Concurrency Control Access of Dynamic XML Document with the Locking Method

Jeang-Kuo Chen*, Kuan-Chang Lu
Department of Information Management, Chaoyang University of Technology
jkchen@cyut.edu.tw, s9514637@cyut.edu.tw

ABSTRACT-*XML is a popular media in many commercial transactions. It is necessary to support XML document access by multiple users concurrently. The access method of single-user for XML document is unsuitable in a multi-user environment. In this paper, we propose three concurrency control algorithms for searching, inserting, and modifying relative element data in a dynamic XML document. With the locking techniques, the algorithms allow multiple users to concurrently access the same data without any error occurrence.*

Keywords : Concurrency Control, XML, Locking, Access Method.

1. Introduction

The XML (eXtensible Markup Language) [12] is popular in many commercial applications such as electronic commerce, data exchange, data warehouse etc [1, 5, 6, 8, 9, 10, 11]. With a DTD or an XML Schema, an XML document can be verified to be a valid document or not. When the quantity of XML documents increases quickly in a company, it is necessary to manage XML documents with a database management system in order to facilitate management and access of XML documents [4, 7, 9, 14, 15]. A concurrency control mechanism is required to maintain multi-user accessing the same data at the same time. If XML database is accessed by many transactions without any concurrency control, some unpredictable problems such as lost update, dirty read, or incorrect summary [3] may occur. An example is illustrated as follows. An XML document is used to record the data of a bookstore. When two transactions access the same magazine data at the same time without any concurrency control, some unexpected mistakes may happen. Suppose the stock of a magazine in the bookstore is 10. Transaction A (T_A) will subtract 3 form the stock because 3 magazines are sold. Transaction B (T_B) will add 5 to the stock because 5 magazines are stocked. T_A includes three steps (1) reading the stock of the magazine, (2) subtracting 3 form the stock, and (3) updating the new stock into database. T_B also includes three steps (1) reading the stock of the magazine, (2) adding 5 to the stock, and (3) updating the new stock into database. If T_A and T_B execute sequentially (T_A then T_B, or T_B then T_A), the result is correct as shown in Table 1. However, if T_A and T_B execute alternately, as shown in Table 2, the lost update problem occurs because T_A loses the new stock value at the order 6 of Table 2. Form the above example, we can understand the importance of concurrency control in a database with multi-user access.

Table 1. Correct result.

order	transaction	operation	stock
1	T_A	Read x	10
2	T_A	x=x-3	10
3	T_A	Write x	7
4	T_A	Commit	7
5	T_B	Read x	7
6	T_B	x=x+5	7
7	T_B	Write x	12
8	T_B	Commit	12

Table 2. Incorrect result.

order	transaction	operation	stock
1	T_A	Read x	10
2	T_B	Read x	10
3	T_A	x=x-3	10
4	T_B	x=x+5	10
5	T_A	Write x	7
6	T_B	Write x	15
7	T_A	Commit	15
8	T_B	Commit	15

This paper proposes three concurrency control algorithms for searching, inserting, and modifying elements in an XML document. With the locking technique, the share lock (s-lock) [2] is used to lock an element by one transaction before reading the element. The s-lock is sharable with other s-locks which means an element can be s-locked by more then one transaction that only read the locked element concurrently. The exclusive lock (x-lock) [2] is used to lock an element by one transaction before changing the contents of this element. The x-lock is exclusive with other s-locks or x-locks that means an element can be x-locked only by one transaction at one time. The s-locks are compatible but x-locks. An x-lock is incompatible with any other s-lock and x-lock. A transaction must lock an element before accessing the element and release (unlock) the lock as soon as possible after handling the element. The techniques of breadth-first search and lock-coupling [2] protocol are used when traversing an index tree associated with an XML document to find one or more target elements.

2. Relative Techniques

Derived form SGML [13], an XML document has two types [12]. The first type is called well-formed while the second type is called valid. A correct XML document must be well-formed if it is verified by a DTD (Document TypeDefinition) or an XML schema.

Concurrency control offers a reliable mechanism for database concurrent access under a multi-user environment. There are three main methods, locking, time stamp, and optimistic [3] for concurrency control. We use the first one because it is the most popular and easy to implement. An XML document is associated with an index tree for speed up in database. As shown in Figure 2. each node in the index tree includes four pointer fields named *Tag_name*, *Content*, *Attribute*, and *Child*[1.. n]. The *Tag_name* points to the tag name of an XML element. The *Content* points to the position of an element content. *Attribute* points to the position of the string for attribute names and their values. The *child*[1.. n] points to each child node of a parent node. When a transaction traverses the index tree, the breadth-first search and lock-coupling [2] techniques can be used to correctly find the target node for searching, or updating element data. The lock types used in this paper are share-lock (s-lock) and exclusive-lock (x-lock). Only s-locks are compatible. An x-lock is incompatible with a s-lock or an x-lock. The compatible condition between the two lock types is shown in Figure 3.

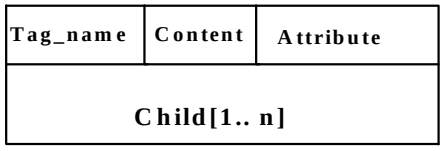

Figure 2. Node structure.

	s-lock	x-lock
s-lock	O	X
x-lock	X	X

Figure 3. Compatible condition of two lock types.

3. Concurrency Control Algorithms

3.1 Search Algorithm

With the breadth-first search, the search transaction descends the index tree to find and return the elements that contain the specific keywords. An element is s-locked before it is read and is unlocked immediately after it would not be used. The simple flow chart of the search algorithm is shown in Figure 4. Some parameters and variables used in this flow chart and algorithm are described as follows. *Tx* is the root of the index tree. *KEY* is a set of search keywords. N_{temp} is a node to be visited currently. *Queue* is a queue used to save nodes for the breadth-first search while *Result* is a set for saving returned elements. The detailed search algorithm is described below.

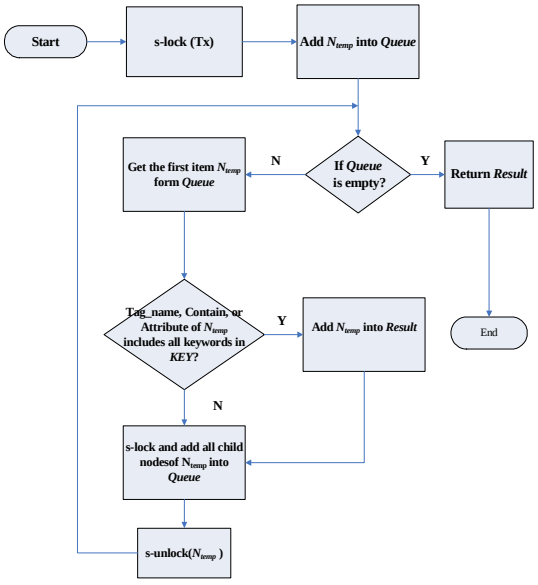

Figure 4. The flow chart of search algorithm.

Algorithm Search(*Tx*, *KEY*）
Input : NODE POINTER *Tx* ;
STRING SET *KEY*;
Output：a set of nodes;
Begin
NODE POINTER N_{temp};
QUEUE *Queue*;
NODE POINTER SET *Result*;
01. s-lock(*Tx*);
02. Add *Tx* to *Queue*;
03. while *Queue* is not empty, do
04. N_{temp} ← get_node(*Queue*);
05. if N_{temp}'s Tag_name, Content, or Attribute includes all keywords in *KEY*, then
06. add N_{temp} to *Result*;
07. end if;
08. for each child[i] of N_{temp}, do
09. s-lock(child[i]);
10. add child[i] to *Queue*;
11. end for;
12. s-unlock(N_{temp});
13. end-while;
14. return(*Result*);
End Search.

3.2 Insertion Algorithm

Given the index tree, an element, and a path, the insertion transaction descends the tree along the path to a parent node and inserts the given element as a child node into its parent node. The nodes on the path must be s-locked sequentially and released immediately after they are processed. The target node must be converted form s-locked into x-locked before the insertion of the given element. The simple flow chart of the insertion algorithm is shown in Figure 5. Some parameters and variables used in this flow chart and algorithm are described as follows. *Tx* is the root of the index tree. *Ins_Element* is an element to be inserted into an element. *Path* is a string composed of tag names and/or attributes. *Queue* is a queue used to save nodes for breadth-first search. N_{temp} is a node to be visited currently. *Aes* is a tag name taken form *Path* and its format is either *tag_name* or *tag_name*[attribute_name=attribute value]. N_{target} is the destination node to be inserted the given element. *Flag* is used as a flag for finding an error situation. Its initial value is set to '0' whenever a node visiting begins at each level. *Flag* is set to '1' when an input tag name (*P_Eelement*) in *Path* at some level *l* is equal to one of the tag names for the nodes in the index tree at the same level *l*. The detailed insertion

algorithm is described below.

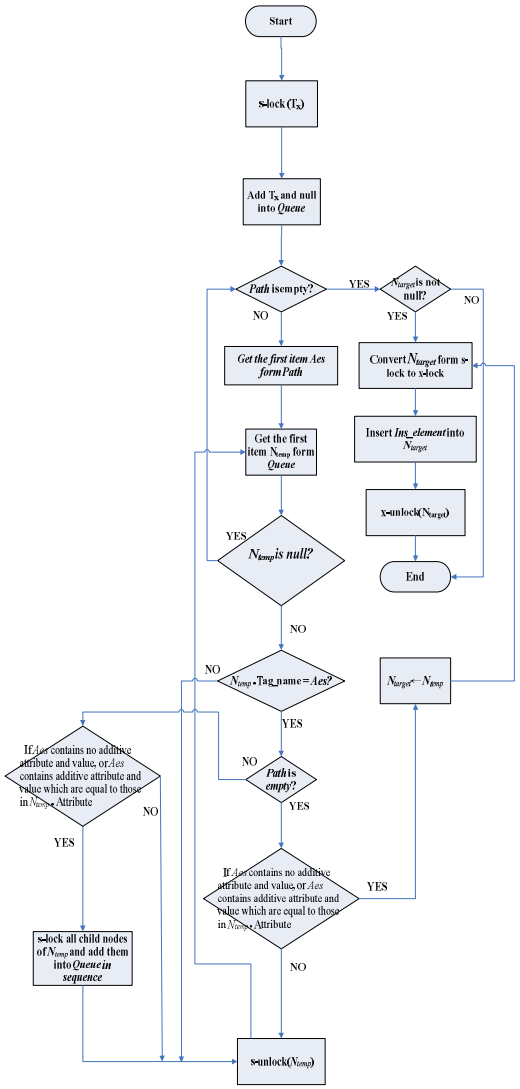

Figure 5. The flow chart of insert algorithm.

```
Algorithm Insert(Tx, Ins_Element, Path)
Input:
  NODE POINTER Tx;
  ELEMENT Ins_Element;
  STRING Path;
Begin
        QUEUE Queue;
        NODE POINTER N_temp;
        STRING Aes; //An element string
                        abstracted form Path//
        NODE POINTER N_target;
        INTEGER Flag;
01.   Flag ← 0; //0: incorrect i/p tag name//
02.   N_target ← null;
03.   s-lock(Tx);
04.   add_node (Tx, Queue);
05.   add_node (null, Queue); // null is a dummy
                        node, to
        separate nodes at different levels.//
06.   while Path is not empty, do
07.   Aes ← get_tagname(Path); //Get the first
          item form Path//
08.    N_temp ← get_node(Queue);
        //Get the first item form Queue//
09.    if N_temp = null, then // level changing //
10.      if Flag = 0, then
11.            print("Invalid tag name or path
data");
12.        for each node i in Queue, do
13.            s-unlock(i); //unlock the nodes
                          being locked//
14.        end for;
15.        return;
16.      end if;
17.      add_node(null, Queue);
18.      Flag ← 0; //reset Flag value//
19.    else
20.      if N_temp.Tag_name = Aes, then
21.        Flag ← 1; //1:correct i/p tag name//
22.        if path is empty, then
23.          if Aes contains no additive
               attribute and value, or Aes
               contains additive attribute and
               value which are equal to those
               in N_temp.Attribute, then
24.            N_target ← N_temp; //find target
node //
25.            break;
26.          end if;
27.        else // path is not empty//
28.          if Aes contains no additive
               attribute and value, or Aes
               contains additive attribute and
               value which are equal to those
               in N_temp.Attribute, then
29.            if N_temp is not a leaf node, then
30.              for each child[i] of N_temp, do
31.                s-lock(child[i]);
                     // lock-coupling//
32.                add_node(child[i],
                          Queue);
33.              end for;
34.            end if;
35.          end if;
36.      end if;
37.      s-unlock(N_temp);
38.      goto Line08;
39.    end if;
40.  end while;
41.  if N_target is not null, then
42.      convert(s, x, N_target); //convert the
          Lock on N_target form s-lock to x-lock//
43.      add Ins_Element to N_target as a child
node;
44.      x-unlock(N_target);
45.  end if;
End insert.
```

3.3 Modification Algorithm

Given the index tree, an element, and a path, the modify transaction descends the tree along the path to a target node and replaces the old element with the new one. The nodes on the path must be s-locked sequentially and released immediately after they are processed. The target node must be converted form s-locked to x-locked before the modification. The simple flow chart of the modification algorithm is shown in Figure 6. The meanings and functions of the parameters and variables used in this algorithm are the same with those in the insertion algorithm except the *Old_Element*. *Old_Element* is an old element to be replaced by the new element. The detailed modification algorithm is described below.

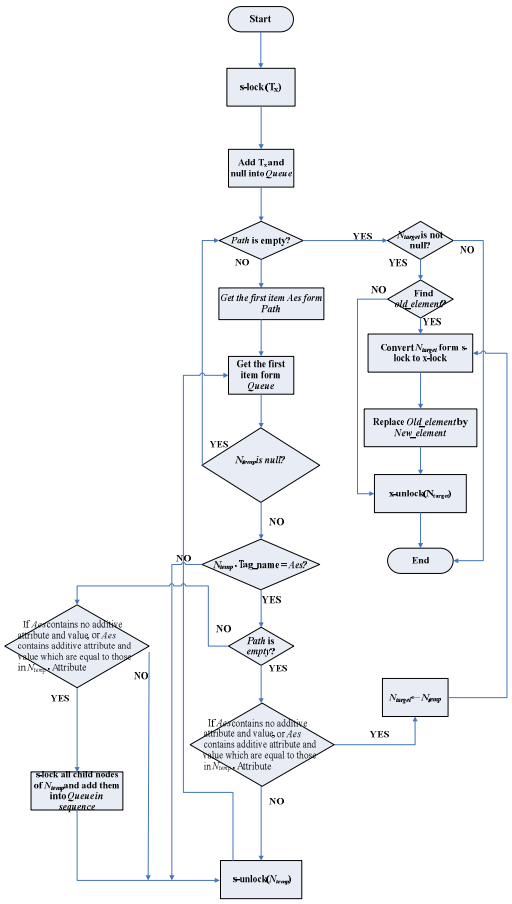

Figure 6. The flow chart of modification algorithm.

Algorithm Modify(*Tx*, *Old_Element*, *New_Element*, *Path*)
Input:
NODE POINTER *Tx*;
ELEMENT *Old_Element*;
ELEMENT *New_Element*;
STRING *Path*;

Begin
QUEUE *Queue*;
NODE POINTER N_{temp};
STRING *Aes*;
NODE POINTER N_{target};
INTEGER *Flag*;
01. *Flag* ← 0;
02. N_{target} ← null;
03. s-lock(*Tx*);
04. add_node (*Tx*, *Queue*);
05. add_node (*null*, *Queue*);
06. while *Path* is not empty, do
07. *Aes* ← get_tagname(*Path*);
08. N_{temp} ← get_node(*Queue*);
09. if N_{temp} = null, then
10. if *Flag* = 0, then
11. print("Invalid tag name or path data");
12. for each node i in *Queue*, do
13. s-unlock(i);
14. end for;
15. return;
16. end if;
17. add_node(*null*, *Queue*);
18. *Flag* ← 0;
19. else
20. if N_{temp} . Tag_name = *Aes*, then
21. *Flag* ← 1;
22. if *path* is empty, then
23. if *Aes* contains no additive attribute and value, or *Aes* contains additive attribute and value which are equal to those in N_{temp} . Attribute, then
24. N_{target} ← N_{temp};
25. break;
26. end if;
27. else
28. if *Aes* contains no additive attribute and value, or *Aes* contains additive attribute and value which are equal to those in N_{temp} . Attribute, then
29. if N_{temp} is not a leaf node, then
30. for each child[i] of N_{temp}, do
31. s-lock(child[i]);
32. add_node(child[i], *Queue*);
33. end for;
34. end if;
35. end if;
36. end if;

37. s-unlock(N_{temp});
38. goto Line08;
39. end if;
40. end while;
41. if N_{target} is not null, then
42. for each child i in N_{target}, do
43. if child i = *Old_Element* , then
44. convert(s, x, N_{target});
45. N_{target}← *New_Element*;
46. x-unlock (N_{target});
47. return;
47. end if;
48. end for;
49. s-unlock (N_{target});
51. end if;

End Modify.

4. Conclusion

This paper proposes search, insertion, and modification algorithms with concurrency control mechanism for XML document access. Two lock types, share lock and exclusive lock, are used to implement concurrency control. With the breadth-first search, the search transaction finds level by level some elements which contain all the input keywords. Only the s-lock is used in the search algorithm. Given tag names and/or attributes in a path, the insertion or modification transaction can find out a suitable node to insert or replace the given element. Both s-lock and x-lock are used in the insertion and modification algorithm. With the three concurrency control algorithms, the XML document can be accessed concurrently by many transactions without any occurrence of unpredictable mistake.

References

[1] M. ARENAS, L. LIBKIN, "XML data exchange: Consistency and query answering" *Journal of the ACM (JACM) Vol. 55 Article No. 7*, May 2008.

[2] J.K.CHEN, Y.F.HUANG, Y.H.CHIN,"A Study of Concurrent Operations on R-Trees" *Information Sciences, Volume 98, Number 1*, pp. 263-300, 1997.

[3] R. Elmasri and S. B. Navathc, "*Fundamentals of Database Systems, 4th Education*." Addison Wesley, 2003.

[4] G. Governatori, B. Stantic , and A. Sattar1, "Handling of Current Time in Native XML Databases", *17th Australasian Database Conference Vol. 49*, pp. 175-182,2006.

[5] A. Heuer, H. Meyer, A. C. Schering, "Managing Highly Correlated Semi-Structured Data"*Proceedings of the ACM first Ph.D. workshop in CIKM, pp* 101-108 , 2007.

[6] M. Kudo, J. Myllymaki, H. Pirahesh, N. Qi, "A function-based access control model for XML databases", *Proceedings of the 14th ACM international conference on Information and knowledge management,* pp 115 – 122, 2005.

[7] E. J. Lu, R.H. Tsai, and S.H. Chou," An Empirical Study of XML/EDI", *Journal of Systems and Software Volume: 58, Issue: 3*, pp. 271-279, 2001.

[8] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. D. Ngoc, "Exchanging Intensional XML data", *ACM Transactions on Database Systems Vol. 30, Issue 1, pp*. 1-40, 2005.

[9] S. Natu and J. Mendonca "Digital Asset Management Using A Native XML Database Implementation" *Proceedings of the 4th Conference on Information Technology Curriculum CITC4 '03*, pp. 237-241, 2003.

[10] R. Rajugan, E.Chang, T.S. Dillon, L. Feng,"A layered view model for XML repositories & XML data warehouses" *Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference*, pp206-213, 2005.

[11] N. Wiwatwattana, H.V. Jagadish, L.V.S. Lakshmanan, D. Srivastava, "X^ 3: A Cube Operator for XML OLAP" Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference, pp916-925, 2007

[12] W3C, ExtensibleMarkupLanguage(XML)1.1, http://www.w3.org/TR/2006/REC-xml11-20060816/

[13] W3C, Standard Generalized Markup Language, *http://www.w3.org/MarkUp/SGML/*

[14] X. Yin and T. B. Pedersen, "Evaluating XML-Extended OLAP Queries Based on a Physical lgebra",*7th ACM International Workshop on Data Warehousing and OLAP*, pp.73-82, 2004.

[15] Boyi Xu, Lihong Jiang, Fanyuan Ma "On the new B to B E-business Enabling platform: cnXML in China", *ACM International Conference Proceeding Series; Vol. 113 Proceedings of the 7th international conference on Electronic commerce*, pp. 681 – 684, 2005.