Hierarchical Indexing Approach to Support XPath Queries

Nan Tang †, Jeffrey Xu Yu †, M. Tamer Özsu ‡, Kam-Fai Wong †

†*The Chinese University of Hong Kong*
{ntang,yu,kfwong}@se.cuhk.edu.hk

‡*University of Waterloo*
tozsu@cs.uwaterloo.ca

***Abstract*— We study new hierarchical indexing approach to process XPATH queries. Here, a hierarchical index consists of index entries that are pairs of queries and their (full/partial) answers (called extents). With such an index, XPATH queries can be processed to extract the results if they match the queries maintained in those index entries. Existing XML path indexing approaches support either child-axis (/) only, or additional descendant-or-self-axis (//) but only in the query root. Different from them, we propose a novel indexing approach to process a large fragment of XPATH queries, which may use /, //, and wildcards ($*$). The key issues are how to reduce the number of index entries and how to maintain non-overlapping extents among index entries. We show how to compress such index and how to evaluate XPATH queries on it. Experiments show the efficiency of our approaches.**

I. Introduction

XPATH is an XML query language operating on tree structured XML documents. XPATH queries are typically expressed by the child-axis (/), the descendant-or-self-axis (//) and wildcards ($*$). An XPATH query can be processed using join-based algorithms (e.g., [1], [2]). In addition to evaluating an XPATH query as joins, many indexing approaches (e.g., [3], [4]) are proposed which build structural summaries over XML tree for maintaining the results of XPATH queries. With such indexes, an XPATH query, e.g., /book/section/title, can be processed by extracting the result maintained in the index entry for /book/section/title. Moro et al. [5] compare indexing approaches with join-based approaches on evaluating XPATH queries. Their result shows that an indexing approach is better if the query is supported by the index. However, most indexing approaches can support XPATH queries with /-axis only but cannot handle //-axis efficiently.

In this paper, we propose a novel hierarchical index, called PP-Index, to process a robust set of XPATH queries. PP-Index is a hierarchical index that supports XPATH queries with any child-axis (/), descendant-or-self-axis (//) and wildcards ($*$). A hierarchical index consists of index entries that are pairs of queries and their (full/partial) answers (called extents). To make such an index approach effective and efficient, the key issue is to reduce the index size (i.e., the number of index entries and the total extents maintained).

The main contributions of this paper are as follows:

- We propose PP-Index for supporting XPATH queries with the /-axis, the //-axis and wildcards $*$.
- We propose a compression technique to significantly reduce the number of entries in PP-Index while supporting //-axis.
- We conduct experiments to demonstrate the efficiency of our approaches.

The rest of the paper is organized as follows. Section II describes the problem studied. Section III proposes PP-Index and its compression technique. Section IV shows experimental results. Section V concludes this paper.

II. Problem Statement

The fragment of XPATH queries we attempt to support is:

$$q ::= q/q \ \mid \ q/\!/q \ \mid \ A \ \mid \ *$$

Here, A is a label in an XML document and $*$ is the wildcard that matches any label. Also, / and // are *child-axis* and *descendant-or-self-axis*, respectively.

An XML index has a set of entries of the form $(q, \mathtt{E}(q))$, where q is an XPATH query, and $\mathtt{E}(q)$ is the extent (i.e., result) of q.

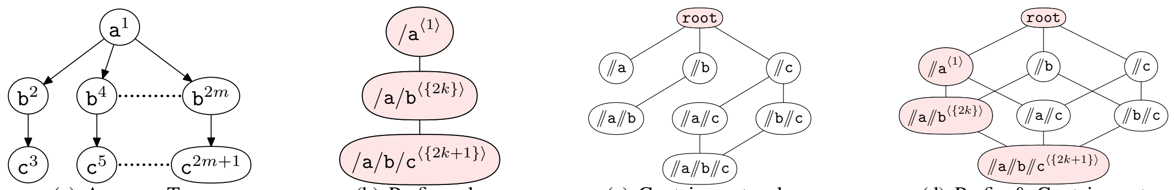
XML ***index problem***. The problem is to construct an effective index that is small in size. The requirement of small size demands the reduction of the number of entries, and the extents should be non-overlapping.

This XML index problem is challenging. With only the two most frequently used XPATH axes (/-axis and //-axis), the number of index entries is exponential (e.g., $O(2.62^n)$ [6] for a linear XML tree with n nodes). This indicates that it becomes infeasible to construct an index for efficiently processing XPATH queries with /-axis and //-axis, if an index is simply constructed as a set of index entries, $(q, \mathtt{E}(q))$. In this paper, we propose to construct such an index with $O(n^2)$ entries, and maintain non-overlapping extents.

III. PP-Index

Let $\mathtt{P}$ denote the set of all path queries (p) where $\mathtt{E}(p) \neq \emptyset$ over an XML tree. We divide $\mathtt{P}$ into three subsets: $\mathtt{P_c}$ is the subset of $\mathtt{P}$ with /-axis only; $\mathtt{P_d}$ is the subset of $\mathtt{P}$ with //-axis only, and $\mathtt{P_x}$ includes the remaining queries, i.e., $\mathtt{P_x} = \mathtt{P} - \mathtt{P_c} - \mathtt{P_d}$.

Consider a linear tree with n nodes. The size of $\mathtt{P_c}$ is $O(n)$. The size of $\mathtt{P_d}$ is $O(2^n)$, which is determined by the following deduction. There are $\binom{n}{i}$ combinations of i labels ($1 \leqslant i \leqslant n$).



(a) An XML Tree (b) Prefix only (c) Containment only (d) Prefix & Containment

Fig. 1. An XML Tree and Indexing Approaches

The size of $\mathtt{P_d}$ is $\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{n}=2^n-1$, as only $/\!/$-axis is allowed in $\mathtt{P_d}$. The size of $\mathtt{P}$ is $O(2.62^n)$ [6]. Due to the exponential sizes for $\mathtt{P_d}$ and $\mathtt{P}$, existing index approaches have primarily focused on $\mathtt{P_c}$.

We consider two interrelated but different relationships among P-queries to construct a hierarchical index for P-queries. They are prefix and path-containment. The former provides a mechanism to find the requested entry for a P-query. The latter provides a way to identify the requested extents to answer such a query.

Let $p = \alpha_1 l_1 \alpha_2 l_2 \ldots \alpha_m l_m$ and $p' = \alpha'_1 l'_1 \alpha'_2 l'_2 \ldots \alpha'_n l'_n$ be two P-queries where $m < n$, α_i is an axis and l_i is a label. We explain *prefix* and *path-containment* below:

Prefix relationship: p is a prefix of p' iff $\alpha_i = \alpha'_i$ and $l_i = l'_i$ $(1 \leqslant i \leqslant m)$. p is the *maximal prefix* of p' iff p is a prefix of p' and $n = m + 1$.

Path-containment relationship: p' is contained in p, denoted as $p' \sqsubseteq p$, if the labels in p match the labels in p' in order, and the last labels match (i.e., $l_m = l'_n$). Furthermore, for two matched labels (e.g., l_i and l'_j), the corresponding axes have that: $/$-axis maps to $/$-axis (i.e., $\alpha'_j = /$ if $\alpha_i = /$) and $/\!/$-axis maps to rightward path (i.e., $\alpha_i = /\!/$).

A. A Prefix Only Approach

A well studied XML path index, 1-Index [3], is a hierarchical index for $\mathtt{P_c}$-queries using prefix relationship. In 1-Index, there is an edge from an entry $(p_1, \mathtt{E}(p_1))$ to another entry $(p_2, \mathtt{E}(p_2))$ if p_1 is the maximal prefix of p_2. Consider an XML tree in Figure 1 (a). Its 1-Index is shown in Figure 1 (b) with three entries: $/a$, $/a/b$ and $/a/b/c$. Each entry p maintains an extent ($\mathtt{E}(p)$), indicated by "$\langle\rangle$". For example, the extent of $/a/b$ for the XML tree (Figure 1 (a)) has m tree nodes $2, 4, \cdots, 2\mathtt{m}$, indicated by $\langle\{2\mathtt{k}\}\rangle$ $(1 \leqslant \mathtt{k} \leqslant \mathtt{m})$ in Figure 1 (b). 1-Index is defined on $\mathtt{P_c}$-queries using prefix relationship. $\mathtt{P_c}$-queries are evaluated by finding a corresponding entry and extracting its extent. Note that for two index entries, $(p_1, \mathtt{E}(p_1))$ and $(p_2, \mathtt{E}(p_2))$, in 1-Index, we have $\mathtt{E}(p_1) \cap \mathtt{E}(p_2) = \emptyset$ if $p_1 \neq p_2$.

However, 1-Index cannot handle all P-queries. The size of a similar approach for P-queries may be sizable. Assume that we build an index for P-queries with prefix relationship, as 1-Index does for $\mathtt{P_c}$-queries. This approach suffers two problems: (i) the number of entries is exponential (in $O(2.62^n)$ [6]) because of the combination of /- and $/\!/$-axis, and (ii) the overlapping between two extents is high. With such redundancy, a large storage is required and the duplications require to be removed during query processing.

B. A Path-Containment Only Approach

A hierarchical index for $\mathtt{P_d}$-queries can be constructed using path-containment relationship. In such an index, there is an edge from an entry $(p_1, \mathtt{E}(p_1))$ to another entry $(p_2, \mathtt{E}(p_2))$ if $p_2 \sqsubseteq p_1$ and there does not exist an entry $(p_3, \mathtt{E}(p_3))$ where $p_2 \sqsubseteq p_3 \sqsubseteq p_1$. Such an index for the XML tree in Figure 1 (a) is shown in Figure 1 (c). This hierarchical index, based on path-containment, suggests that an entry $(p, \mathtt{E}(p))$ does not need to maintain its extent $\mathtt{E}(p)$ if $\mathtt{E}(p)$ can be identified by searching its descendants in the index. Consider four $\mathtt{P_d}$-queries: $/\!/c$, $/\!/a/\!/c$, $/\!/b/\!/c$, and $/\!/a/\!/b/\!/c$, in Figure 1 (c). The results for them are the same. There is no need to maintain the same extent four times. The same extent for the four queries can be maintained at $\mathtt{E}(/\!/a/\!/b/\!/c)$ only. Processing one of the four $\mathtt{P_d}$-queries requires searching through the hierarchical index. The path-containment based index can efficiently support path-containment relationship, but it cannot support prefix relationship as 1-Index does, e.g., it cannot easily identify $/\!/a/\!/b$ from $/\!/a$ in Figure 1 (c).

C. A New Prefix/Containment Approach

We propose a new hierarchical index which supports both prefix and path-containment and can answer any P-query. We explain the main idea using an example, and omit detailed discussion due to space constraints.

In order to reduce the number of index entries and maintain non-overlapping extents, we introduce *weak-extent*, denoted as $\overline{\mathtt{E}}(p)$, such that $\overline{\mathtt{E}}(p) \subseteq \mathtt{E}(p)$. With weak-extents, a query (q) may need to be answered by either a single entry or several entries.

The main idea behind the hierarchical index is as followings. Index entries are $\mathtt{P_d}$-queries, since any P-query (p) has a unique corresponding $\mathtt{P_d}$-query, $p_d = \textsc{double}(p)$, where $\textsc{double}(p)$ replaces all $/$-axes in p by $/\!/$-axes. Such index entries are used to find the requested entries for a given P-query in a top-down search. On the other hand, any P-query (p) has a unique corresponding $\mathtt{P_c}$-query, $p_c = \textsc{single}(p)$, where $\textsc{single}(p)$ replaces all $/\!/$-axes in p with $/$-axes. The weak-extent maintained for $(p_d, \overline{\mathtt{E}}(p_d))$ is $\overline{\mathtt{E}}(p_d) = \mathtt{E}(p_c)$ where $p_c = \textsc{single}(p_d)$. There is no overlapping between weak-extents, since an XML tree node x with label-path p_x is uniquely maintained at entry q where $q = \textsc{double}(p_x)$.

For the XML tree in Figure 1 (a), our proposed hierarchical index is shown in Figure 1 (d). The nodes (entries) and edges are explained as follows:

1) A query (p_d) in an entry $(p_d, \overline{\mathrm{E}}(p_d))$ is a $\mathtt{P_d}$-query. Comparing Figure 1 (d) with Figure 1 (c), the number of entries in both figures are the same.
2) The edges that represent path-containment/prefix relationships in Figure 1 (b) and (c) are all maintained in Figure 1 (d).
3) The weak-extents maintained for $\mathtt{P_d}$-queries (p_d) are $\overline{\mathrm{E}}(p_d) = \mathrm{E}(p_c)$ where $p_c = \mathrm{SINGLE}(p_d)$. Consider Figure 1 (b) and Figure 1 (d). The weak-extents maintained in $\overline{\mathrm{E}}(/\!/a)$, $\overline{\mathrm{E}}(/\!/a/\!/b)$, and $\overline{\mathrm{E}}(/\!/a/\!/b/\!/c)$ are in fact $\mathrm{E}(/a)$, $\mathrm{E}(/a/b)$ and $\mathrm{E}(/a/b/c)$, respectively.

With such a hierarchical index (Figure 1 (d)), any $\mathtt{P}$-query can be efficiently processed. First, a $\mathtt{P_c}$-query (p_c) can be processed to find its corresponding entry $(p_d, \overline{\mathrm{E}}(p_d))$ where $p_d = \mathrm{DOUBLE}(p_c)$, and return $\overline{\mathrm{E}}(p_d)$ since $\overline{\mathrm{E}}(p_d) = \mathrm{E}(p_c)$. Consider $p_1 = /a/b$, $p_2 = \mathrm{DOUBLE}(p_1) = /\!/a/\!/b$, and the result is $\mathrm{E}(p_1) = \overline{\mathrm{E}}(p_2) = \{2, 4, \cdots\}$. Second, a $\mathtt{P_d}$-query (p_d) can be processed to find the entry $(p_d, \overline{\mathrm{E}}(p_d))$, and combine $\overline{\mathrm{E}}(p_d)$ and all the weak-extents $\overline{\mathrm{E}}(p'_d)$ if $(p'_d, \overline{\mathrm{E}}(p'_d))$ is a descendant of $(p_d, \overline{\mathrm{E}}(p_d))$ in the index and both p_d and p'_d have the same last label (path-containment). Consider $/\!/c$ with the result $\mathrm{E}(/\!/c) = \overline{\mathrm{E}}(/\!/c) \cup \overline{\mathrm{E}}(/\!/a/\!/c) \cup \overline{\mathrm{E}}(/\!/b/\!/c) \cup \overline{\mathrm{E}}(/\!/a/\!/b/\!/c)$. The weak-extents for $\overline{\mathrm{E}}(/\!/c)$, $\overline{\mathrm{E}}(/\!/a/\!/c)$, and $\overline{\mathrm{E}}(/\!/b/\!/c)$ are empty. Therefore, we have $\mathrm{E}(/\!/c) = \overline{\mathrm{E}}(/\!/a/\!/b/\!/c) = \{3, 5, \cdots\}$. Finally, any other $\mathtt{P}$-queries (i.e., $\mathtt{P_x}$-queries) can be processed by combining the techniques mentioned above.

D. PP-Index *Compression*

A node may have an empty weak-extent. We call a node *real-node* if its weak-extent is non-empty; otherwise we call it *virtual-node*. A virtual-node v in a PP-Index $(\mathcal{G})$ is *removable* if all nodes (u), which have p-edges from u to v, are virtual-nodes. All removable nodes (v) and the incoming/outgoing edges around them can be removed, which results in a compressed graph $\mathcal{G}'$. The number of virtual-nodes is large. The compression greatly reduces the entry size from $O(2^n)$ to $O(n^2)$, where n is the height of an XML tree.

IV. Performance Study

We report the performance of the proposed index in terms of efficiency. We compared our algorithms with join-based approaches TSGeneric [7] and Twig^{2}Stack [8], and disk-based F&B-Index [9]. TSGeneric optimizes TwigStack by leveraging XR-Tree for skipping irrelevant data items. Twig^{2}Stack outperforms TwigStack in that it can avoid useless path matches, especially for the /-axis. The test document is 226M XMark XML document. The queries are selected from randomly generated queries, listed in Table I.

From Figure 2 we can see that PP-Index (PPE) outperforms the other approaches in one to two order of magnitude, since the size of corresponding compressed PP-Index is small. We only need to scan a small number of index nodes to identify the results.

$\mathtt{PQ_1}$: //people/person//homepage
$\mathtt{PQ_2}$: //site//people//person
$\mathtt{PQ_3}$: //site//regions//item/location
$\mathtt{PQ_4}$: /site/open_auctions//annotation/description//listitem

TABLE I
XMark Queries

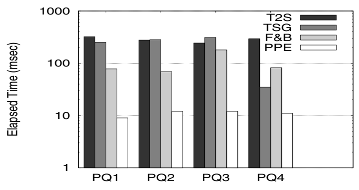


Fig. 2. Path queries on XMark

F&B outperforms Twig^{2}Stack (T2S) and TSGeneric (TSG) in most cases, since it optimizes the index lookup with join-based method, which can prune many nodes that do not contribute to the result. For $\mathtt{PQ_4}$, TSGeneric outperforms F&B, since XR-Tree can improve performance for queries having low selectivity as $\mathtt{PQ_4}$. For the other queries, T2S and TSG have similar performance since the node selectivity on XMark is high and XR-Tree cannot accelerate processing much.

V. Conclusion

In this paper, we propose PP-Index for path queries. We then show how to compress the index entries, and maintain non-overlapping weak-extents. Experiments show the efficiency of our proposed approaches. For future work, we plan to compare our approaches with the "join-in-the-middle" type index (e.g., FIX [10]). We also plan to further compress the number of index entries by mining frequently used XPATH queries.

Acknowledgment

This work was partially supported by the CUHK strategic grant (No. 4410001) and by a grant of RGC, Hong Kong SAR, China (No. 418205).

References

[1] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava, "Structural joins: A primitive for efficient XML query pattern matching." in *ICDE*, 2002, pp. 141–.
[2] N. Bruno, N. Koudas, and D. Srivastava, "Holistic twig joins: optimal XML pattern matching." in *SIGMOD*, 2002, pp. 310–321.
[3] T. Milo and D. Suciu, "Index structures for path expressions." in *ICDT*, 1999, pp. 277–295.
[4] R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth, "Covering indexes for branching path queries." in *SIGMOD*, 2002, pp. 133–144.
[5] M. M. Moro, Z. Vagena, and V. J. Tsotras, "Tree-pattern queries on a lightweight XML processor." in *VLDB*, 2005, pp. 205–216.
[6] B. Mandhani and D. Suciu, "Query caching and view selection for XML databases." in *VLDB*, 2005, pp. 469–480.
[7] H. Jiang, W. Wang, H. Lu, and J. X. Yu, "Holistic twig joins on indexed XML documents." in *VLDB*, 2003, pp. 273–284.
[8] S. Chen, H.-G. Li, J. Tatemura, W.-P. Hsiung, D. Agrawal, and K. S. Candan, "Twig^{2}stack: Bottom-up processing of generalized-tree-pattern queries over XML documents." in *VLDB*, 2006, pp. 283–294.
[9] W. Wang, H. Wang, H. Lu, H. Jiang, X. Lin, and J. Li, "Efficient processing of XML path queries using the disk-based f&b index." in *VLDB*, 2005.
[10] N. Zhang, M. T. Özsu, I. F. Ilyas, and A. Aboulnaga, "FIX: Feature-based indexing technique for XML documents." in *VLDB*, 2006, pp. 259–270.