

## Singapore Management University Institutional Knowledge at Singapore Management University

---

Research Collection School Of Information Systems

School of Information Systems

---

12-2003

# XStamps: A multiversion timestamps concurrency control protocol for XML data

Khin-Myo WIN


Wee-Keong NG

Ee Peng LIM

Singapore Management University, [eplim@smu.edu.sg](mailto:eplim@smu.edu.sg)

**DOI:** <https://doi.org/10.1109/ICICS.2003.1292748>

Follow this and additional works at: [https://ink.library.smu.edu.sg/sis\\_research](https://ink.library.smu.edu.sg/sis_research)

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

---

### Citation

WIN, Khin-Myo; NG, Wee-Keong; and LIM, Ee Peng. XStamps: A multiversion timestamps concurrency control protocol for XML data. (2003). *2003 International Conference on Information, Communications and Signal Processing and Pacific Rim Conference on Multimedia 4th ICICS-PCM 2003, 15-18 December 2003, Singapore*. 3, 1650-1654. Research Collection School Of Information Systems.

**Available at:** [https://ink.library.smu.edu.sg/sis\\_research/921](https://ink.library.smu.edu.sg/sis_research/921)

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email [libIR@smu.edu.sg](mailto:libIR@smu.edu.sg).

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/4072509>

# XStamps: A multiversion timestamps concurrency control protocol for XML data

**Conference Paper** · January 2004

DOI: 10.1109/ICICS.2003.1292748 · Source: IEEE Xplore

---

CITATIONS

4

READS

204

**4 authors**, including:



**Wee Keong Ng**

Nanyang Technological University

**427** PUBLICATIONS **5,028** CITATIONS

[SEE PROFILE](#)



**Ee-Peng Lim**

Singapore Management University

**448** PUBLICATIONS **9,087** CITATIONS

[SEE PROFILE](#)

**Some of the authors of this publication are also working on these related projects:**



ABECOS: Agent Based E-Commerce System [View project](#)



Job analytics [View project](#)

# XStamps: A Multiversion Timestamps Concurrency Control Protocol for XML Data \*

Khin-Myo Win Wee-Keong Ng Qincai Liu Ee-Peng Lim  
Nanyang Technological University  
Nanyang Avenue, Singapore 639798, SINGAPORE  
{p121902, awkng, 145935763, aseplim}@ntu.edu.sg

## Abstract

With the tremendous growth of XML data over the Web, efficient management of such data becomes a new challenge for database community. Several data management solutions, proposed in recent years, extend the capability of traditional database systems to meet the needs of XML data while alternative approaches introduce new generation databases, named native XML database management systems. Although traditional databases have mature transaction management and concurrency control techniques, there still need tailored techniques for native XML databases in order to deal with distinct characteristics of XML. In this paper, we propose XStamps, a multiversion timestamps concurrency control protocol for XML data, which is integrated in NextDB, a native XML database management system. XStamps is designed based on multiversion timestamps protocol, and additional features are added to enable flexible control of the isolation level of transactions and allow such transactions to commit early. Experimental results show that XStamps works well with XML data and provides performance gain over other concurrency protocols like Tree and Two-phase locking.

**Keywords:** Transaction Management for XML, Concurrency Control Protocol, Native XML Data Management

## 1. Introduction

The volume of semistructured data over the Web has been rapidly increasing, and XML becomes popular for representing such data and emerges as a *de facto* standard for data exchange among various applications. With the tremendous growth of XML data, efficient management of such data becomes a new challenge for database community. In order to deal with this problem, several researchers proposed various alternatives in recent years. Most solutions extend the capability of traditional databases such as relational and object-oriented database systems to meet the needs of XML data. Although these systems provides mature and robust data management features, they are not ideal solutions for XML which has a characteristic of flexible structure since they force all data to adhere with a predefined rigid schema. Additional layer, in this case, is necessary

\*This work was supported in part by The Singapore Research and Education Network (SingAREN) Broadband 21 Program under Project number ICT/00/013/011.

to do mapping from XML data into their own data structure and vice-versa. This transformation is time consuming and causes performance degradation.

In order to overcome such deficiencies, new data models which are specifically designed for storing XML data are proposed in recent years. These systems are tailored to maintain XML data in its native hierarchical structure in order to eliminate data mapping and transformation processes [9]. It speeds up query processing since XML queries can be directly manipulated without converting to different query statements, like SQL.

For a database system, transaction management and concurrency control are some of key components that can affect the performance of the system. Although traditional systems have mature techniques for transaction management and concurrency control, there still need tailored techniques for native XML databases to deal with distinct characteristics of XML. Currently, there is no standardized method to partly update XML documents due to the lack of efficient concurrency control protocols that would ensure the correctness and efficiency of concurrent operations. This situation motivates us to develop a transaction management system with elegant concurrency control algorithm for XML data.

In this paper, we propose XStamps, a concurrency control protocol for NextDB, a native XML database management system. XStamps is designed based on multiversion timestamps protocol, and additional features are added to enable flexible control of the isolation level of transactions and allow such transactions to commit early.

This paper is organized as follows. Section 2 defines data model and basic operations of XStamps, and Section 3 covers the design of XStamps, and outlines the concurrency control algorithm. The performance evaluation and throughput comparisons among XStamps, Tree and Two-phase locking protocols are presented in Section 4, and Section 5 describes related work. Section 6 concludes the paper with discussion on future work.

## 2. Data Model and Basic Operations

XStamps is a part of the transaction management system which is integrated in NextDB [11, 12]. NextDB is specifically designed for storing XML data, and its database components are tailored to deal with distinct features and characteristics of XML.

NextDB data model used in XStamps is a tree-like structured model where all elements and values are stored as nodes linked by references.

## 2.1 Data Model of XStamps

The storage model of NextDB focuses on the scenario that all nodes from multiple XML documents are clustered according to its paths, and collectively stored together in physical blocks so that necessary space can be reduced and user queries can be resolved with less I/O accesses while loading and scanning required data from physical storage [10, 11]. In NextDB, all element nodes and value nodes are stored in node repository and value repository respectively, and path index and node groups are embedded in the storage structure to support direct access to the desired nodes. Schema tree and value index are also included to deal with queries having regular path expressions and predicate over values. The overall storage and index architecture of NextDB is depicted in Fig. 1.

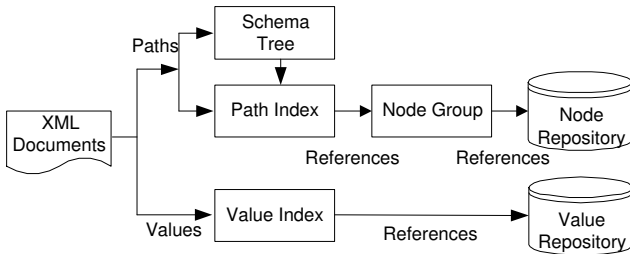


Figure 1: The Storage and Index Architecture of NextDB

Generally, the data model in NextDB is specifically tailored to maintain XML documents in its native tree-structured format. Each node is assigned with a unique node ID, and stored in a record together with ID of the document the node belongs to. Parent-child relationship between each pair of nodes is preserved so that the entire tree can be traversed back and forth.

## 2.2 Basic Operations of XStamps

A transaction is a set of basic operations that are applied to database items. When we determine a set of basic operations for XStamps, we consider the following design goals:

- *Completeness* - Any XML document that conforms to our data model can be built by applying a sequence of basic operations.
- *Soundness* - Applying a sequence of basic operations to a valid XML document will result in another valid XML document.
- *Unambiguity* - The effects of the basic operations and the necessary parameters should be clearly defined in order to support the manipulation of the data model.
- *Efficiency* - Any common and useful tasks can be expressed efficiently as a sequence of basic operations.

Basically, only two operation, READ and WRITE, for XStamps are insufficient to meet our design goals but we define four basic operations: READ, UPDATE, INSERT and DELETE

abbreviated as  $R$ ,  $U$ ,  $I$ , and  $D$  respectively in order to provide accuracy and flexibility. The first two operations do not change the structure of the document but the last two do.

In XStamps, read and traverse operations are treated as the same, and defined by read operation  $R$ . An important point is that the effect of  $R$  is different from that of a formal read operation. For a hierarchical structure, concurrency control algorithms generally define an operation on a granule, i.e., the effect of the operation is the entire granule, including all sub-granules. In our case, we may not always read the entire sub-tree when reading or traversing a node; thus,  $R$  operation performs only at a particular node. The purpose of reading a node is either for reading the content associated with it or for traversing forward and backward.

$U$  operation in XStamps is also slightly different from a formal update operation. As mentioned,  $U$  updates the data, but not the structure of the XML tree. In other words,  $U$  operation always performs over the leaf nodes in XML tree, which contains values or contents.  $I$  operation appends a node to the XML tree conforming to the definition of DTD. Iterated  $I$  can be used to insert a branch or a group of nodes. The effect of  $D$  operation is different from other because it deletes the entire branch rooted by that node.

## 3. XStamps Protocol

The concept of maintaining multiple versions of data to increase the degree of concurrency was introduced since 1970s, and it was formalized and extended to be Multiversion Timestamps Ordering Protocol (MTOP) [6]. Under MTOP, each transaction is assigned a unique timestamp upon initiation, and multiple versions of each database item  $X$ , one for each time the  $X$  has been written, are maintained. Each version  $V$  of  $X$  has a read-timestamp  $RTS(V)$ , that is the greatest timestamp among transactions which read that version, and a write-timestamp  $WTS(V)$ , that is the greatest timestamp among transactions which write that version. An attempt to read a data item  $X$  is always successful if there exists a version  $V'$  with  $WTS(V') < TS(T)$  and no other version  $V''$  with  $WTS(V') < WTS(V'') < TS(T)$ . An attempt to write a data item  $X$  is successful if the version  $V$  of  $X$  with  $WTS(V)$  maximal, subject to  $WTS(V) < TS(T)$ , also has  $RTS(V) \leq TS(T)$ ; otherwise  $T$  is aborted and restarted with a new timestamp [6].

The XStamps protocol is designed based on MTOP by adding new features. The reason for choosing MTOP is that it has several advantages; it provides high effective concurrency levels, deadlock-free operation (no or limited blocking) and efficient operation unless there is a conflict. In addition, timestamps-based protocols perform better in a low conflict environment because the number of rollbacks is less when the degree of conflicts is low. Unlike timestamps-based protocols which require transactions to commit in timestamps order, the XStamps is designed to make use of its new features that allow transactions to commit early.

### 3.1 Features of the XStamps Protocol

There are two features that make XStamps distinct from traditional timestamps protocols. The first feature is the classification of transactions. Upon initiation, a transaction is classified to be either a read or write transaction. The criterion is that it is a write transaction if a transaction contains at least one action that belongs to  $\{I,U,D\}$ , otherwise, it is a read. Read transactions operate on the read version of data while write transactions operate on the write version.

The second feature is the creation of two parameters: *safety coefficient (SCO)* and *safety threshold (STH)*. In XStamps, a safety coefficient is assigned to each database item (node in XML) and a safety threshold is assigned to each transaction upon initiation. *SCO* indicates the certainty of how sure the latest write transaction is able to successfully complete without rollback, and *STH* indicates the strictness of the transaction. For the environment where a transaction needs the most up-to-date and absolutely correct data, *STH* should be a very high value, while it can be a lower value for the environment where returning of slightly old data is not matter or the correctness of data is not important (it does not mean the high chance of getting the incorrect data). The advantage is obvious that it allows early transaction commit and more flexible control.

With the introduction of *SCO* and *STH* parameters, an issue arose here is that how to decide a good *SCO* value. We consider *SCO* as a function of a composite variable  $F$ , where  $F$  is a composition of factors carrying different weights. Factors are the system state information that determines whether an action can be physically unrealizable or not based on system information. Some of such information includes:

- Number of write operations in a write transaction.
- Total number of transactions in the database.
- Contention level that is decided by the:
  - size of database;
  - length of transaction in terms of number of operations;
  - average number of write operations in each transaction

With this set of factors, an efficient estimation algorithm can be designed to estimate the next state of the system based on current situation. Certainly, designing the algorithm for estimating a good *SCO* value is not simple and straightforward since it involves thorough investigation against several factors. Instead, we set *SCO* to the lowest value permitted before the current write transaction is committed, and reset it to maximum value when it completes successfully. In addition, a highest *STH* value is set to all transactions as default in order to provide accessing to the most updated data.

### 3.2 Data Access Rules in XStamps

We define the following access rules to guarantee the serializability of transactions and achieve a high degree of concurrency.

**Transaction Classification Rule.** Upon entry to the transaction management system, a transaction is classified as either a read or a write transaction. A transaction is a write transaction if and only if it contains at least one non-read operation.

**Data Node Access Rule.** We define the following algorithm to guarantee the data accesses.

1. If a transaction  $T$  issues a read request  $R(N)$  on node  $N$ ,
  - a)  $TS(T) \geq WTS(N)$ 
    - i) If Safety Coefficient of  $N$  is greater than or equal to the safety threshold of  $T$ , i.e.  $SCO(N) \geq STH(T)$ , and  $N$  is not marked as deleted,  $R(N)$  is granted and  $RTS(N)$  is set to  $TS(T)$  if  $TS(T) > RTS(N)$ . If  $SCO(N) \geq STH(T)$  but  $N$  is marked as deleted, operation is cancelled, transaction continues.
    - ii) If  $SCO(N) < STH(T)$ , transaction needs to wait until  $SCO(N) \geq STH(T)$ .
  - b)  $TS(T) < WTS(N)$ 
    - i) If  $SCO(N) \geq STH(T)$  and  $N$  is marked as deleted, if old write timestamp when it is marked  $WTS'(N) \leq TS(T)$ , grant  $R(N)$ . If  $SCO(N) \geq STH(T)$  and  $N$  is marked as inserted,  $R(N)$  is cancelled and transaction continues.
    - ii) Find a latest version  $N'$  of  $N$  with no version  $N''$  fall in between  $N$  and  $N'$  that has  $TS(T) \geq WTS(N')$  and repeat  $R(N)$  on this version. If  $N'$  can not be found, rollback.
2. If a transaction  $T$  issues a write request  $I(N)$ ,  $D(N)$  or  $U(N)$ ,
  - a)  $TS(T) \geq WTS(N)$  and  $TS(T) \geq RTS(N)$ 
    - i) If request is  $U(N)$   
Create a new node  $N'$  based on  $N$  with the updated value, set  $RTS(N')$  to 0 and  $WTS(N')$  to  $TS(T)$ , update  $SCO(N')$ .
    - ii) If request is  $I(N)$ , construct a new node  $N'$  and set  $RTS(N')$ ,  $WTS(N')$ ,  $SCO(N')$  accordingly, insert a new mark to  $N'$  and append  $N'$  as a child of  $N$ .
    - iii) If request is  $D(N)$ , mark node  $N$  as deleted if  $N$  does not have any mark, set  $WTS(T)$   $TS(T)$  and update  $SCO(N)$ . Wait otherwise until mark disappears.
  - b)  $TS(T) \geq RTS(N)$  but  $TS(T) < WTS(N)$ 
    - i) If  $SCO(N) \geq STH(T)$   
Cancel the operation and transaction continues.
    - ii) If  $SCO(N) < STH(T)$ , wait until  $SCO(N) \geq STH(T)$  or occurrence of other case.
  - c)  $TS(T) < RTS(N)$   
Rollback transaction.

As all accesses start from root and proceed in a top-down fashion, we can easily derive from the access rules that the most updated version of a node  $N$  always have the greatest read and write timestamps among the subtree rooted by this node. To access a lower level node, the request must fulfill all the requirements to all nodes along the path to avoid inconsistency. For instance, it is impossible to access a version of node that has been written by a future transaction.

## 4. Evaluation

We implement the transaction management system for NextDB with two main components: *Transaction Manager (TM)* and *Resource Manager (RM)*. *TM* makes assuring to execute all transactions correctly in order while *RM* manages recently uploaded data for preventing direct accesses to the database. Under *TM*, we implement XStamps protocol, Tree protocol, and Two-phase locking protocol (2PL) to compare their performances. To make the evaluation efficient, we develop a transaction generator to generate various types of operations and transactions. The high level system architecture of transaction management system is shown in Fig. 2.

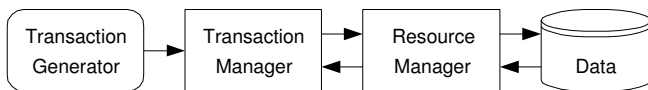


Figure 2: The Architecture of Transaction Management System.

### 4.1 Experimental Model

The system was developed using J2SDK 1.4.1 and evaluated on MS Windows 2000 Professional operating system which is running over IBM compatible PC with 2.4 GHz CPU and 256MB memory. The data set *Shakespeare's Plays*<sup>1</sup> which contains 327K elements in 37 XML documents (total 7.6MB in size) is used in the experiment. Various types of read and write transactions to the data are arbitrarily generated based on following parameters:

- *Number of transactions* - To control the maximum number of transactions that will be executed simultaneously.
- *Average number of operations* - To specify number of operations, in average, per transaction. Read transactions have 50% more chances to have more number of operations than average.
- *Percentage of write transactions* - To specify approximate percentage of write transactions in the system.
- *Simulated maximum I/O time* - To simulate the time each operation takes to be executed.

<sup>1</sup><http://www.ibiblio.org/bosak/xml/eg/>

## 4.2 Experimental Results

Fig. 3 shows the throughput comparisons among XStamps, Tree and Two-phase locking protocols based on evaluation. We set simulated I/O time to 10ms in all graphs. The Write% curve indicates the percentage of write operations (ranging from 0 to 1) in a transaction. Since the result of Tree protocol is almost identical to 2PL, we hide its curve.

The figure indicates that three protocols perform differently at different contention levels. The 2PL and Tree protocols outperform XStamps at very high contention levels, while XStamps reveals definite gain when the degree of contention is relatively low. In addition, XStamps demonstrates its strength in the situation when the average number of operations between 50% and 60% of transactions are write operations regardless of the number of transactions in the system. The phenomenon is due to higher rollback rate in a high contention environment; XStamps allows higher concurrency in all situations.

In a lower contention situation, at some point, all protocols yield unexpected low throughput as the set of transactions generated contains more DELETE operations which perform over very high level of hierarchy. This may cause later transactions to become invalid and thus aborted.

## 5. Related Work

The consistency issues in hierarchical database systems, and Tree protocols were introduced in [7] since 1980. Based on that, many variants were introduced; some of them includes dynamic tree-locking protocol [3] and dynamic directed acyclic graph policy [2] which is a generalized and extended protocol of [13]. With the emergence of XML, XML-specific lock-based protocols such as strict two-phase locking based solution [4], locking on directed acyclic graphs with DGLOCK [8] and XPath-based locking approach [5] are introduced in recent years. However, there has been relatively little attention to concurrency control policy based on timestamps. Although the extension of timestamps and multiversion timestamps protocols to hierarchical databases are introduced in [1], the extension does not quite fit to XML scenario. XStamps considers the operation semantics in XML and offers a more flexible concurrency control mechanism. Our experimental results have shown substantial improvements in the level of concurrency.

## 6. Conclusion

This paper proposes a new protocol for XML data, named XStamps, that is designed based on traditional multiversion timestamps concurrency control protocol. XStamps classifies the transactions into read and write categories, and creates read and write versions of data implicitly. XStamps also distinguishes the write operation to the database and handles them with different methods. Additional parameters, *SCO* and *STH*, are introduced in XStamps to enable flexible control of the isolation level of transactions and allow transactions to commit early depending on requirements.

The evaluation results show that XStamps works well with XML data and provides performance gain over Tree and 2PL protocols in a low conflict environment. In future, we plan to do more rigorous testing and looking more fine-grained concurrency control protocols for XML data. In addition, we will improve XStamps by introducing an efficient algorithm for generating dynamic *SCO* values for transactions, enhancing the ability of data version creation and maintenance, and integrating with recovery features to be an efficient transaction and recovery system for NextDB.

## References

- [1] M. Carrey. Granularity Hierarchies in Concurrency Control. *Journal of Association for Computing Machinery*, 83(3):156–165, 1983.
- [2] V. Chaudhri. *Transaction Synchronization in Knowledge Bases: Concepts, Realization and Quantitative Evaluation*. Phd thesis, University of Toronto, 1995.
- [3] A. Croker and D. Maier. A Dynamic Tree-Locking Protocol. In *Proc. Int. Conf. on Data Engineering*, pages 49–56, Los Angeles, USA, Feb. 1986.
- [4] S. Helmer, C. Kanne, and G. Moerkotte. Anatomy of a Native XML Base Management System. Technical report, University of Mannheim, 2001.
- [5] C. E. Hye and K. Tatsunori. XPath-based Concurrency Control for XML Data. In *Proc. on 14th Data Engineering Workshop*, Japan, 2003.
- [6] D. P. Reed. Naming and Synchronization in a Decentralized Computer System, Sept. 1978.
- [7] A. Silberschatz and Z. Kedem. Consistency in Hierarchical Database Systems. *Journal of Association for Computing Machinery*, 27(1):72–80, Jan. 1980.
- [8] G. Torsten, B. Klemens, and S. Hans-Jorg. XMLTM: Efficient Transaction Management for XML Documents. In *Proc. 11th Int. Conf. on Information and Knowledge Management*, VA, USA, 2002.
- [9] K. M. Win, W. K. Ng, and E. P. Lim. An Architectural Framework for Native XML Data Management. In *Proc. Int. Conf. on Cyberworlds*, Singapore, Dec. 2003.
- [10] K. M. Win, W. K. Ng, and E. P. Lim. ENAXS: Efficient Native XML Storage System. In *LNCS 2642: Proc. 5th APWeb Conference, X'ian, China*, pages 59–70, Springer Verlag, Apr. 2003.
- [11] K. M. Win, W. K. Ng, and E. P. Lim. NextDB: A Native Database Management System for XML Data (submitted for publication). 2003.
- [12] K. M. Win, E. Rosasillfiani, W. K. Ng, and E. P. Lim. A Visual Interface for Native XML Database. In *DEXA Workshop*, Prague, Czech Republic, Sept. 2003.
- [13] M. Yannakakis. A Theory of Safe Locking Policies in Database Systems. *Journal of Association for Computing Machinery*, 29(3):718–740, 1982.

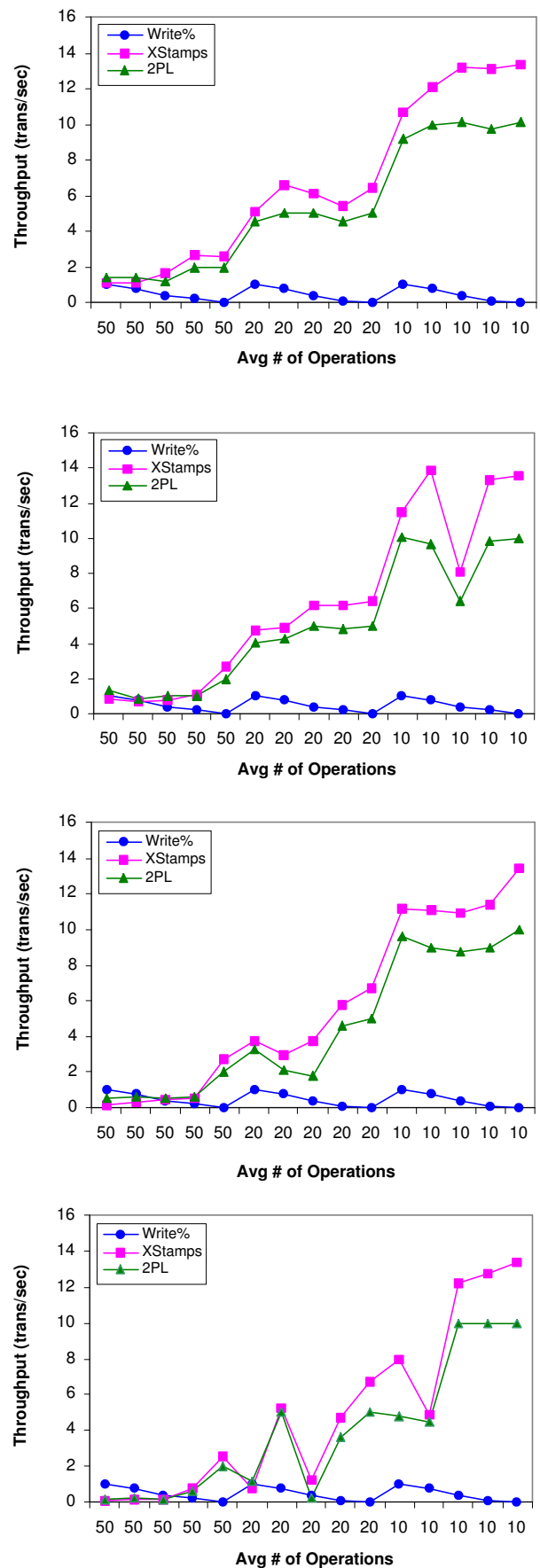


Figure 3: Throughput Comparison for 10, 50, 100 and 200 Transactions.