SXCCP+: Simple XML Concurrency Control Protocol for XML Database Systems

Krzysztof Jankiewicz[1]

Abstract: Increasing significance and popularity of XML is the main reason why many commercial object-relational database management systems (ORDBMS's) developed XML storage and processing functionality. Additionally, there are new types of specialized database systems known as 'native' XML database systems. As we know, concurrency control is one of the most important mechanisms in DBMS's. Unfortunately, concurrency control mechanisms used so far in commercially available native XML DBMS's offer very low degree of concurrency. The development of universal and effective concurrency control protocol for accessing XML data, with high degree of concurrency, is a necessary condition to growth of native XML databases.

The aim of this paper is a proposal of new concurrency control protocol in XML document access. This protocol is based on primitive operations which can be treated as unification platform for any of XML access methods.

Keywords: concurrency control, XML, databases, protocols

1. Introduction

Currently, XML document processing is one of the major areas in data processing technology. The popularity of XML is a result of its simplicity and elasticity. Due to these features, XML is the main standard in the complex, variable and semi structured data exchange. Increasing significance and popularity of XML is the main reason why many commercial object-oriented-relational DBMS's developed XML storage and processing functionality. Additionally, there are new types of specialized database systems known as 'native' XML database systems.

Concurrent and uncontrolled access to XML database systems, like in relational and object-oriented database systems, may lead to data inconsistency. Concurrency control problem was widely considered in the literature. The variety of correctness criteria for concurrency control in database systems and variety of concurrency control protocols were proposed. Conflict serializability is the main, commonly accepted concurrency control criterion. Developed protocols represent three main approaches: locking, time stamp ordering and optimistic. Mechanisms used so far in commercially available native XML DBMS are based on locking protocols and offer very low degree of concurrency and, thus, very low processing performance. There is an obvious need to develop new methods of concurrency control in XML database systems, which provide database consistency and acceptable degree of concurrency. These methods should provide acceptable performance taking

[1] Wydzial Informatyki i Zarzadzania, Politechnika Poznanska, Piotrowo 2, 60-965 Poznan e-mail: `Krzysztof.Jankiewicz@cs.put.poznan.pl`

```
<book isbn='KD-12345-XY'>
  <title>XML</title>
  <year>1999</year>
  <authors>
    <author>Smith</author>
    <author>Wilder</author>
  </authors>
</book>
```

Figure 1. XML document

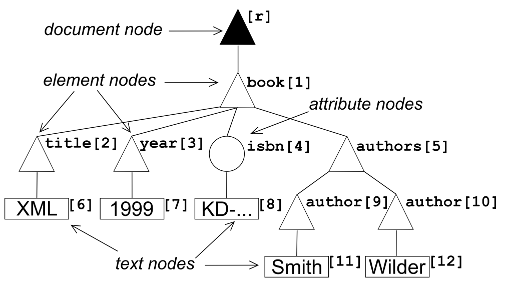

Figure 2. SXCCP+ data model

into account specific XML document features and variety of modification methods. There have been few propositions to solve this problem so far (Kanne, 2001 and 2003; Haustein, 2003; Grabs, 2002; Dekeyser, 2002; Choi, 2003; Jea, 2002). These solutions differ in degree of concurrency and assumed access methods to XML documents.

In our paper (Jankiewicz, 2006) we presented a brief survey of proposed concurrency control protocols and their critical analysis. The protocols' classification was also presented. The criteria of evaluation were defined and protocols' analysis was made on the basis of proposed criteria. Concurrency control protocols which were analyzed in (Jankiewicz, 2006) can be classified in several ways. For example, according to assumed access method to XML documents, we can distinguish following classes of protocols: based on DOM API and based on XPath standard. There is no protocol unassigned to the particular access method. The aim of this paper is the proposal of new concurrency control protocol for XML database systems, which is independent from access method.

The structure of this article is as follows. Section 2 presents proposal of new concurrency control protocol. Section 3 concentrates on conducted experiments and analysis of the results. In section 4 conclusions and directions for future work are presented.

2. SXCCP+

SXCCP+ protocol, presented in this section, is based on locking mechanism and two phase locking protocol. The main idea of the SXCCP+ is concurrency control, which will be independent from (and not assigned to) particular XML document access method. We assume that every access to XML document can be translated into set of primitive and indivisible equivalent operations. Only these primitive operations are taken into account by the SXCCP+. For example, we assume that every DOM API function, and every XPath operation, performed on XML document, can be expressed in one, or in sequence of primitive low-level operations. Fact that protocol is founded on primitive operations makes it autonomous from specific XML document access method. In the later part of this section we describe set of primitive operations of SXCCP+.

DOM API methods	Equivalent primitive operations
`n.firstChild`	`C(n); T(first(n))`
`n.previousSibling`	`T(previous(n))`
`n.getElementsByTagName`	`C(n); C(desc(n)); R(desc(n))`
`n.nodeName`	`R(n)`
`n.getAttribute(s)`	`C(n); R(attrs(n))`
`insertBefore(n',n)`	`I(n',parent(n),pos(n))`

Table 1. Conversion of DOM API methods to primitive operations

2.1. Data model

Data model used by the SXCCP+ to concurrency control is a simple DOM tree extension. Original DOM tree is extended by text nodes of attribute nodes. These additional nodes contain text value of attributes. In data model we distinguish following node types: document nodes, element nodes, attribute nodes and text nodes (which contain text value of elements and attributes). For example, data model for XML document from Fig.1 is given on Fig.2. Additional node types like processing instructions and comments are treated in the same way as text nodes, therefore are omitted in this paper.

Aim of introduction of new text node for attributes is to increase degree of concurrency and unification of protocol's rules which take place in the attribute and text node access. In the data model all nodes are lockable entities, on which concurrency control mechanism, due to execution of primitive operations, can acquire locks.

All primitive operations are performed transparently to users of DBMS by concurrency control mechanism on data model of XML document.

2.2. Primitive operations

We defined a set of primitive operations based on analysis of existing interfaces to XML document access. Before we introduce primitive operations, we define node content concept. *Node content* is (according to node type) a tag (element node) or name of attribute (attribute node) or text value (text node) or node value (other node types which can't have children nodes). In the SXCCP+ the following set of primitive operations is used: `C(n)` – test of child node existence, `T(n)` – node access (without access to its contents), `R(n)` – node content access, `U(n)` – node modification, it is an update of content of node, `D(n)` – node deletion, `I(n',n,l)` – insertion of `n'` node into `n` node at `l`th position of children of `n` node.

In order to use a SXCCP+ in real XML database system, it is necessary to define the method of assigning sequence of primitive operations equivalent to every operation existing in used XML interfaces. In the case of procedural interfaces we can use association matrix, where for each interface method we define sequence of equivalent primitive operations. Fragment of such association matrix for DOM API is presented in Tab.1. In the case of declarative interfaces, sequence of equivalent primitive operations can be obtained according to defined set of rules. We have defined such set of rules for XPath-based interfaces. These rules are based on semantics of XPath expressions. Before we present these rules, we introduce some definitions partially based on definitions introduced in (Jea, 2002). Each XPath location path L_j consists of location steps $S_{i,j}$, $1 \leq i \leq |L_j|$, where $|L_j|$ is a length of L_j. The syntax for a location step is as follows: `axisname::nodetest[predicate]`

The set of *context nodes* of a location step $S_{i,j}$ of location path L_j includes nodes that $S_{i,j}$ begins with. Context nodes are denoted by $C(S_{i,j})$.

The set of *opening nodes* of a location step $S_{i,j}$ of location path L_j, denoted by $O(S_{i,j})$, includes nodes satisfying `axisname` in $S_{i,j}$.

The set of *mid-result nodes* of a location step $S_{i,j}$ of location path L_j, denoted by $M(S_{i,j})$, is the selection of $O(S_{i,j})$ satisfying `nodetest` in $S_{i,j}$.

The set of *result nodes* of a location step $S_{i,j}$ of location path L_j, denoted by $R(S_{i,j})$, is the selection of $O(S_{i,j})$ satisfying `predicate` in $S_{i,j}$. In fact $R(S_{i,j}) = C(S_{i+1,j})$.

Let's notice that predicate of a location step $S_{i,j}$ of location path L_j can be expressed by another location path $L_{i,j}$ which consists of another location steps $S_{k,i,j}$, $1 \leq k \leq |L_{i,j}|$. $C(S_{1,i,j})$ of location path $L_{i,j}$ is $M(S_{i,j})$.

The *destination nodes*, denoted by $N_d(L_j)$ of location path L_j is $R(S_{|Lj|,j})$.

Now, we can present set of rules which assign the sequence of primitive operations equivalent to XPath location path L_j. In SXCCP+ protocol the sequence of primitive operations results from evaluation of every location step. At the beginning of location step $S_{i,j}$ evaluation, operation `T` is performed on $C(S_{i,j})$. Then, if `axisname` has a `child`, `descendant` or `descendant-or-self` format[2], operation `C` is performed on $C(S_{i,j})$. Then, according to `nodetest` function, operation `T` or `R` is performed on $O(S_{i,j})$. If `nodetest` function is expressed by `*`, then `T` operation is performed, otherwise `R` operation is performed.

XPath expressions are used in many XPath-based interfaces which can read XML document fragments and also modify them. XQuery and XUpdate are one of the most popular. XQuery and XUpdate expressions allow modification of destination nodes $N_d(L_j)$, of location path L_j, by following update operations: `IB(n',n)` – inserts new node `n'` before node `n`, `IA(n',n)` – inserts new node `n'` after node `n`, `IF(n',n)` – inserts new node `n'` as the first child of node `n`, `IL(n',n)` – inserts new node `n'` as the last child of node `n`, `IU(n',n,l)` – inserts new node `n'` as `l`th child of node `n`, `UT(n,t)` – updates text value of node `n` to `t` value, `RN(n',n)` – replaces node `n` to new node `n'`, `DN(n)` – deletes node `n`, `CN(n,t)` – changes node name `n` to `t` value. Sequence of primitive operations equivalent to each of update operations is presented in Tab.2.

Let's analyze the following examples.

Exemplary DOM API operations are presented on Fig.3. According to association matrix for DOM API mentioned before, following primitive operations

[2] Axisname which has `following` or `preceding` format as not allowed in SXCCP+ protocol

update operation	primitive operations	update operation	primitive operations
`IB(n',n)`	`I(n',parent(n),pos(n)-1)`	`UT(n,t)`	`U(text(n))`
`IA(n',n)`	`I(n',parent(n),pos(n)+1)`	`RN(n',n)`	`D(n); I(n', parent(n),pos(n))`
`IF(n',n)`	`C(n); I(n',n,1)`	`DN(n)`	`D(n)`
`IL(n',n)`	`C(n); I(n',n, max(pos(childs(n)))`	`CN(n,t)`	`U(n)`
`IU(n',n,l)`	`C(n); I(n',n,l)`		

Table 2. Sequence of primitive operations equivalent to XPath-based update operations in SXCCP+ protocol

```
(1) doc = getDocument();
(2) node = doc.getFirstChild();
(3) node = node.getLastChild();
(4) node = node.getLastChild();
(5) node.getNodeName();
(6) node = node.getFirstChild();
(7) node.setNodeValue('Speed');
```

Figure 3. DOM API example

```
for $i in /book[title/text()='XML']
            //author[1]
do rename $i as writer
```

Figure 4. XQuery example

are equivalent to each DOM operation: (1): `T([r])`; (2): `C([r])`, `T(book[1])`; (3): `C(book[1])`, `T(authors[5])`; (4): `C(authors[5])`, `T(author[10])`; (5): `R(author[10])`; (6): `C(author[10])`, `T([12])`; (7): `U([12])`. Exemplary XQuery expression is presented on Fig.4. It changes node names. XPath location path used in this expression has format L_1=`/book[title/text()='XML']//author` and has two location steps $S_{1,1}$=`child::book[title/text()='XML']`, and $S_{2,1}$=`descendant::author`. Additionally, location step $S_{1,1}$ has predicate which uses location path $L_{1,1}$=`title/text()`. Location path $L_{1,1}$ has two location steps $S_{1,1,1}$=`child::title` and $S_{2,1,1}$=`child::text()`. On destination nodes of location path L_1 XQuery expression performs `CN` operation. Let's analyze primitive operations assigned to location step $S_{1,1}$. Operation `T` is performed on $C(S_{1,1})$ nodes – `[r]`. According to `axisname` format, `C` operation is performed on `[r]` node. Opening node $O(S_{1,1})$ of location step $S_{1,1}$ is `[1]`, and according to `nodetest` format `R` operation is performed on this node. The same node (`[1]`) is a $M(S_{1,1})$. Before we get $R(S_{1,1})$ of location step $S_{1,1}$, we have to evaluate its predicate. $C(S_{1,1,1})$ node of location step $S_{1,1,1}$ is `[1]`, and `T` operation and then `C` operation is performed on this node. Opening nodes $O(S_{1,1,1})$ of location step $S_{1,1,1}$ are `[2]`, `[3]`, `[5]`, and `R` operation is performed (according to `textnode` format) on this nodes. $M(S_{1,1,1})$ and $R(S_{1,1,1})$ of location step $S_{1,1}$ is `[2]`. The same node is a $C(S_{2,1,1})$, then `T` and `C` (according to `axisname` format) operation is performed on this node. Opening node $O(S_{2,1,1})$ of location step $S_{2,1,1}$ is `[6]`, and `R` operation is performed on this node. Context, opening, mid-result, results nodes and corresponding primitive

S	$C(S)$	$O(S)$	$M(S)$	$R(S)$	primitive operations
$S_{1,1}$	`[r]`	`book[1]`	`book[1]`	`book[1]`	`T([r])`, `R([1])`
$S_{1,1,1}$	`book[1]`	`title[2]`, `year[3]`, `authors[5]`	`title[2]`	`title[2]`	`T([1])`, `R([2])`, `R([3])`, `R([5])`
$S_{2,1,1}$	`title[2]`	`[6]`	`[6]`	`[6]`	`T([2])`, `R([6])`
$S_{2,1}$	`book[1]`	`title[2]`, `year[3]`, `authors[5]`, `author[9]`, `author[10]`, `[6]`, `[7]`, `[11]`, `[12]`	`author[9]`, `author[10]`	`author[9]`, `author[10]`	`T([1])`, `R([2])`, `R([3])`, `R([5])`, `R([9])`, `R([10])`, `R([6])`, `R([7])`, `R([11])`, `R([12])`
`CN`	$N_d(L_1)$: `author[9]`, `author[10]`				`U([9])`, `U([10])`

Table 3. Context, opening, mid-result, results nodes and corresponding primitive operation

operation for analyzed XQuery expression are presented at Tab.3.

2.3. Lock modes

Concurrency control mechanism of SXCCP+ is based on the two phase locking method. For each of primitive operations, introduced in previous subsection, corresponding *operational basic lock mode* was defined. Thus, SXCCP+ uses following operational basic lock modes: `LC` – child test lock, `LT` – traversal lock, `LR` – read lock, `LU` – update lock, `LD` – delete lock, `LW` – write lock, corresponding to primitive operations: `C`, `T`, `R`, `U`, `D`, `I`, respectively.

Additionally, we introduce the set of *operational tree lock modes*. These lock modes prevent lock escalation when performed operation has descendant nodes range. For example, in XQuery expression presented on Fig.4, location step $S_{2,1}$=`descendant::author` implies `R` operation performed on all descendant nodes of `n` (`[1]`) node. In such cases, instead of acquiring operational basic locks on all descendant nodes of `n` node, SXCCP+ acquires operational tree lock on `n` node only. SXCCP+ uses following operational tree lock modes: `LCC` – child test tree lock, `LTT` – traversal tree lock, `LRR` – read tree lock, `LUU` – update tree lock, `LDD` – delete tree lock, `LWW` – write tree lock.

Additionally, for each operational lock mode corresponding *intentional lock mode* was defined. Therefore, we have the following intentional lock modes: `LIC` – intentional child test tree lock, `LIT` – intentional traversal lock, `LIR` – intentional read lock, `LIU` – intentional update lock, `LID` – intentional delete lock, `LIW` – intentional write lock, `LICC` – intentional child test tree lock, `LITT` – intentional traversal tree lock, LIRR – intentional read tree lock, LIUU – intentional update tree lock, LIDD – intentional delete tree lock, LIWW – intentional write tree lock. Additionally, SXCCP+ uses two intentional lock modes LICW and LICWW. Meaning of these lock modes is presented in next subsection.

2.4. Locking rules

Concurrency control mechanism controls every transaction which performs access to documents in XML database system. SXCCP+ for every primitive operation (implied by operations of transactions) acquires locks with corresponding modes in two phases: the *phase of intentional locks* which is followed by the *phase of operational locks*. When primitive operation is performed on n node, during the phase of intentional locks, the intentional locks are acquired on all ancestor nodes of n node. The intentional locks are acquired from root to parent node order. When all required intentional locks are granted, operational lock is acquired on n node. When any of required locks can not be acquired due to incompatibility with other lock acquired by other transactions, then transaction and its locking mechanism stop, and wait for the release of incompatible lock. When locking mechanism stops, all previously acquired locks are held. Mode of the intentional locks, as well as mode of the operational locks, corresponds to the type of primitive operation which is performed on XML document, and its range. For example, XQuery update expression on Fig.4 implies U operation performed on the author[9] node. Thus, it requires intentional LIU locks on: [r], book[1] and authors[5] node, then, it requires LU lock on author[9] node.

Slightly different situation is when I operation is performed. In this case, two intentional lock modes LIW and LICW (LIWW and LICWW, when operation has descendant nodes range) are used. During the phase of intentional locks LIW (LIWW) lock mode are acquired on all ancestor nodes except parent node, then LICW (LICWW) lock mode is acquired on a parent node. Lock modes LICW and LICWW are used to avoid phantom anomaly. Tab.4 presents locks acquired by SXCCP+ protocol according to primitive operations and their range.

2.5. Lock matrix compatibility

As a result of the analysis of commutation of primitive operations, the complete compatibility matrix was built. Due to page limitation it is not presented in this paper. Analysis of the complete lock compatibility matrix gives as results the following equivalence classes of lock modes: EQ_1= {LT, LIC, LIT, LICC, LITT}, EQ_2= {LTT, LCC}, EQ_3= {LIR, LIRR}, EQ_4= {LIU, LIUU}, EQ_5= {LIW, LID, LIWW, LIDD}, EQ_6= {LW, LD, LDD, LWW}, EQ_7= {LICW, LICWW}. Therefore, we introduce representative for each equivalence class. Let the representative of equivalence classes EQ_1, EQ_2, EQ_3, EQ_4, EQ_5, EQ_6, EQ_7 are lock modes LT, LTT, LIR, LIU, LIW, LW and LICW respectively. Lock modes not included in presented equivalence classes are, in the fact, members of the one element equivalence classes, and they

primitive operation	range of operation	acquired locks in SXCCP+ protocol
C	n	LIC(ancestor(n)), LC(n)
C	desc(n)	LICC(ancestor(n)), LCC(n)
T	n	LIT(ancestor(n)), LT(n)
T	desc(n)	LITT(ancestor(n)), LTT(n)
R	n	LIR(ancestor(n)), LR(n)
R	desc(n)	LIRR(ancestor(n)), LRR(n)
U	n	LIU(ancestor(n)), LU(n)
U	desc(n)	LIUU(ancestor(n)), LUU(n)
D	n	LID(ancestor(n)), LD(n)
D	desc(n)	LIDD(ancestor(n)), LDD(n)
I	n	LIW(ancestor(n)-parent(n)), LICW(parent(n)), LW(n)
I	desc(n)	LIWW(ancestor(n)-parent(n)), LICWW(parent(n)), LWW(n)

Table 4. Locks acquired by SXCCP+ protocol

are this class's representatives. After this introduction we can present on Tab.5 the lock compatibility matrix of SXCCP+ with the usage of the representatives of equivalence classes.

Lock modes, which are members of equivalence class, are replaced in SXCCP+ by equivalence class's representative. For example, `D` operation performed according to XQuery update expression `do delete /book/title`, requires `LIW` locks, instead of `LID` locks, on document node and `book[1]` node as well as `LW` lock, instead of `LD` lock, on `title[2]` node. It is due to the fact that `LID` lock mode is a member of EQ_5 where `LIW` lock mode is the equivalence class representative, and `LD` lock mode is a member of EQ_6 where `LW` lock mode is the equivalence class representative.

Additionally, SXCCP+ can use lock modes which are the combinational lock modes, and which could be used with the conversion of locks. Due to page limitation these lock modes, lock conversion matrix and lock compatibility matrix of combinational lock modes are not presented in this paper.

Let's analyze the following example. Assume that transactions T_2 and T_2 are performed concurrently. Transaction T_1 executes DOM API operations presented on Fig.3, whereas transaction T_2 executes XQuery expression presented on Fig.4. Tab.6 presents example of theirs realization and required locks. Realization from 1 to 4 executes smoothly – only node access (`T`) and node content access (`R`) operations are performed. Their locks are compatible. In 5 transaction T_1 performs `U` operation on `[12]` node, it requires `LIU` lock mode on `[r]`, `[1]`, `[5]`, `[10]` nodes. Unfortunately, transaction T_2 aquired `LRR` lock mode on `[4]` node. According to

	granted											
requested	`LT`	`LC`	`LR`	`LU`	`LW`	`LTT`	`LRR`	`LUU`	`LIR`	`LIU`	`LIW`	`LICW`
`LT`	+	+	+	+	−	+	+	+	+	+	+	+
`LC`	+	+	+	+	−	+	+	+	+	+	+	−
`LR`	+	+	+	−	−	+	+	−	+	+	+	+
`LU`	+	+	−	−	−	+	−	−	+	+	+	+
`LW`	−	−	−	−	−	−	−	−	−	−	−	−
`LTT`	+	+	+	+	−	+	+	+	+	+	−	−
`LRR`	+	+	+	−	−	+	+	−	+	−	−	−
`LUU`	+	+	−	−	−	+	−	−	−	−	−	−
`LIR`	+	+	+	+	−	+	+	−	+	+	+	+
`LIU`	+	+	+	+	−	+	−	−	+	+	+	+
`LIW`	+	+	+	+	−	−	−	−	+	+	+	+
`LICW`	+	−	+	+	−	−	−	−	+	+	+	+

Table 5. Lock compatibility matrix of SXCCP+

compatibiliy matrix (Tab.5) `LRR` lock mode is incompatible with `LIU` lock mode. Therefore transaction T_1 stops until transactions T_2 ends in 6.

3. Experimental results

Our experiments were conducted with support of XML database system simulator, which was created at Institute of Computing Science at Poznan University of Technology. This simulator was constructed due to needs of researches on concurrency control mechanisms in XML database systems.

3.1. Tested protocols

We conducted our experiments for the following concurrency control protocols: NO2PL, Node2PL, OO2PL, taDOM and SXCCP+. Protocols NO2PL, Node2PL, OO2PL introduced in (Kanne, 2001) have two versions: basic and extended. In extended versions of these protocols additional locks were added. These additional locks correspond with read content node and modification node operations. We performed experiments with extended versions of these protocols. Protocol taDOM has special lock mode U which supports read with potential write access and prevents granting further read locks. This lock mode decreases deadlock probability, but also decreases the degree of concurrency. Similar lock modes do not exist with other tested protocols, although they could. Therefore, in our tests we use version of taDOM protocol without U lock mode. Additionally, due to lack of lock conversion tables in NO2PL, Node2PL and OO2PL protocols, we do not do such conversions. This fact has influence on number of acquired locks, but this way this number is comparable between different protocols.

no.	operation	aquired locks	no.	operation	aquired locks
1	T_1: (1)–(3)	`LT([r])`, `LC([r])`, `LIT([r])`, `LT([1])`, `LIC([r])`, `LC([1])`, `LIT([1])`, `LT([5])`	4	T_2: $S_{2,1}$	`LT([1])`, `LRR([1])`
2	T_2: $S_{1,1}$, $S_{1,1,1}$, $S_{2,1,1}$	`LT([r])`, `LIR([r])`, `LR([1])`, `LIT([r])`, `LT([1])`, `LIR([1])`, `LR([2])`, `LR([3])`, `LR([5])`, `LIT([1])`, `LT([2])`, `LIR([2])`, `R([6])`	5	T_1: (6)–(7)	`LIC([5])`, `LC([10])`, `LIT([10])`, `LT([12])`, `LIU([r],[1],[5],[10])`, `LU([12])`
3	T_1: (4)–(5)	`LIC([1])`, `LC([5])`, `LIT([5])`, `LT([10])`, `LIR([5])`, `R([10])`	6	T_2: `CN`	`LIU([r],[1],[5])`, `LU([9])`

Table 6. Concurrent realization of transactions T_1 and T_2

3.2. Characteristic of XML document

Our experiments were performed on several types of XML documents. We present results of experiments performed on one XML document bench0065.xml, which was generated by xmlgen – The Benchmark Data Generator, created as a part XMark – An XML Benchmark Project (Schmidt, 2002).

3.3. Transaction classes

Access to XML documents was realized by set of transactions which were concurrently started. Cardinality of transactions set was changed from 1 to 49. Operations performed by transactions were based on DOM API. Each transaction was one out of four transaction classes. The choice of transaction class for each transaction was random with the same probability. Each of transaction classes has characteristics as follows. OneDocumentPointModify – transaction navigates from root node to one of leaf nodes, and modifies its content. OneDocumentRandomLevelModify – transaction navigates from root node to node on one of random levels, and modifies the node's content. OneDocumentRandomLevelDelete – transaction navigates from root node to node on one of random levels, and removes destination node. OneDocumentRandomLevelInsert – transaction navigates from root node to node on one of random levels, and inserts a new node as a child or neighbor of destination node.

3.4. Results

Presented figures reflect the results of performed experiments as the average value of the following chosen measures: time of test (Fig.5) – the time required to service all concurrent transactions as a function of cardinality of transactions' set, number

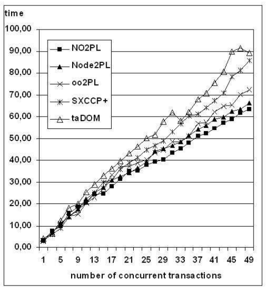

Figure 5. Time of test

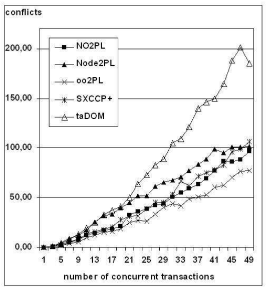

Figure 6. No. of conflics

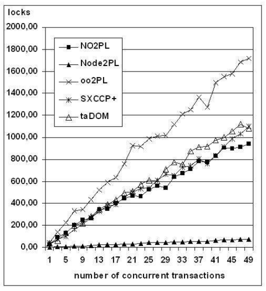

Figure 7. No. of locks

of conflicts (Fig.6) – the number of conflict situations as a function of cardinality of transactions' set. Each conflict situation was resolved by restart or waiting of transaction because of WAIT-DIE algorithm was used, number of locks (Fig.7) – the maximum number of locks held by the transactions as a function of cardinality of transactions' set.

3.5. Analysis of the results

Let us take a closer look at the results. As we can see in Fig.7, number of locks held by OO2PL protocol is higher then in other protocols. This difference is due to the number of node locks which are acquired for every operational lock in OO2PL. Number of locks held by SXCCP+ is not lower then in other protocols. It is because of their intentional locks. These intentional locks have crucial meaning because they enable SXCCP+ usage with other XML interfaces and they enable operational tree lock mode usage which can decrease number of locks in many cases (tree lock modes were not used in presented experiments). Fig.5 and Fig.6 reflect that the SXCCP+ does have medium ability to lead transaction to the successful commitment, but SXCCP+ does not allow phantom anomaly, and gives serializable realizations, and therefore it must be more restrictive then OO2PL, NO2PL or Node2PL which allow phantom anomaly.

4. Conclusions and future work

In this paper we have presented a new locking protocol for concurrency control access in XML database systems named SXCCP+. It is the first protocol which is not assigned to any particular XML interface. It is based on primitive and indivisible operations which may be treated as components of any operations of any XML access interface. It means that SXCCP+ is the protocol which can be treated as general and neither assigned to nor dependent from particular XML interface. This fact has the key meaning for most XML database systems, when different interfaces coexist. Moreover, presented results of conducted experiments show that SXCCP+ is not worse then specialized protocols, assigned to particular interface, like taDOM, OO2PL, NO2PL, and Node2PL.

Our experiments presented in this work are focused on DOM API based protocols. Therefore, in our next work we will conduct series of experiments which examine SXCCP+ with other concurrency control protocols which are based on XPath expressions. Then, we plan to introduce some modifications into SXCCP+ which give as results higher degree of concurrency.

References

Choi, E.H., Kanai, T. (2003) *Xpath-based concurrency control for xml data.* In: Proceedings of the 14th Data Engineering Workshop (DEWS 2003), Kaga city, Ishikawa, Japan

Dekeyser, S., Hidders, J. (2002) *Path locks for xml document collaboration.* In: WISE '02: Proceedings of the 3rd International Conference on Web Information Systems Engineering, Washington, DC, USA, IEEE Computer Society 105–114

Grabs, T. and Böhm, K. and Schek, H.J. (2002) *Xmltm: Efficient transaction management for xml documents.* In: CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management, New York, NY, USA, ACM Press 142–152

Haustein, M.P. and Härder, T. (2003) *tadom: A tailored synchronization concept with tunable lock granularity for the dom api.* In: ADBIS. 88–102

Jankiewicz, K. (2006) *Survey and analysis of concurrency control methods for xml database systems.* In: Proceedings of the 5th International Conference MISSI'06 - Multimedia and Network Information Systems, Wroclaw, Poland 51–66

Jea, K.F. and Chen, S.Y. and Wang, S.H. (2002) *Concurrency control in xml document databases: Xpath locking protocol.* In: ICPADS '02: Proceedings of the 9th International Conference on Parallel and Distributed Systems, Washington, DC, USA, IEEE Computer Society 551

Kanne, C.C. and Moerkotte, G. and Helmer, S. (2003) *Lock-based protocols for cooporation on xml documents.* Technical Report Reihe Informatik 06/2003, University of Mannheim, Germany

Kanne, C.C. and Moerkotte, G. and Helmer, S. (2001) *Isolation in xml bases.* Technical Report Reihe Informatik 15/2001, University of Mannheim, Germany

Schmidt, A. and Waas, F. (2002) *Xmark: A benchmark for xml data management.* In 28th International Conference on Very Large Data Bases, Hong Kong, China. 974–985