AN EFFICIENT LOCKING MODEL FOR CONCURRENCY CONTROL IN OODBS

G Arumugam[1*] and M Thangaraj[2]
[*1]*Department of Computer Science, Madurai Kamaraj University, Madurai 625021 TN India*
Email : gurusamyarumugam@yahoo.co.in
[2]*Department of Computer Science, Madurai Kamaraj University, Madurai 6250 21 TN India*
Email : thangarajmku@yahoo.com

ABSTRACT

When several transactions execute concurrently in a database, the isolation property may no longer be preserved. It is necessary for the system to control the interaction among the concurrent transactions. This paper presents a new locking model for concurrency control in object oriented database systems. This model is motivated by a desire to provide high concurrency and low locking overheads in accessing objects. The proposed model consists of a rich set of lock modes, hash table, lock-based signatures and B+ trees. The performance study result shows that the proposed model performs well for all possible operations on objects.

Key words: Concurrency, lock modes, hash table, B+ tree, signatures, object

1 INTRODUCTION

Recently, there has been increased interest in object oriented database systems, which appear to be driven mainly by the demands of data intensive applications such as CAD/CAM, office information systems and software development environments. With its flexible data model and object-oriented programming paradigm, it is believed that object oriented databases have great potential to be applied widely.

A transaction is a collection of read or update processes, each accessing a data item in a database. The read processes do not change the contents of the database but the update processes would modify the contents of the database. The concurrency control system should control the simultaneous execution of reads and updates, preserving the consistency so that the computer resources are used as efficiently as possible. The problems of concurrency control in conventional databases such as lost updates and uncommitted dependency (Datta & Son, 2002) remain unresolved in object oriented databases. Furthermore, due to the complexity of the object oriented data model, the problems become more complicated. One of the main techniques used to control concurrency is based on the concept of locks. These locks are categorized as S (shared lock), X (exclusive lock), IS (intention-shared lock), IX (intention-exclusive lock), and SIX (shared and intention-exclusive lock). Various locking models (Lee & Liou, 1996) have been developed to manage concurrency issues and their details are explained in Section 2.

Since different transactions obviously have different characteristics and requirements, it is desirable that the DBMS should provide a range of locking granularities (Pollari, Soisalon-Soininen & Ylonen, 1996). The term 'granularity' refers to the size of the object that can be locked. The objective of this paper is to develop a feasible locking model for an object-oriented database system (Norvag, Sandsta & Bratbergsengen, 1997), which overcomes the shortcomings of other locking models while retaining their advantages. The proposed model uses a new structure against B+ trees (Bertino, 1994) which are mostly used as an indexing structure to make efficient querying on objects.

The remainder of the paper is organized as follows: Section 2 is devoted to issues relevant to concurrency control. In Section 3, we describe the architecture for concurrency control. Section 4 shows the results of evaluating the performance of the algorithm. Finally, in Section 5 we present the conclusions.

2 RELATED WORK

Database systems frequently use indexes to access data. These systems typically operate at a high level of concurrency and, as the transactions have a high probability of accessing an index, it is necessary to ensure that concurrent access to indexes is not a bottleneck in the system. B+ trees are the most common dynamic index structures (Tao & Papadias, 2002) in database systems. The following algorithms have been developed to take care of the concurrent access of an object in object oriented databases.

Bayer-Schkolnick Algorithms (Srinivasan & Carey, 1993): All operations that get an X lock on the root, lock-couple their way to the leaf using X locks and then release the locks at higher levels whenever a safe node is encountered. This strategy ensures that when an update operation reaches a leaf, it holds X locks on all pages in its scope and there are no locks on any other index nodes. Updates and reads whose scopes do not interfere can thus execute concurrently. However, a considerable number of conflicts such as the non-availability of a node and deadlocks may be caused at higher-level nodes due to the use of X locks.

Locking model in ORION (Thangaraj, Kuppuswami, Prasanna & Paul, 1999): The locking model in ORION is based on Gray's hierarchy locking model. Two level hierarchies are used to model the lockable granules in Object Oriented databases. A coarse granule is at the class level and a fine granule is at the instance level. The instance objects in an ORION system are locked only in S or X mode. Class objects can be locked in S, X, IS, IX, or SIX modes.

B+ tree concurrency control algorithm (Lee & Liou, 1996): This method does not use all the lock modes but only the IS, IX, SIX and X locks. The first three locking modes are used in a hierarchical locking protocol. The X lock mode is the traditional lock mode. This algorithm can be improved by considering every index page as an independently lockable item instead of locking the entire tree.

3 THE PROPOSED MODEL

The model named as Arumugam-Thangaraj Concurrency Model (ATCM), is an integrated model supporting both inheritance and aggregation hierarchies The ATCM model (in Figure 1) is an improved version of the model proposed by Thangaraj et al, (1999) and Thangaraj & Sujatha (2001). The additional features of the data structure of the ATCM model bring efficiency when compared with B+ tree models. It consists of two indexes, namely primary and auxiliary indexes. Both indexes use the data structures with a combination of signatures, hash tables and B+trees.

The primary index is organized on the key to be indexed and it consists of a hash table, signature and B+trees. Each bucket in the hash table has a signature pattern (Tao & papadias, 2002) and a pointer to a B+ tree. The signature is a bit pattern that has a collection of *n* bits, which indicates the type of lock on a particular object. The columns in the pattern matrix are used to represent S, X, IS, IX, SIX locks and availability of the object. A bit corresponding to a class in the bit map is set if and only if there is at least one object that belongs to that class having the key value K. During access, the objects availability and types of lock on the object can be determined easily. Any effective hashing function can be used to hash the given key into the hash table and hence into the corresponding B+ tree.

The leaf node in the primary B+tree has the format shown in Figure 1. It consists of the classid, record length, key value, class directory and signature having instances with the key value in the indexed attribute, their lock details and the offset where the object identifiers (OIDs) of the classes are stored. The remaining part of the node consists of the number of OIDs in each class and the OID along with the pointer to the corresponding object in the auxiliary index.

An auxiliary index is maintained for all objects except for those that belong to the root class and its subclasses. The auxiliary index also consists of a hash table, signature patterns and B+ trees. Every entry in the hash table has a signature pattern that indicates the availability of the object and a pointer to a B+ tree. Any effective hashing

technique can be used to hash the OID of the object to the hash table and hence to the corresponding B+ tree. The format of a leaf node in the auxiliary B+ tree is shown in Figure 1. It consists of the OID of the object, the record length (the number of parent OIDs), a pointer to the primary record and list of its parents.

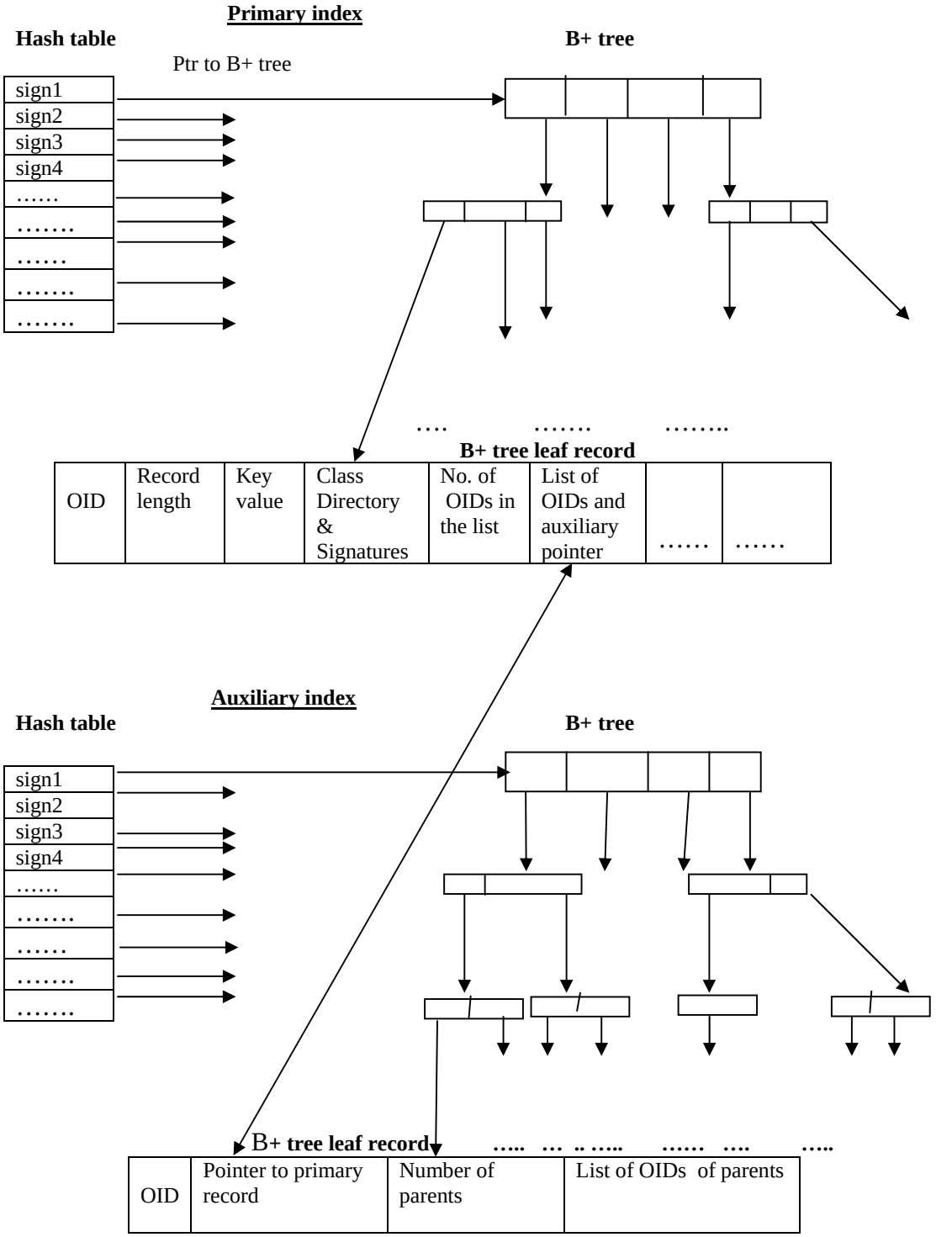

Figure 1. ATCM architecture.

3.1 Operations

The proposed model facilitates operations like reading an object, writing an object and updating the contents of an object. The following algorithms describe how each operation is carried out:

a)

```
Algorithm              : Read
Data structure         : See Figure 1
Input                  : search value K
Output                 : object

Steps :
        Hash the key K, find the relevant bucket in primary hash table
        If the K is in Bit matrix  and  S bit =1 and not IX bit =1
      {
        func find ( nodepointer, search value K)
        {
        if *nodepointer is a leaf
          Check the signature for the type of lock on the class
          Return the object.
        else
                if K < K1 then return find(P0, K)

                else

                        if  K >= Km  then return find ( Pm, K)
                                //m = number of entries
                        else
                                Find i such that Ki <= K <= K i+1

                                Return find (Pi, K)
        }
        find the class directory in the  primary record node
        check for the signature
        retrieve  the object.
       }
```

b)

```
Algorithm              : Update
Data structure         : See Figure 1
Input                  : Key  K, V new value
Output                 : updated object

Steps :
        Hash the key K, find the relevant bucket in auxiliary hash table
        If the K is in Bit matrix and X = 0
        {
           set the corresponding bit in the matrix
           func modify (value V, pointer P)
            {
               find(node pointer, search value K)
              find the leaf node L that contains value K
              find the primary record using the pointer
              using class directory  access the OID and  modify with V
            }
        }
```

In the case of B+ tree model the tree is searched to find a particular object but in the ATCM case, the signature pattern helps the user to identify the availability of an object and the type of lock on that object.

3.2 Locking request scheduling

The set of all requests for a particular object is kept in a queue and sorted by a type of fair scheduler. A fair scheduler must guarantee that no particular transaction is blocked indefinitely. Most of the granted requests are

placed in a queue called the granted buffer. Even though the First-Come-First-Served technique is simple and fair for scheduling requests, it fails to maximize the degree of concurrency. So, a dual-queue scheduling (Lee & Liou, 1996) scheme is used in our ATCM model.

4 PERFORMANCE ANALYSIS

This section provides experimental results of the performance of the ATCM model at accessing objects concurrently. The experiments were designed to answer the following questions:

- How good is the performance of the ATCM model compared with the B+ tree model ?
- What is the improvement achieved by this new model?

The ATCM model was implemented using C++ in Windows. The experiments were performed on Pentium machines using a data set of 10 million objects with a uniform distribution of search keys (ie. OID). The primary performance metric used was the *miss ratio (*fraction of transactions that miss their deadlines) and was calculated as

$$\textit{miss ratio} = \frac{\text{number of transactions missing deadline}}{\text{total number of transactions arriving into the system}}$$

It is found that the *miss ratio* is very minimal in our model because of the introduction of dual-queue scheduling.

In our experiments, three factors were varied: the workload (the percentage of searches and updates), the system (number of CPUs and disks) and the structure of the tree (the fan-out and the number of keys). These factors determine the data and resource conflict levels of the system. The parameters used in our experiments are given in Table 1.

Table 1 : Simulation parameters

Nc	Number of CPU
Nd	Number of disk
Dsk time	Min: 0 msec; Max:30 msec
Lock time	0.05
Fanout	Number of index entries per page
Mpl	Multiprogramming level

It is essential to study the performance of these models when executing requests for reading and updating an object. As the ATCM model distributes objects uniformly among the B+ trees in various buckets, carrying out concurrent request for various operations is effective. Accessing all the objects through a single root path is a major drawback of the B+ tree model. The following graphs show the outcome of the experiments when read and update operations are done on objects using the two models:

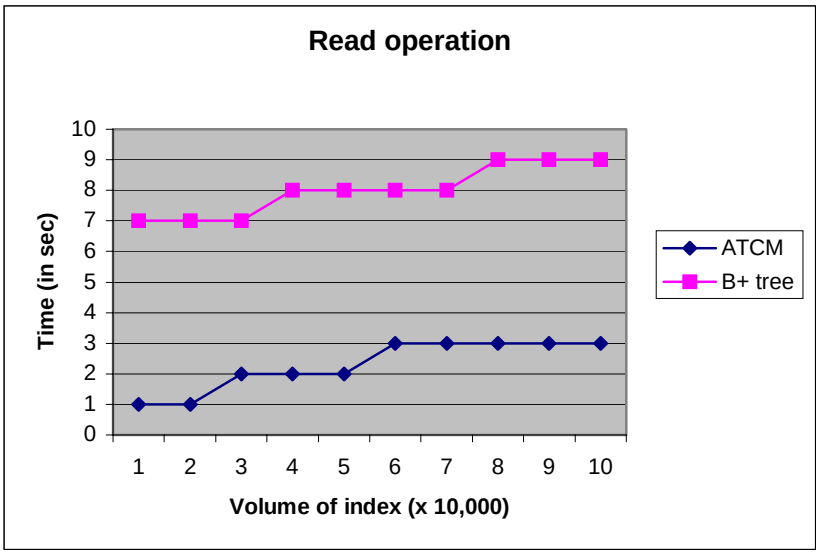

Figure 2 shows the read operation in the indexing structures

The graph in Figure 2 shows the performance of the two models when applying read operation on a particular object. The new model performs well compared to the B+ tree model.Figure 3 shows the performance of these two models on update operation.

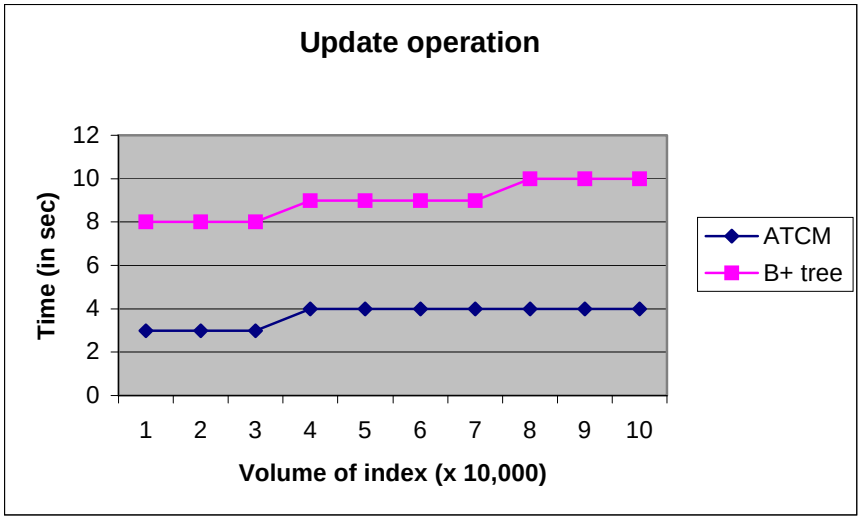

Figure 3 shows the update operation in the index structures

The performance of the two models were evaluated when multiple transactions concurrently access an object for a particular operation. In Figure 4, we present the result of the concurrent read operations by multiple transactions and it can be seen that ATCM model performs better than B+ tree model.

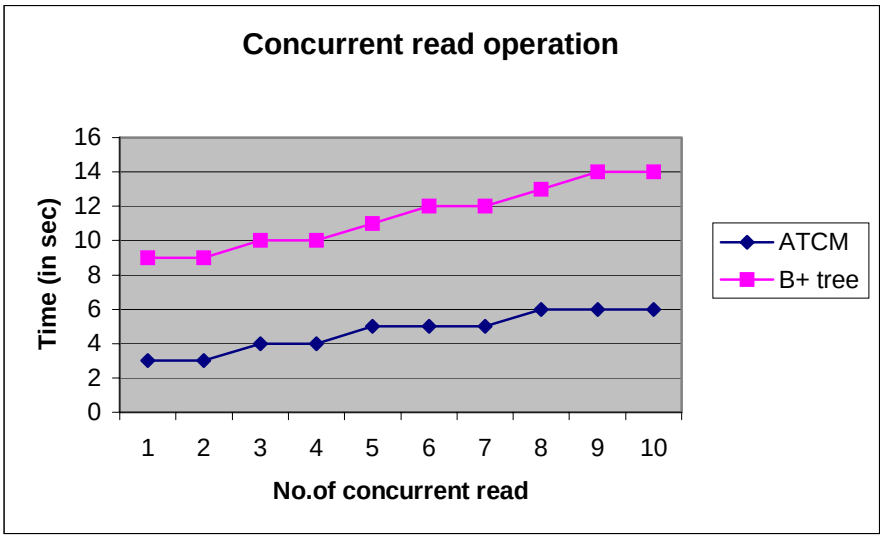

Figure 4 shows the concurrent read operation in the index structure

By supporting a full set of lock modes for classes, ATCM has a low overhead on locking classes. It also gains a higher degree of concurrency by applying both the techniques of hierarchical locking on composite objects and a dual queue-scheduling scheme. These results state that the proposed ATCM is quite an efficient locking model with a higher degree of concurrency.

5 CONCLUSION

In this paper, a new model is proposed to control the concurrent access of objects in an object database. ATCM model uses a signature pattern, hash table and B+ trees. This paper builds upon previous work on locking strategies in object databases. Due to some fundamental differences between B+ tree and ATCM model architectures, the algorithms developed for B+ tree model do not provide a feasible solution for concurrent access. Motivated by the limitations of the previous approach, we have developed a new granular locking approach suited for concurrency control in OODB. This model provides a high degree of concurrency and has a low lock overhead. Our experiments have shown that the proposed model works well under various system loads and significantly outperforms the B+ tree model for small to medium sized data sets. This work can further be extended towards range queries and concurrent access in distributed objects.

6 ACKNOWLEDGEMENTS

We would like to thank F J Smith, the editor-in-chief and the anonymous reviewers whose careful reading and constructive criticisms on the work helped to improve the clarity and content of this paper.

7 REFERENCES

Bertino, E. (1994) A survey of indexing techniques for OODB. *Proc.of Query processing for advanced database systems, Dagstuhl,* Morgan Kaufmann Publishers, Inc.

Datta, A., & Son, S.H. (2002) A study of concurrency control in Real-time, active Database systems. *IEEE Transaction on Knowledge and Data Engg. 14(3),* 465-484.

Lee, S.Y., & Liou, R.L. (1996) A multi-granularity locking model for Concurrency control in OODB. *IEEE Trans on Knowledge and Data Engg. 8(1),* 144-156.

Norvag, K., Sandsta, O., & Bratbergsengen, K. (1997) Concurrency control in Distributed object-oriented database systems. *Proc. of ADBIS,* St.Petersburg, Russia.

Pollari, K., Soisalon-Soininen, E., & Ylonen, T. (1996) Concurrency Control in B-Trees with Batch Updates. *IEEE Trans. on Knowledge and Data Engg. 8(6),* 975-984.

Srinivasan, V., & Carey, M.J. (1993) Performance of B+ tree concurrency control Algorithms. *VLDB Journal,* 2, 361-406.

Tao, Y., & Papadias, D. (2002) Adaptive index structures. *Proc.of VLDB Conference,* Hong Kong, China.

Thangaraj, M., Kuppuswami, S., Prasanna, V., & Paul, R. (1999) A new integrated indexing model for OODB. *Proc. Int. Conf. ROVIPIA,* Ipoh, perak, Malaysia.

Thangaraj, M., & Sujatha, G. (2001) An efficient locking model for concurrency control in object oriented data base systems. *Proc. Of NCHPC.* Pollachi, India.