL clustering. If a document is stored by SP clustering method, only a SP cluster with identifier '*/A/D/X*' is returned. On the other hand, if the document is stored by SL clustering method, all elements that have label '*X*' are stored in the same cluster. In this case, it is difficult to guarantee that elements which have the absolute path '*/A/D/X*' will be stored closely, because nodes that have different absolute path are mixed in the same cluster. In the worst case, if we need *n* pages (disk blocks) for a SL cluster with identifier '*X*', elements having an absolute path '*/A/D/X*' might be stored scattered through *n* pages. Meanwhile, SL clustering is more efficient for processing a query like '*//X*' than SP clustering. If a document is stored by SL clustering method, only a SL cluster with identifier '*X*' is returned. On the other hand, if the document is stored by SP clustering method, all elements having label '*X*' are stored in a scattered mannered in several clusters.

As was mentioned above, SL and SP clustering method have a limitation, in that they are effective only on the specific type of query for each method. So we need more flexible clustering methods that are less dependent on the type of query. PathGuide [5] is one alternative that can overcome this limitation. It sets nodes whose absolute paths are the same as a base cluster and then groups clusters sequentially based on their path suffix. For example, assume there are three clusters *C1*, *C2* and *C3* whose absolute paths are '*/A/X/Y*', '*/B/X/Y*' and '*/A/C/Y*', respectively. PathGuide merges cluster *C1* and *C2* that have the suffix '*X/Y*' into a new cluster group *G1*. Finally, PathGuide completes its clustering by merging *C3* and *G1* which have the suffix '*Y*' into a new cluster group *G2*. Even after creating a new cluster group, base clusters and cluster groups that are merged into that group maintain their original path information in them. This makes it efficient to process a query which has the matching suffix with base clusters or cluster groups.

This PathGuide method can be regarded as the type that it has SP clusters as base clusters, and then creates a SL cluster by merging SP clusters. In that respect, it is a more flexible clustering method as it can take the advantages of both SL and SP methods. But PathGuide also has a limitation that it is effective only when the query has a long suffix pattern. It cannot benefit from its clustering method when the query is too complex, especially with a short suffix pattern because of its suffix based nature. In the previous example, if a query '*//X/Y*' is requested, PathGuide can easily process it by scanning *G1*. But if another query '*//A//Y*' is requested, PathGuide must scan *G2*, which is the biggest cluster unit, to process it. In particular, even if a specific query such as '*/A/B/C/D/E//Y*' is requested, PathGuide scans a large cluster group that have a suffix '*Y*'.

Considering all these limitations, we suggest a more flexible clustering method, *PSim clustering method*, which can process various types of queries efficiently and includes all of the advantages of the basic clustering methods, namely the SL and SP clustering. PSim clustering performs merging based on the path similarity of nodes, contrary to PathGuide where merge is done based on the path suffix. This enables PSim clustering to be less dependent on the type of query.

4. PSim clustering

Data nodes stored in a SL cluster can be divided again into one or more SP clusters according to their absolute path. Therefore, a simple method of securing the advantages of both SL and SP clustering is to apply SP clustering to the data nodes in a SL cluster after applying SL clustering to an XML document; SP clusters whose identifiers end with the same label are grouped together. In this case, the efficiency of clustering depends on the arrangement order of SP clusters in a SL cluster. For example, assume there are three SP clusters *sp1*, *sp2* and *sp3* whose identifiers are '*/A/B/C*', '*/X/Y/C*' and '*/D/B/C*' each. When a query '*//C*' is requested, all data stored in *sp1*, *sp2* and *sp3* are returned. In this case, the arrangement order of SP clusters has no effect on the page I/Os required for query processing. On the other hand, when a query '*//B/C*' is requested, only the data stored in *sp1* and *sp3* are returned. So, in this case, it is more efficient to arrange *sp1* and *sp3* as close as possible. Accordingly, to process various types of queries flexibly, it is important to arrange SP clusters properly in a SL cluster. Our method, PSim clustering, compares the identifiers of SP clusters, and then arranges SP clusters with similar paths very close to one another by merging them into a new cluster. In brief, the process of the PSim clustering method can be understood as follows: first, data nodes in a SL cluster are divided into SP clusters, and then rearranged by merging them properly. By doing this, we can secure the advantages of both SL and SP clustering and thus can process various types of path queries.

4.1. Base unit

PSim clustering uses a SP cluster instead of a data node as a base unit for clustering. This means that every node which has the same absolute path is always stored in the same cluster in the PSim clustering method. The maximum cluster unit for PSim clustering is restricted to the label of a data node. That is, similarly to SL clustering, data nodes with different labels are stored in separate clusters.

4.2. Path similarity

PSim clustering merges SP clusters based on their path similarities. Although it has too much overhead to calculate path similarities between all data nodes, PSim clustering has less overhead because it uses a SP cluster as a base unit and it compares the identifiers of SP clusters instead of the paths of data nodes. In general, since one-to-many data nodes are stored in a SP cluster, the number of SP clusters is much fewer than the number of data nodes.

Each SP cluster has its own unique absolute path as an identifier, and thus, we can compute a *path similarity* between two SP clusters by comparing their identifiers. To compare identifiers, we use the edit distance algorithm [13] that is generally used to compare strings except that we use a path instead of a string and labels in the path instead of words in the string. Given two absolute paths $A = /a_1/a_2/\ldots/a_n$ and $B = /b_1/b_2/\ldots/b_m$, the edit distance $\delta(A,B)$ between these two paths can be computed as follows using a dynamic programming technique [22]. We shall write A_i for $/a_1/a_2/\ldots/a_i$, B_j for $/b_1/b_2/\ldots/b_j$ and $\delta(A_i,B_j)$ for $\delta_{i,j}$. Now we construct a $(n+1)\times(m+1)$ edit matrix $D = (d_{i,j})$ with indices running from 0 to n and from 0 to m. The first row and column are simply given by $d_{0,0} = 0$, $d_{0,j} = \delta(\varepsilon, B_j)$, $d_{i,0} = \delta(A_i, \varepsilon)$. Then, the elements in the edit matrix are computed using the following equation:

$$d_{i,j} = \min(d_{i-1,j-1} + \delta(a_i,b_j), d_{i-1,j} + \delta(a_i,\varepsilon), d_{i,j-1} + \delta(\varepsilon,b_j)), \quad (1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m)$$

After computing all elements in the edit matrix, we can get the edit distance $\delta(A,B) = d_{n,m}$. Finally, using the edit distance between two absolute paths, a *path similarity* between two SP clusters, *sp1* and *sp2* whose identifiers are *id1* and *id2*, respectively is computed as follows:

$$path\_similarity(sp1,sp2) = 1 - \frac{\delta(id1,id2)}{\max(path\ length\ of\ id1, path\ length\ of\ id2)}$$

If *id1* and *id2* are the same, $\delta(id1,id2)$ will return 0 and $path\_similarity(sp1,sp2)$ will be 1. On the other hand, if *id1* and *id2* have no common parts on their path, $\delta(id1,id2)$ will be the same as $max(path\ length\ of\ id1, path\ length\ of\ id2)$ and $path\_similarity(sp1,sp2)$ will be 0.

4.3. PSim clustering algorithm

Path similarities between SP clusters can be represented as a weighted graph $G = (V,E)$, where V is the set of SP clusters, E is the set of edges connecting SP clusters and the weight of each edge is the path similarity between two nodes which are connected by that edge. We call this graph as a *path similarity graph*. Now, PSim clustering can be considered as the graph partitioning problem of this path similarity graph. For an XML document, there exist path similarity graphs for each distinct label in the document. So, PSim clustering is done by

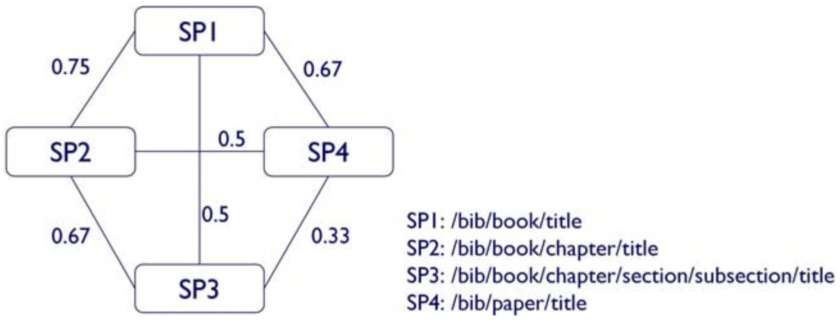

SP1
SP2
SP3
SP4
0.75
0.67
0.5
0.5
0.67
0.33
SP1: /bib/book/title
SP2: /bib/book/chapter/title
SP3: /bib/book/chapter/section/subsection/title
SP4: /bib/paper/title

Fig. 3. Path similarity graph.

performing partitioning on every path similarity graph. Fig. 3 shows a sample path similarity graph for the label ‘title’ in Fig. 1(b) (SP clusters in the graph can be seen in Fig. 2(b)).

The graph partitioning problem is an NP complete problem and it is normally solved by using a heuristic. In this paper, we used the greed algorithm as in [9] to partition the path similarity graph. This PSim clustering algorithm for a path similarity graph of a label is shown in Fig. 4. The algorithm proceeds with the partitioning process sequentially from the edge with the highest weight in the sorted edge list in order to consider the most similar nodes first (*lines 2–14*). First, it gets a pair of sp nodes that are connected by the selected edge (*line 3*). Next, it finds each cluster where the selected nodes are stored. If there is no cluster where the node is stored, that is, if it is not clustered yet, a new cluster is allocated for that node (*lines 4–9*). Then it tries to create a new cluster by merging two clusters (*lines 10–11*). Note that the size of a cluster is restricted so as not to exceed a page (disk block) as the benefit in page I/O is no longer meaningful in that case (*line 10*). Finally, when the weight of the current edge is below the pre-defined threshold value and the remaining (those which have not been clustered yet) nodes can be stored in a page, the process is discontinued and the algorithm is completed after storing them in a new cluster (*lines 12–14*). This prevents nodes from being fragmented into numerous pages by grouping the remaining nodes that have relatively less similar paths.

```
Algorithm PSim(node sl_node)
01 list elist = a node pair list sorted by weight with descending order
                  from the path similarity graph for sl_node;
02 for each edge e in elist do
03     <sp1, sp2> = a pair of vertices which are connected by e;
04     if (there exists a cluster where sp1 is stored)
05        c1 = the cluster where sp1 is stored
06     else allocate a new cluster to c1 and store sp1 in it;
07     if (there exists a cluster where sp2 is stored)
08        c2 = the cluster where sp2 is stored
09     else allocate a new cluster to c2 and store sp2 in it;
10     if (c1 != c2 && the size of c1 + the size of c2 < PAGESIZE)
11        merge c1 and c2;
12     if (the weight of e < THRESHOLD
               && the size of unclustered sp nodes < PAGESIZE)
13        create a new cluster c3 and store unclustered sp nodes in it;
14        break;
```

Fig. 4. PSim clustering algorithm.

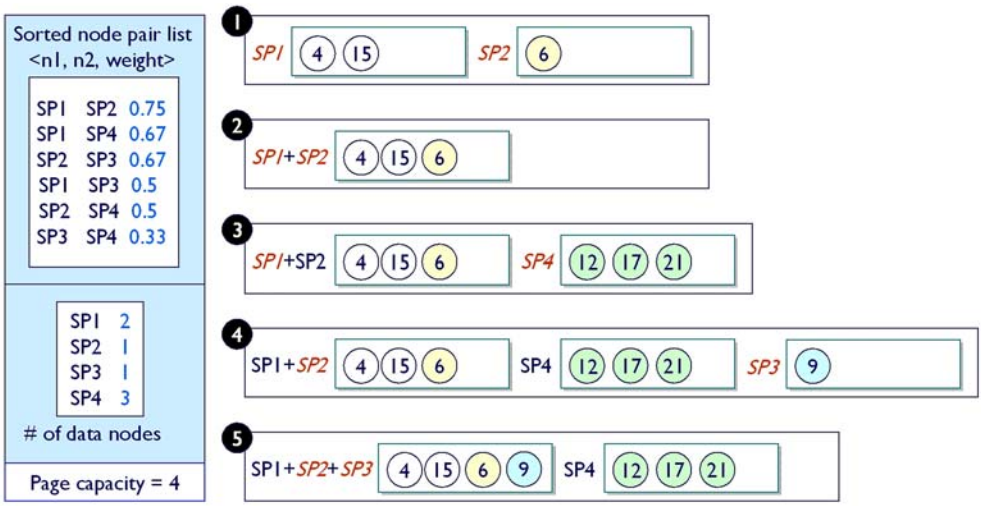


Fig. 5. The execution process of PSim clustering.

With this algorithm, the execution process of PSim clustering over the pass similarity graph of Fig. 3 is shown in Fig. 5. In the figure, for the node pair $\langle SP1, SP2\rangle$ which has the highest weight, two clusters [SP1] and [SP2] are created (step 1 in Fig. 5) and merged (step 2 in Fig. 5). Note that elements in the SP cluster are from Fig. 2(b). Then the algorithm proceeds to the node pair $\langle SP1, SP4\rangle$ which has the next highest weight in the sorted node pair list. In this case, only one new cluster [SP4] is created because the node SP1 is already clustered in the cluster [SP1, SP2] (step 3 in Fig. 5). Note that two clusters [SP1, SP2] and [SP4] are not merged. If two clusters [SP1, SP2] and [SP4] are merged, the size of the result cluster [SP1, SP2, SP4] will be 6. This exceeds a page size (which is 4 in Fig. 5). So those two clusters are not merged. The algorithm proceeds to the rest of the node pairs in the list likewise.

4.4. Update issues

When an update occurs, if there is no schema change, the update overhead of the PSim method is same as the SP method and PathGuide because they use a SP cluster as the base unit of the clustering. Therefore, the update process is to find an appropriate cluster among existing ones as follows: (1) Find the cluster where the original data is stored, and delete the original data. (2) Find the SP cluster that have the same absolute path as the updated data, and store the data in it. In this case, the update overhead of the SL method is smaller than other methods. It considers only the labels of elements instead of their individual paths, and thus, the update process can be simplified as to find the cluster with the corresponding label.

On the contrary, if there is a schema change, the update overhead of the PSim method is more complex than other methods. If the updated data has a new label that no existing elements have such a label, all methods have the same overhead as the data is simply stored in a new cluster. But if the updated data has a new absolute path that no existing elements have such a path but the label of the data is same as the already existing one, then the update process of the PSim method is as follows: (1) Create a new SP cluster for the updated data. (2) Find SP clusters with the same label as the updated data. (3) Choose the SP cluster that has the highest path similarity value with the newly created SP cluster. (4) Merge the new SP cluster into the PSim cluster where the chosen SP cluster is stored. In this case, the update overhead of the SP clustering method is smaller than other methods. It simply stores the updated data in a new cluster just as it has done when the updated data has a new label. In the case of the SL method, the update overhead is same as when there is no schema change because the cluster with the same label as the updated data already exists. Finally, the update overhead of the PathGuide is affected by how to choose the group that has the same suffix as the updated data.

5. Query processing using PSim

When a query is requested, in many cases, data nodes related to that query can exist in the small part of the whole document. So if we can access only the parts of the data that we need, the query processing can be conducted more efficiently because the search space is reduced by skipping unnecessary data during query processing.

Fortunately, a cluster, which is the unit of logical partition created by clustering, offers the foundation which makes it easy to access partial data. That is, if we can select only the clusters that are required to process a query, the query processing phase can be completed simply by scanning only those data in the selected clusters instead of each and every data in the document. The PSim clustering method partitions a document based on the path of data. This makes it easy to select needed clusters when processing XML queries where path queries are used. So, if we know the path information of data stored in each cluster, we can easily select the relevant clusters to the requested path query. Therefore, in this section, we propose a query processing method using signatures that facilitate the cluster-level access on the stored XML document to benefit from the proposed clustering method.

5.1. Cluster signature

A *cluster signature* is a hint as to the paths of data nodes stored in a cluster. Each cluster is allocated with one. It helps to decide if data nodes with a particular path exist or not. So when a query is requested, we can select clusters with the relevant path information to the query using the signature.

A cluster signature is a bit string whose length is the number of distinct labels in the document. Each bit of a signature corresponds to a label. For example, assume there are n distinct labels $l_1, l_2, \ldots, l_n$ in a document. Then a cluster signature for this document becomes an n-bit length string and the first bit corresponds to the label l_1, the second bit corresponds to the label l_2 and the nth bit corresponds to the label l_n. If a data in the cluster includes a label l_x in its path, the xth bit of the cluster signature for this cluster is set to 1. So if a data node has m distinct labels on its path, a total of m bits of the cluster signature are set to 1. Finally, by executing this bit setting procedure for all of the data nodes in the cluster, the generation of the cluster signature for that cluster is completed. Likewise, a cluster signature represents the label information on the paths of all data nodes stored in a cluster.

A cluster signature can be generated with less overhead using SP clusters which organize a PSim cluster instead of using real data nodes. First, we generate a SP cluster signature for each SP cluster. Since a SP cluster has an absolute path as an identifier, we can generate a SP cluster signature by setting the bits that corresponds to all labels on the identifier. This is just like the generation process of the PSim cluster signature. Now, a PSim

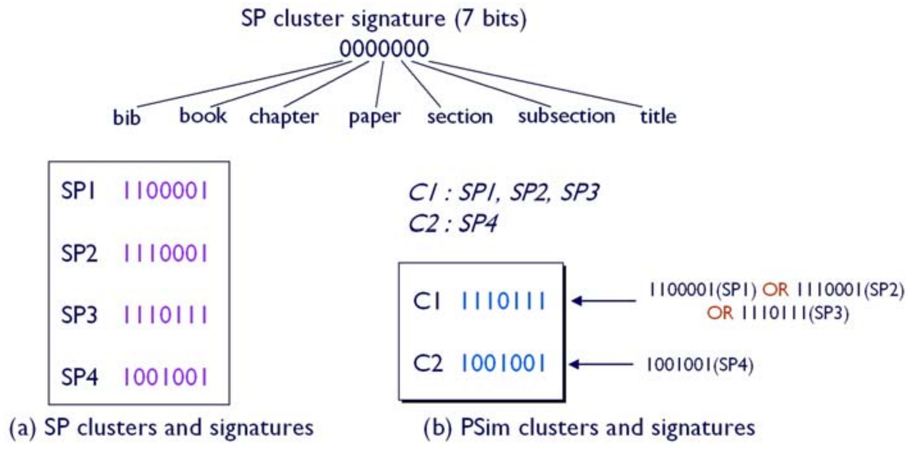


Fig. 6. Cluster signatures.

cluster signature can be generated by bitwise ORing all signatures of the SP clusters stored in a PSim cluster. Fig. 6 shows an example of PSim cluster signatures being generated from SP cluster signatures. In this figure, SP clusters and PSim clusters are those in Fig. 5. For simplicity, assume there are only seven labels in the document. Then, a cluster signature becomes a 7-bit string. Fig. 6(a) shows SP clusters with the label 'title' and their signatures. And Fig. 6(b) shows the generation process of PSim cluster signatures.

5.2. Cluster selection

Cluster signatures allocated to PSim clusters are stored in a separate index and used to select clusters needed after a query is requested. When a query is requested, a *query signature* for that query is generated. A query signature is also generated by setting the bits which correspond to the labels on the query string as in the cluster signature. Now, we can select the needed clusters by bitwise ANDing the query signature with each of the cluster signatures of PSim clusters. If there is a cluster whose result string of the bitwise AND operation is the same as the query signature, it is selected as a candidate cluster that will be scanned during query processing. In this case, for every bit position whose value is 1 in the query signature, the value of the same bit position in the cluster signature is also 1. This means the candidate cluster is a cluster that includes all labels on the query string and thus may have the result data for the query.

For example, let us assume that a query *Q1*:'*//book//section//title*' is sent to the document in Fig. 6. Then a query signature '*0100101*' is generated for this query and the results of bitwise ANDing the query signature with each of cluster signatures '*1110111*' and '*1001001*' are '*0100101*' and '*0000001*' respectively. So the cluster *C1* is selected as a candidate cluster because it has the same result string as the query signature: '*0100101*'. Now, the query processing is done only on the candidate cluster *C1*. Cluster *C2* is not considered anymore as the query processing proceeds. In Fig. 6, we can also confirm that '*/bib/book/chapter/section/subsection/title*', which is the result of $Q1$, exists only on cluster $C1$.

Likewise, the cluster selection using the cluster signature has the advantage in that the search space can be reduced by skipping unnecessary data during query processing. However, since the cluster signature is generated without considering the order of labels on the path, unnecessary clusters that have no relevance to query may be selected as candidate clusters. For example, when a query '*//A/B//C*' is requested, a cluster which includes data nodes with the path '*/B/A/C*' can be selected as a candidate cluster. To avoid this, we can also take the ordering into consideration by maintaining additional information in addition to the signature. For example, we can maintain an order list among signature bits or even generate an integer array as a signature where each slot of the array implies the ordering information. However, additional overheads are incurred to maintain and process this ordering information. What is worse, even if we use signatures which take the ordering into consideration, unnecessary clusters still can be returned as candidate clusters. But consider the purpose of using the signature in this paper is to reduce the search space required to process a query by selecting 'candidate clusters' rather than 'exact clusters'. From this point of view, it may rather cause an unnecessary overhead to use the signature with the ordering information, and thus, we decided to use the proposed signature method that is simple but considers no ordering information. The experiment that will be discussed in the following section shows that in many case the proposed method can significantly reduce the search space of data.

6. Performance analysis

In this section, we compare the performance of the PSim clustering method with other clustering methods mentioned in this paper. Four other methods were compared in this section: 'RAW' which stores XML data in a document order, two basic clustering methods 'SL' and 'SP' of Section 3.2, and 'PG' which means Path-Guide [5].

6.1. Experiment environment

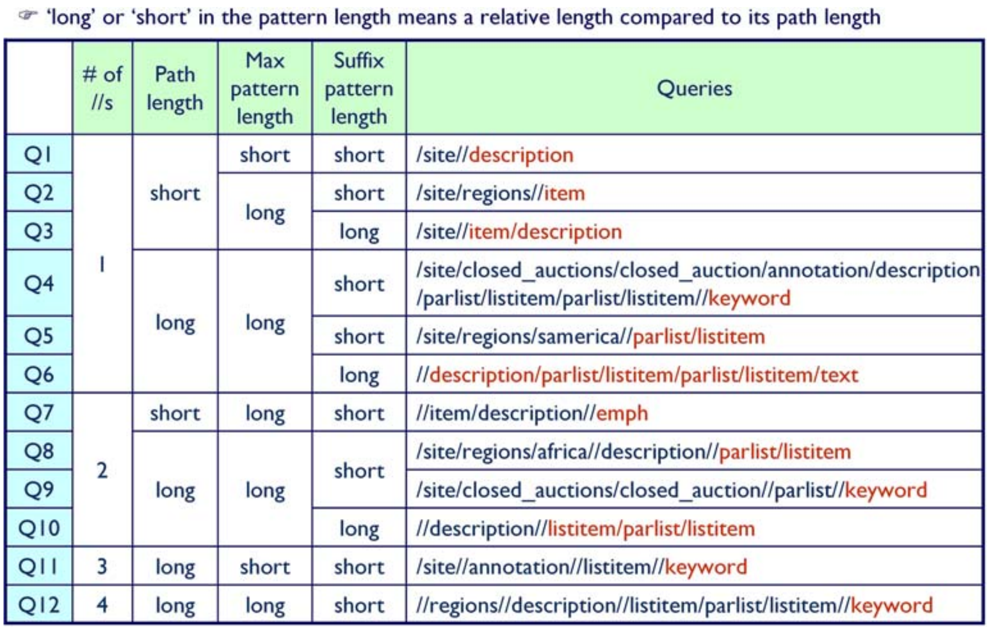
We used an XML document generated by XMark [16] and the document had 200,000 elements. For queries, we used 1000 randomly generated path queries with '//', which means the ancestor–descendant relationship. In our experiments, all XML documents were stored in XDBox [21], which is the object based XML storage. XDBox stores XML data nodes as a collection of objects and supports XPath [3] query language. We set the page size as 4 KB and the size of buffers as 20.

6.2. Performance analysis of various clustering methods

We first evaluated the performance of various clustering methods on the queries in Fig. 7. Queries in Fig. 7 show various patterns in their path lengths, pattern lengths, suffix lengths and numbers of the ancestor–descendant relationship. Q1, Q2 and Q3 are simple queries with short path lengths. Q4, Q5 and Q6 are queries with relatively long path lengths. Among them, Q6 is a query with a very long suffix pattern. On the other hand, Q4 is a query with a very long prefix but a short suffix pattern. More complex queries with two or more ancestor–descendant axes are shown in the latter part of query samples, from Q7 to Q12. Fig. 8 shows the page I/Os required to process queries in Fig. 7. Each query was executed on the same five XML documents where each of the documents was clustered using the respective clustering methods: RAW, SL, SP, PG and PSim.

In this experiment, PSim showed the best performance (with the fewest page I/Os) for all queries except Q1, Q3, Q6 and Q10. Among them, Q1 is a simple query to find all elements with the label ‘description’. So, both PG and PSim, which have ‘label’ as the largest clustering boundary, showed the same page I/Os as SL. But SP showed more page I/Os than other methods as we expected in Section 3.3. Q6 and Q10 are queries with relatively long suffix patterns and PG showed the best performance with these types of queries. Q3 is a query with the suffix pattern ‘item/description’, but unlike Q6 or Q10, PG did not show a better performance than PSim because the query length was too short and thus could not have a specific suffix pattern in it. In particular, PSim showed an outstanding performance for queries Q4, Q5, Q8 and Q9. These are queries that have a relatively long prefix and a short suffix pattern. This result showed that PSim could efficiently cluster elements with common patterns (in this case, a long prefix) independent of the location where the common patterns are, while PG clustered elements with common patterns only in the suffix parts of their paths. As for basic methods, SP showed a better performance than SL when the query has a long path pattern in it.

Next, we executed our experiment on the randomly generated path queries (Fig. 9). Total 1000 random queries were used where from one to five //s are included (200 queries respectively for each of the number of //s included in a query). In the figure, the *x*-axis shows how many //s are included in a query, that is,


☞ ‘long’ or ‘short’ in the pattern length means a relative length compared to its path length

	# of //s	Path length	Max pattern length	Suffix pattern length	Queries
Q1	1	short	short	short	/site//description
Q2			long	short	/site/regions//item
Q3				long	/site//item/description
Q4		long	long	short	/site/closed_auctions/closed_auction/annotation/description /parlist/listitem/parlist/listitem//keyword
Q5				short	/site/regions/samerica//parlist/listitem
Q6				long	//description/parlist/listitem/parlist/listitem/text
Q7	2	short	long	short	//item/description//emph
Q8		long	long	short	/site/regions/africa//description//parlist/listitem
Q9					/site/closed_auctions/closed_auction//parlist//keyword
Q10				long	//description//listitem/parlist/listitem
Q11	3	long	short	short	/site//annotation//listitem//keyword
Q12	4	long	long	short	//regions//description//listitem/parlist/listitem//keyword

Fig. 7. Query samples used in our experiments.

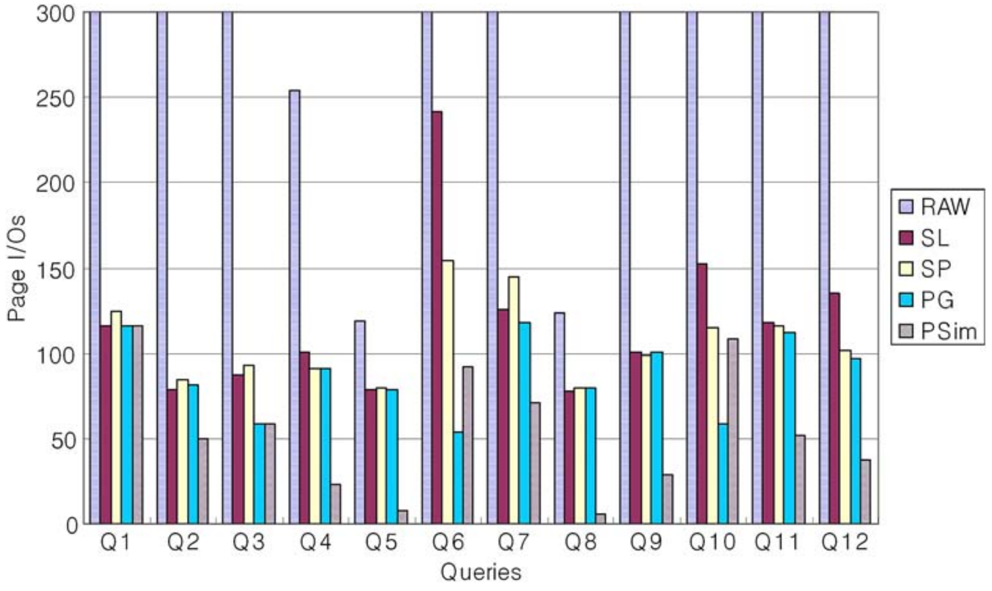


Fig. 8. Performance analysis for sample queries.

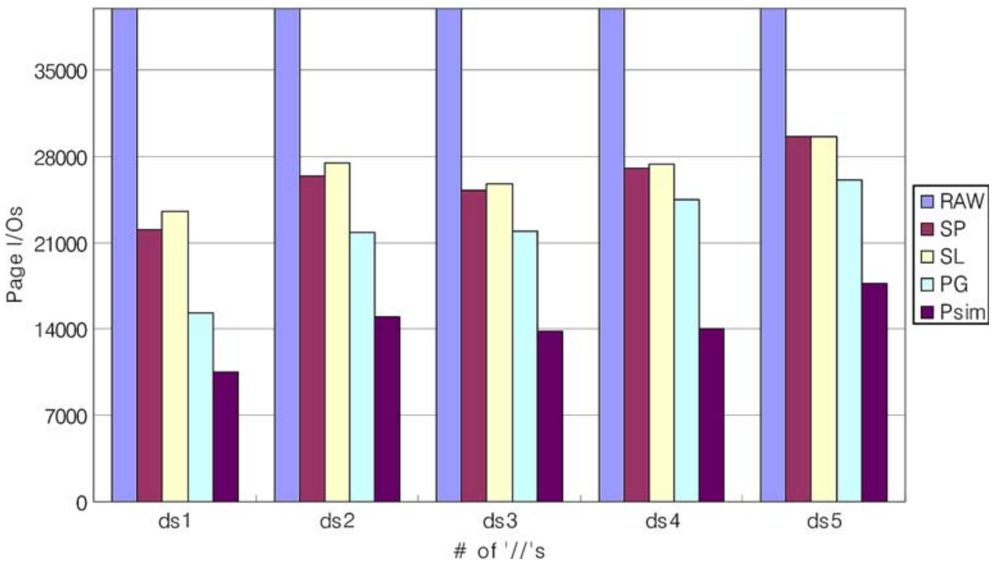


Fig. 9. Performance analysis for random queries.

the number of the ancestor–descendant relationship in a query. And the y-axis shows the sum total of page I/Os required for query processing.

In this experiment, under RAW method, where no clustering method was applied to, over 10 times more page I/Os occurred than under other methods. This confirms that any of the above-mentioned clustering methods can improve the performance of the XML storage for the query processing purpose. In the basic clustering methods, in the case of small number of //, SP showed better performance than SL as we expected in Section 3.3 because queries still had long path patterns in them. On the other hand, as the number of // increased, SL, which is a course grained clustering method, showed better performances as we also expected in Section 3.3. Among the various clustering methods, PG and PSim which are the advanced clustering methods showed outstanding performance. This is due to the fact that the advanced clustering methods secure the advantages of the basic methods and furthermore overcome the limitations of the basic ones. In particular,

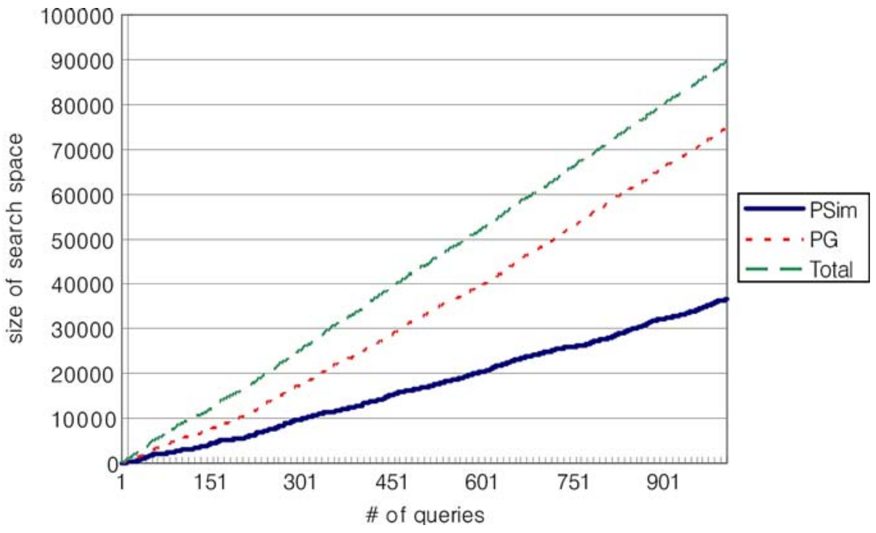


Fig. 10. The size of the search space required for query processing.

PSim showed much better performance than PG in all cases and this performance gap between the two methods had increased as the number of // increased. One of the major factors that caused the performance gap between these two methods is the size of the search space and this will be discussed in the next section. In conclusion, this result confirms that the PSim clustering method is a more flexible method which can process various types of queries efficiently than any other methods mentioned in this paper.

6.3. Comparison of the search space

Fig. 10 shows the size of the search space required for query processing when using two clustering methods, PG and PSim, respectively. We compared only the size of both methods since other methods provide no search space reduction method. In this figure, 'Total' represents the full size of the search space required when no search space reduction method applied to. In this experiment, we executed 1000 queries and recorded the accumulated number of the stored pages of data which needed to be accessed for processing those queries. As shown in the figure, both methods could reduce the size of the search space compared with 'Total'. In particular, PSim required much less search space than PG during query processing. Although PG could reduce the search space when a query has a long suffix pattern, it was still possible to search unnecessary data in many cases due to its limitation of the suffix based nature. Consequently, this result confirms that the proposed cluster selection method in Section 5.2 can reduce the search space required for query processing more efficiently than any other methods mentioned in this paper.

6.4. Scalability

To evaluate the scalability of our method, we scaled-up the document size from 10 M to 50 M and 100 M. Fig. 11 shows the performance comparison of various clustering methods for query samples in Fig. 7 on these three different sized documents. The graph for each query shows the result page I/Os on 10 M, 50 M and 100 M sized documents, respectively. In the figure, page I/Os of each method increased linearly in proportion to the size of the document on most queries. And PSim still showed a better performance than other methods for most queries in spite of the increase in the size of the document. In particular, PSim showed a lower increasing rate than other methods on queries Q4, Q5, Q8 and Q9. As we described in Section 6.2, these were queries that had a relatively long prefix and a short suffix pattern, and unlike other methods, PSim could efficiently cluster those elements required to these queries. Moreover, in this case, the result set sizes were small and PSim could get the result set by accessing fewer elements using only a small part of the total search space. The graph showed that this performance gain increased as the size of the document became larger.

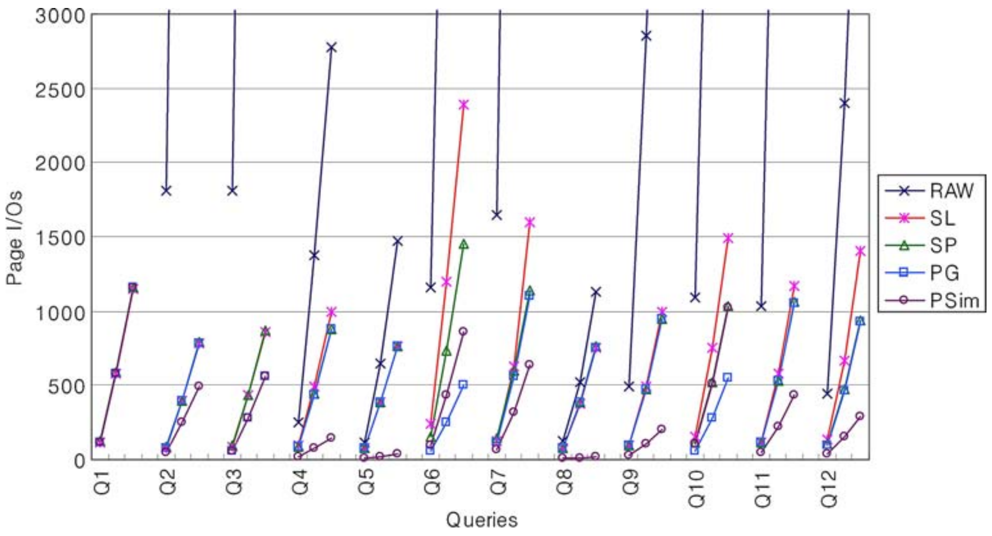


Fig. 11. Scale-up performance analysis for query samples.

7. Conclusion

We studied clustering issues for XML storage. In this paper, we proposed PSim clustering which is a clustering method that clusters the data nodes with similar paths and thus can reduce page I/Os required for query processing. Proposed method set the data nodes that have the same absolute paths as a base cluster unit, and then compares the path similarity between those units. Finally, clustering is done by applying the graph partitioning technique to the path similarity graphs. Through our experiments, we proved that the PSim clustering method is a flexible method which can process various types of queries efficiently.

In addition, we proposed a query processing method using signatures to benefit from the proposed clustering method. Cluster signatures facilitated the cluster-level access on the stored data, and thus, we could process a path query by accessing as small number of clusters as possible. In this method, each cluster has its own signature that stands for the path information of stored data nodes. We can select needed clusters to process a query by comparing their signatures with the query signature. As a result, the query processor need not search for clusters unrelated to the query, which in turn means that the search space of data can be reduced significantly.

Based on the foregoing, the PSim clustering method shows good performance in query processing. However, the proposed method assumes that the stored document is not updated frequently. So, further studies on the clustering method for frequently updated XML document will need to be conducted. And efficient query optimization techniques for the clustered document such as an indexing technique will need to be studied as well. When a query is processed by using one of the existing indexing techniques, the clustered document can still be more effective than the unclustered one because the result can be stored close to each other. However, most of the existing techniques were primarily studied on the base of the unclustered document. Thus more efficient indexing technique will need to be studied that takes advantage of partitioned data in the clustered document.

References

[1] J. Banerjee, W. Kim, S.-J. Kim, J.F. Garaza, Clustering a DAG for CAD databases, IEEE Trans. Software Eng. 14 (11) (1988) 1684–1699.

[2] D. Barbosa, A. Barta, A. Mendelzon, G. Mihaila, F. Rizzolo, P. Rodriquez-Gianolli, ToX – The Toronto XML Engine, in: Proc. WIIW'01, Brazil, 2001, pp. 66–73.
[3] A. Berglund, S. Baog, D. Chamberlin, et al., XML Path Language (XPath), ver. 2.0, W3C Working Draft, Tech. Report, 2001, Available from: <http://www.w3.org/TR/4>.
[4] E. Bertino, A.A. Saad, M.A. Ismail, Clustering techniques in object bases: a survey, Data Knowledge Eng. 12 (3) (1994) 255–275.
[5] J. Cheng, G. Yu, G. Wang, J.X. Yu, PathGuide: an efficient clustering based indexing method for XML path expressions, in: Proc. DASFAA'03, IEEE Computer Society, Kyoto, 2003, p. 257.
[6] A. Deutsch, M. Fernandez, D. Suciu, Storing semistructured data with STORED, in: Proc. SIGMOD 1999, ACM Press, Philadelphia, 1999, pp. 431–442.
[7] Excelon Corp., Excelon – the EBusiness Information Server, Available from: <http://www.exln.com/>.
[8] D. Florescu, D. Kossman, Storing and querying XML data using an RDBMS, IEEE Data Eng. Bull. 22 (3) (1999) 27–34.
[9] C. Gerlhof, A. Kemper, C. Kilger, G. Merkotte, Partition-based clustering in object bases, in: Proc. FODO'93, Lecture Notes in Computer Science, vol. 730, Springer, Chicago, 1993, pp. 301–316.
[10] R. Goldman, J. Widom, DataGuides: enabling query formulation and optimization in semistructured databases, in: Proc. VLDB'97, Morgan Kaufmann, Athens, 1997, pp. 436–445.
[11] H.V. Jagadish, S. Al-Khalifa, A. Chapman, Laks V.S. Lakshmanan, A. Nierman, S. Paparizos, J.M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, C. Yu, TIMBER: a native XML database, VLDB 11 (4) (2002) 274–291.
[12] C.C. Kanne, G. Moerkotte, Efficient storage of XML data, in: Proc. ICDE'00, IEEE Computer Society, San Diego, 2000, p. 198.
[13] V. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Sov. Phys. Dokl. 10 (8) (1966) 707–710.
[14] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom, Lore: a database management system for semistructured data, SIGMOD Record 26 (3) (1997) 54–66.
[15] X. Meng, D. Luo, M.L. Lee, J. An, OrientStore: a schema based native XML storage system, in: Proc. VLDB'03, Morgan Kaufmann, Berlin, 2003, pp. 1057–1060.
[16] A. Schmidt, F. Waas, M.L. Kersten, M.J. Carey, I. Manolescu, R. Busse, XMark: a benchmark for XML data management, in: Proc. VLDB'02, 2002, pp. 974–985.
[17] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D.J. DeWitt, J.F. Naughton, Relational databases for querying XML documents: limitations and opportunities, in: Proc. VLDB'99, Morgan Kaufmann, Edinburgh, 1999, pp. 302–314.
[18] Software AG. Tamino Information Server for Electronic Business, Technical Whitepaper, Available from: <http://www.tamiono.com/tamino/Download/tamino.pdf>.
[19] F. Tian, D.J. DeWitt, J. Chen, C. Zhang, The design and performance evaluation of alternative XML storage strategies, SIGMOD Record 31 (1) (2002) 5–10.
[20] M.M. Tsangaris, J.F. Naughton, A stochastic approach for clustering in object bases, in: Proc. SIGMOD, ACM Press, Colorado, 1991, pp. 12–21.
[21] J. Kim, I. Choi, H.-S. Lee, H.-J. Kim, XDBox: implementation of XML object repository, in: Proc. KISS Spring, Jeju, 2003.
[22] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, ACM 21 (1) (1974) 168–173.

Ilhwan Choi is a Ph.D. candidate at the School of Computer Science and Engineering, Seoul National University. His current research is focused on developing efficient clustering methods for XML database systems. He is also interested in XML indexing and database management systems. He received his M.S. degree in School of Computer Science and Engineering from Seoul National University in February 1998.

Bongki Moon is an Associate Professor of Computer Science at the University of Arizona. His current research is focused on developing high performance database systems and scalable web servers for large scale, data-intensive applications. He is also interested in XML indexing, data mining and warehousing, and parallel and distributed processing. He won the career award from the National Science Foundation in April 1999. He received his Ph.D. degree in Computer Science from University of Maryland, College Park, in December 1996.

Hyoung-Joo Kim is a Professor of the School of Computer Science and Engineering at Seoul National University. His current research is focused on the database system for Gene Ontology. He is also interested in XML, object-oriented systems and database management systems. He received the B.S. degree in Computer Engineering from Seoul National University in 1982, and the M.S. and Ph.D. degrees in Computer Science from The University of Texas at Austin in 1985 and 1988, respectively.