Security-Conscious XML Indexing

Yan Xiao, Bo Luo, and Dongwon Lee

The Pennsylvania State University, University Park, USA
xiaoyan515@gmail.com, {bluo,dongwon}@psu.edu

Abstract. To support *secure* exchanging and sharing of XML data over the Internet, a myriad of XML access control mechanisms have been proposed. In the setting of node-level fine-grained access control, query evaluation is a process of locating XML nodes that (1) satisfy query constraints, and (2) do not violate security policies. In this regard, we propose and empirically validate a suite of XML indices for multi-level XML security model.

1 Introduction

Recently, many proposals focusing on XML security models or enforcement mechanisms have appeared (e.g., XACML, [3], [1]). However, XML query processing issues using indices in dealing with secure XML data has gotten little attention. In this paper, we are interested in devising XML indexing methods to efficiently support Multi-level security model (e.g., [7]) for XML data.

Motivation: Imagine a company that has three-level security policy: {`Top Secret`, `Secret`, `Public`}, denoted as {3, 2, 1}, respectively. In the following XML data, each node has an associated security level in the "`s_nodename`" attribute:

```
<Dept s_Dept='1'>
 <Manager s_Manager='1'><Name s_Name='1'>Tom</></><Staff s_Staff='1'><Name s_Name='1'>Jane</></>
 <Proj s_Proj='2' pname='Security' s_pname='2'>
 <Year s_Year='2'>2004</> </Year> <Budget s_Budget='3'>300K</> </Proj>
</Dept>
```

When a user "Tom" with security level '2' issues Q:`//Proj/Budget`, he would not receive budget information for the `pname='Security'` project due to insufficient security level. In this case, enforcing the right access controls of the query Q by "Tom" is amount to evaluating Q':`//Proj[@s_Proj<=2]/Budget[@s_Budget<=2]`. When there are hundreds of such `Proj` and `Budget` elements in documents, therefore, quickly locating those elements with security level ≤ 2 plays a critical role in improving the "secure" query processing. The goal of this paper is, therefore, to devise efficient indexing schemes for such a scenario. We use the notations: $SL(n)$, $SL(q)$ (or $\{L\} : q$) for the security level of node n and query q, and $MinSec(n, D)$ for minimum $SL(n)$ of nodes in document D.

2 Background and Related Work

Multi-Level Access Control model (e.g. [7]) assigns each object (e.g., node) and each subject (e.g., user) a *security level*, and enforces a rule: "*a level L_i subject*

R. Kotagiri et al. (Eds.): DASFAA 2007, LNCS 4443, pp. 949–954, 2007.

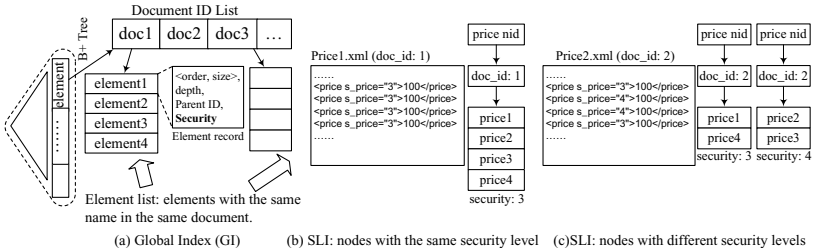


Fig. 1. Global Index and Single-Level Index

cannot access a level L_j object unless $L_i \geq L_j$". Its simplicity enables the wide acceptance in military or organizational applications, but since it requires "total order" among levels, it is less flexible. More flexible models (DAC or RBAC), allow lattice-based levels and are supported by commercial DBMS. In [4], a specific authorization sheet is associated with each XML document/DTD. [3] extends [4] by enriching the supported authorization types, providing a complete description of specification and enforcement mechanism. In [1], an access control environment for XML documents and techniques to deal with authorization priorities and conflict resolution issues are proposed.

We base our approach on XISS [6]. It uses a unique numbering scheme to quickly join ancestor-descendent nodes, and at the bottom layer of the index, information per node is sorted by document order to support fast sort-merge. [2] studies query evaluation methods by exploiting the properties of security model. Our indexing method is complementing [2]. The concept of "two-tier" introduced in [5] (for RDBMS) is the basis of our work. We first adopt two-tier index to XML context, and improve it further.

3 Security-Conscious XML Index

Global Index. As the baseline approach, *Global Index* constructs a regular XML index. Here we choose *element index* of XISS. As shown in Figure 1 (a), all element tags are assigned a unique ID, then indexed via B^+ tree at top. The bottom of the tree points to a *document ID list* that contains a list of document IDs where the element appears. Further, each item of the document ID list points to an *element list*, which stores element-related information. To support XML access control, we add additional "security level" of each element in the element list. In this scheme, security check is done at individual element level. Therefore, it could be acceptable for users with high security levels, since they are likely to access most data anyway, but could be inefficient for most users.

Single-Level Index (SLI). This approach is to have separate index for each security level. Thus, when a query `{2}://Proj` is issued, one can simply look for `Proj` elements from two *single-level* indices: security level 1 and 2. Since all elements in that index are guaranteed to satisfy the specified security constraint,

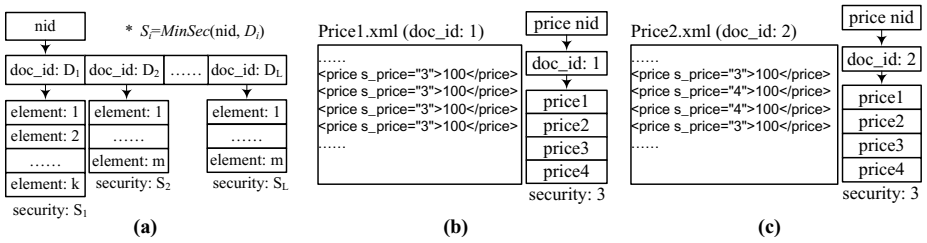


Fig. 2. (a) Modified SLI; (b) and (c) Examples of the modified SLI

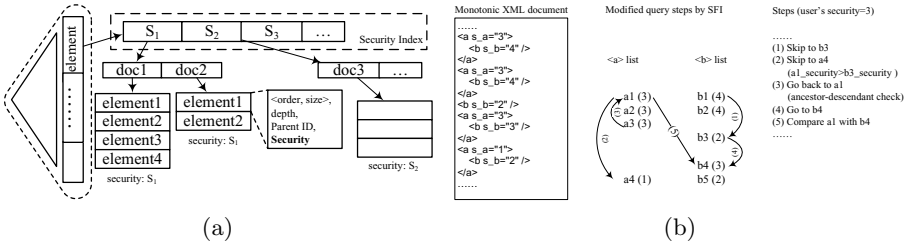


Fig. 3. (a) Minimum-Security Index (MSI); (b) Special case of SFI

there is no need for additional check. This approach is efficient when elements with the same name have the same security level within an document because the document ID and elements can be stored in one security class. However, if elements have different security levels within a document, all information has to be stored in different security levels. This will cause large storage and query overhead. Figure 1 (b) and (c) show examples of both scenarios.

Minimum-Security Index (MSI). Both the GI and SLI have pros and cons. To retain only pros of each, we adopt *Two-Tier Coarse Index* [5] (from relational model) and fit it into XML model. Further, we improve it with the $MinSec$ concept, making it *Minimum-Security Index (MSI)*.

A problems of GI is that the document IDs we get from B^+ tree may not contain satisfactory IDs at all. Suppose an element `Budget` appears in document D_7 with security level 4. For a query q_1, `{3}://Budget`, we do not have to retrieve `<Budget>` from D_7 to check their security levels since $SL(q_1) = 3 < MinSec(\texttt{Budget}, D_7) = 4$. If we maintain a reverse link atop single-level index that, for each $MinSec$, points to a document ID list (e.g., $MinSec$ 4 points to D_7), then significant saving can be made by not visiting unnecessary documents. To use $MinSec$, SLI is modified as shown in Figure 2 (a). Here, we keep element list the same as GI, and store document ID into separate security classes based on $MinSec$ values. The modified SLI improves SLI by reducing multiple storage and avoiding querying the same document multiple times. The example XML documents in modified SLI are shown in Figure 2 (b) and (c).

The MSI, illustrated in Figure 3 (a), combines pros of GI and SLI, but exploits the document-level security check by avoiding retrieving unnecessary documents. Therefore, when elements in a document have unique security level (e.g., all `Proj` element in D_1 have security level 4), the MSI is the most advantageous. However, when elements in a document have various security levels, it becomes less efficient. For instance, a document D_1 has 1,000 `Budget` elements, where all have security level 5, except one with 2. Then, for a query {3}`://Budget`, even if there is only one element satisfying the security constraint, the MSI still retrieves all 1,000 `Budget` elements since $MinSec(\texttt{Budget}, D_1)$ is 2.

Skip-Record Index (SRI). One of the most time-consuming steps in query processing is to retrieve elements from element list and perform sort-merge using ancestor-descendent relationship. To speed up this step, we can avoid sort-merge for those element pairs which cannot satisfy security constraint. One can sort elements by security levels to quickly determine whether or not to continue checking security, but this is not possible since elements in the element list are already sorted by their order (i.e., pre-ordering in the XISS). To solve this, we maintain another number per each element e_1, called *Skip-Record* that either (1) quickly tells how many element records to skip to get to the next element e_2 that satisfies $SL(e_2) < SL(e_1)$, or (2) is "-1" if there is no more such element left in the element list. Consider the following situation:

```
1:  <Proj s_Proj='3'/>  # Skip-Record=2      4:  <Proj s_Proj='2'/>  # Skip-Record=-1
2:  <Proj s_Proj='4'/>  # Skip-Record=1      5:  <Proj s_Proj='6'/>  # Skip-Record=0
3:  <Proj s_Proj='5'/>  # Skip-Record=0      6:  <Proj s_Proj='3'/>  # Skip-Record=-1
```

The Skip-Record value "2" for node 1 implies that one needs to skip "two" elements to get to lower security level. The Skip-Record value "-1" of node 4 suggests that there is no elements with lower security level. For query {1}`://Proj`, since the security level of the first `Proj` is 3, it is not satisfactory for the given query. Instead of checking element 2 and 3, we can use the Skip-Record to quickly "skip" two records and go to element 4 directly. When this forth element is not again satisfying the security constraint, one can quit searching in this element list since the Skip-Record of the forth `Proj` is "-1".

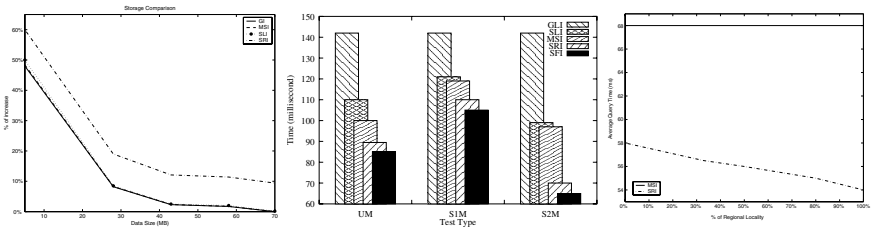
Skip-Forward Index (SFI). Consider the following document:

```
<root>
  <a s_a='3'> <b s_b='4'/> </>  # a1, b1      <a s_a='3'> <b s_b='4'/> </>  # a2, b2
  <a s_a='3'> <b s_b='4'/> </>  # a3, b3      <a s_a='3'> <b s_b='4'/> </>  # a4, b4
  <a s_a='1'> <b s_b='2'/> </>  # a5, b5
</root>
```

In processing query "`a/b`", sort-merge between two lists are needed. Furthermore, the following depicts two such lists with (SL, Skip-Record) for each element.

```
a-list -- a1(3,3) a2(3,2) a3(3,1) a4(3,0) a5(1,-1) ...
b-list -- b1(4,3) b2(4,2) b3(4,1) b4(4,0) b5(2,-1) ...
```

Consider query {3}`://a//b`. `a1-b1` pair is first compared, since it satisfies ancestor-descendent relationship, its security is checked; `a1`'s security is satisfied, but `b1`'s is not, thus `b1` is not returned. At this point, according to `b1`'s Skip-Record, we can skip the next "3" `<b>`, and examine `b5` immediately. However, for the `a`-list side,



(a) (b) (c)

Fig. 4. (a) Index size comparison; (b)-(c) Query evaluation for monotonic model

next record to be evaluated is `a2` since `a1` was satisfied. `a2` again satisfies security constraint, but `a2-b5` fails ancestor-descendent relationship check. Then we have to continue checking all remaining `<a>` until it reaches `a5-b5`, which satisfies both ancestor-descendent relationship and security constraint (i.e., ≤ 3).

However, in the "monotonic" security model where ancestor's security levels are guaranteed to be not higher than descendants', we can avoid security check of ancestors. In the above example, after `a1-b1` pair is examined and the next element to examine is determined to be `b5`, according to the Skip-Record number of `b1`, we can immediately rule out `a2-a4` since their security levels are higher than `b5`'s and thus none of them can be ancestor of `b5` in the monotonic model. In general, in the monotonic security model, if a's SL is greater than b's SL, then using the Skip-Record numbers, we can skip to the next a whose SL is no greater than b's SL. If current a's Skip-Record number is "-1", then we move to the next a and b, and continue the above steps. If a's SL is not greater than b's, we can then check ancestor-descendant relationship. There is a special case as shown in Figure 3(b): `b3` has no `a` ancestor. If we use above algorithm, we jump to `a4` after find `a1`'s security value is greater than `b3`'s. So, we miss `a3-b4` because we skipped `a3`. Therefore, after we find that `a4` is after `b3` and is not its ancestor, we need to go back and check with `b4`.

4 Experimental Validation

We implemented the five variations in the XISS system for evaluation. We have generated three variations of security distribution for monotonic data: uniform distribution for (UM), skewed security distribution with more low security level data (S1M), skewed security distribution with more high security level data (S2M); and the same for non-monotonic data: UNM, S1NM and S2NM.

Index Size: The index sizes are compared in Figure 4(a), where % of the increase of index size was measured for different XML sizes. SRI incurs the most index space increase since it maintains Skip-Record for each item in element list, but the additional storage overhead is not substantial. When data size is small, the huge increase is due to the default size allocation by XISS. As data size increases, the overhead is within 10% range and almost 1% for other methods.

Query Evaluation Time: The impact of security model selection (monotonic or non-monotonic) towards system performance is minor. Figure 4(b) shows the result of five variations over three data sets under monotonic model, and the case for non-monotonic is similar. Regardless of the skewness of the security information, SRI and SFI outperforms conventional ones significantly.

SRI is more efficient when elements in the list are sorted by security values in ascending order. We test query performance on different regional locality in a document. First, we define a *block* for element E in an XML document as the region where all security levels of E elements are the same or sorted by ascending order. Then, suppose in document D, E appears n times in b blocks. The regional locality of E in D is calculated as: $R(E, D) = 100\%$, when $b = 1$; $R(E, D) = (n-b)/n$, otherwise. As shown in is shown in Figure 4(c), SRI is more efficient when $R(E, D)$ increases. This is because the whole block of records is skipped if the first element can not satisfy security check.

5 Conclusion and Future Work

In this paper, we consider five index schemes that support multi-level XML access control – Global Index, Single-level Index, Minimum Security Index, Skip-Record Index, and Skip-forward Index. By utilizing the characteristics of XML model and monotonic/non-monotonic security models, SRI or SFI improves other variations up to 130% at best. In general, all the proposed indices can effectively take advantage of pre-security checks, while not intruding the original XML database like XISS and their path join algorithms. Thus, our extension is quite practical.

References

1. E. Bertino and E. Ferrari. "Secure and Selective Dissemination of XML Documents". *IEEE Trans. on Information and System Security*, 5(3):290–331, Aug. 2002.
2. S. Cho, S. Amer-Yahia, L. V.S. Lakshmanan, and D. Srivastava. "Optimizing the Secure Evaluation of Twig Queries". In *VLDB*, Hong Kong, China, Aug. 2002.
3. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. "A Fine-Grained Access Control System for XML Documents". *IEEE Trans. on Information and System Security*, 5(2):169–202, May 2002.
4. E. Damiani, S. De Capitani Di Vimercati, S. Paraboschi, and P. Samarati. "Design and Implementation of an Access Control Processor for XML Documents". *Computer Networks*, 33(6):59–75, 2000.
5. S. Jajodia, R. Mukkamala, and I. Ray. "A Two-tier Coarse Indexing Scheme for MLS Database Systems". In *IFIP WG 11.3 Working Conf. on Data and Applications Security (DBSec)*, Lake Tahoe, CA, Aug. 1998.
6. Q. Li and B. Moon. "Indexing and Querying XML Data for Regular Path Expressions". In *VLDB*, Roma, Italy, Sep. 2001.
7. S. Osborn. "Mandatory Access Control and Role-Based Access Control Revisited". In *ACM Workshop on Role Based Access Control*, pages 31–40, Fairfax, VA, 1997.