ISSN:0975-9646

Advanced Indexing Techniques for Achieving Concurrency in Multidimensional Data Sets

N Krishna Kumari, Dr. M.H.M. Krishna Prasad
Department of Information Technology, University College of Engineering
JNTU Kakinada, Vizianagaram Campus, Vizianagaram, India

Abstract: In multidimensional datasets concurrent accesses to data via indexing structures introduce the problem protecting ranges specified in the retrieval from phantom insertions and deletions. This paper proposes a novel approach for concurrency in multidimensional datasets using Advanced Indexing Technique like generalized search tree, R tree and its variants, constitutes an efficient and sound concurrency access model for multidimensional databases and it supports efficient operations with serializable, isolation, consistency and deadlock free.

Index terms: Concurrency Control, Locking Management, Phantom Problem.

1. INTRODUCTION:

Over the last decade (1988-98), the R tree has emerged as one of the most robust multidimensional access methods. However, before the R tree can be integrated as an access method to a commercial strength database management system, efficient techniques to provide transactional access to data via R trees need to be developed. Concurrent access to data through a multidimensional data structure introduces the problem of protecting ranges specified in the retrieval from phantom insertions and deletions (the phantom problem). Existing approaches to phantom protection in B trees (namely, key range locking) cannot be applied to multidimensional data structures since they rely on a total order over the key space on which the B tree is designed. The paper presents a dynamic granular locking approach to phantom protection in R trees. To the best of our knowledge, the paper provides the first solution to the phantom problem in multidimensional access methods based on granular locking.

The phantom problem is defined as follows (from the ANSI/ISO SQL-specifications Transaction T1 reads a set of data items satisfying some <search condition>. Transaction T2 then creates data items that satisfy T1's <search condition> and commits. If T1 then repeats its scan with the same <search condition>, it gets a set of data items (known as "phantoms") different from the first read. Phantoms must be prevented to guarantee serializable execution. Object level locking *does not* prevent phantoms since even if all objects currently in the database that satisfy the search predicate are locked, concurrent insertions into the search range cannot be prevented.

1.1. Modes of lock: Let me go ahead in explaining you the various modes available with a typical table as above:

Modes of Lock	Comment
Shared	This is a typical mode of operation for Select statements where the resource can be shared by multiple users. Since you are just reading the data you can share the resource.
Exclusive	The next operation we can think of is to do an DML operation (Insert / Update / Delete). This lock ensures that two people do not do the modification on the same data at the same time.
Update	This lock mode is used to mark an object for the update operation.
Intent	This mode establishes an locking tree mechanism wherein it can include an intent shared, intent exclusive and share with exclusive locks. This mode does stress the point that the data can be updated at any time. For example, this happens when you take a cursor for update.
Schema	This lock is taken when we do an DDL operation where the integrity of the database schema needs to be verified. This can include from creating a table, procedure, alter a column width, adding an column etc.
Bulk Update	This is taken when we do an Bulk-Upload of data or Bulk Copying of data into the system.

1.2. Approaches to Phantom Protection:

There are two general strategies to solve the phantom problem, namely *predicate locking* and its engineering approximation, *granular locking*. In predicate locking, transactions acquire locks on predicates rather than individual objects. Although predicate locking is a complete solution to the phantom problem, the cost of setting and clearing predicate locks can be high since (1) the predicates can be complex and hence checking for predicate satisfiability can be costly and (2) even if predicate satisfiability can be checked in constant time, the complexity of acquiring a predicate lock is proportional in the number of concurrent transactions which is an order of magnitude costlier compared to acquiring object locks that can be set and released in constant time [9]. In contrast, in granular locking, the predicate space is divided into a set of lockable resource granules. Transactions acquire locks on granules instead of on predicates. The locking protocol guarantees that if two transactions request conflicting mode locks on predicates p and p0 such that p^p0 is satisfiable, then the two transactions will request conflicting locks on at least one granule in common. Granular locks can be set and released as efficiently as object locks. For this reasons, all existing commercial DBMSs use granular locking in preference to predicate locking.

An example of the granular locking approach is the *multigranularity locking protocol* (MGL) [12]. MGL exploits additional lock modes called *intention* mode locks which represent the intention to set locks at finer

granularity (see Table 1). Application of MGL to the key space associated with a B-tree is referred to as *key range locking* (KRL) [12, 13]. KRL cannot be applied for phantom protection in multidimensional data structures since it relies on the total order over the underlying objects based on their key values which do not exist for multidimensional data. Imposing an artificial total order (say a Z-order [14]) over multidimensional data to adapt KRL would result in a scheme with low concurrency and high lock overhead since protecting a multidimensional region query from phantom insertions and deletions will require accessing and locking objects which may not be in the region specified by the query (since an object will be accessed as long as it is within the upper and the lower bounds in the region according to the superimposed total order). It would severely limit the usefulness of the multidimensional AM, essentially reducing it to a 1-d AM with the dimension being the total order.

1.2. Desiderate of the Solution:

Since KRL cannot be used in multidimensional index structures, new techniques need to be devised to prevent phantoms in such data structures. The principal challenges in developing a solution based on granular locking are:

Defining a set of lockable resource granules over the multidimensional key space such that they (1) dynamically adapt to key distribution (2) fully cover the entire embedded space and (3) are fine enough to afford high concurrency. The importance of these factors in the choice of granules has been discussed in [9]. The lock granules (i.e. key ranges) in KRL satisfy these 3 criteria.

Easy mapping of a given predicate onto a set of granules that needs to be locked to scan the predicate. Subsequently, the granular locks can be set or cleared as efficiently as object locks using a standard lock manager (LM).

Handling overlap among granules **effectively**. This problem does not arise in KRL since the key ranges are always mutually disjoint. In multidimensional key space partitioning, the set of granules defined may be, in GiST terminology, "mutually consistent". For example, there may be spatial overlap among R-tree granules.

This complicates the locking protocol since a lock on a granule may not provide an "exclusive coverage" on the entire space covered by the granule. For correctness, the granular locking protocols must guarantee that any two conflicting operations will request conflicting locks on at least one granule in common. This implies that at least one of the conflicting operations must acquire locks on all granules that *overlap* with its predicate while the other must acquire conflicting locks on enough granules to fully *cover* its predicate [5]. This leads to two alternative strategies:

Overlap-for-Search and Cover-for-Insert Strategy (OSCI) in which the searchers acquire shared mode locks on all granules consistent with its search predicate whereas the inserters, deleters and updators acquire IX locks on a minimal set of granules sufficient to fully cover the object being inserted, deleted or updated.

Cover-for-Search and Overlap-for-Insert Strategy (CSOI) in which the searchers acquire shared mode locks on a minimal set of granules sufficient to fully cover its search predicate whereas the inserters, deleter's and updater's acquire IX locks on all granules consistent with the object being inserted, deleted or updated.

While the former strategy favors the insert and delete operations by requiring them to do minimal tree traversal and disfavors the search operation by requiring them to traverse all consistent paths, the latter strategy does exactly the reverse. Intermediate strategies are also possible. For GL/GiST, we choose the OSCI strategy in preference to the rest. The OSCI strategy effectively does not impose *any* additional overhead on any operation as far as tree traversal is concerned since searchers in GiST anyway follow all consistent paths. The CSOI strategy may be better for index structures where inserters follow all overlapping paths and searchers follow only enough paths to cover its predicate.

The R+-tree is an example of such an index structure [15]. We assume that the OSCI strategy is followed for all 1In this paper, we use the term "granules" to mean lock units – resources that are locked to insure isolation and not in the sense of granules in "granule graph" of MGL [9]. This is discussed in further detail in Section 4.1. discussions in the rest of the paper.

Preventing Phantoms

- Table locking prevents phantoms; row locking does not *Predicate locking* prevents phantoms
- A predicate describes a set of rows, some are in a table and some are not
- Every SQL statement has an associated predicate
- When executing a statement, acquire a (read or write) lock on the associated predicate
- Two predicate locks conflict if one is a write and there exists a row (not necessarily in the table) that is contained in both.

Terminology

In developing the algorithms, we assume, as in [12], that a transaction may request the following types of operations on GiST: Search, Insert, Delete, Read Single, Update Single and Update Scan. In presenting the solution to the phantom problem, we describe the lock requirements of each of these and present the algorithms used to acquire the necessary locks. The lock protocols assumes the presence of a standard LM which supports all the MGL locks modes (as shown in Table 1) as well as conditional and unconditional lock options [16]. Furthermore, locks can be held for different durations, namely, instant, short and commit durations [16]. While describing the lock requirements of various operations for phantom protection, we assume the presence of some protocol for preserving the physical consistency of the tree structure in presence of concurrent operations. The lock protocol presented in this paper guarantees phantom protection independent of the specific algorithm used to preserve tree consistency. In our implementation, we have combined the GL/GiST protocol with the latching protocol proposed in [10]. We do not describe the combined algorithms in this paper due to space limitations but can be found in the longer version of this paper [5].

2 RELATED RESEARCH AND MOTIVATION

In this section, we review the structure of the R-tree family, discuss some limitations that affect R+-trees, survey major concurrency control algorithms based on B-trees and R-trees, and summarize the challenges inherent in applying concurrency control to R+-trees.

2.1 R-Tree Index Structure:

An R-tree [2] is a height-balanced tree similar to a B-tree with index records in its leaf nodes containing pointers to data objects Nodes correspond to disk pages If the index is disk resident, and the structure is designed so that a spatial search requires visiting only a small number of nodes The index is completely dynamic; inserts and deletes can be inter- mixed with searches and no periodic reorganization is required.

2.2. R-Plus Tree:

We move now to formally describe the structure of R+ Tree [3]. A leaf node is of the form (oil, RECT) where oil is an object identifier and is used to refer to an object in the database. RECT is used to describe the bounds of data objects. For example, in a 2-dimensional space, an entry RECT will be of the form (Xlow,Xhigh,YIow,Yhigh) which represents the coordinates of the lower-left and upper-right corner of the rectangle. An inter- mediate node is of the form where p is a pointer to a lower level node of the tree and RECT is a representation of the rectangle that encloses.

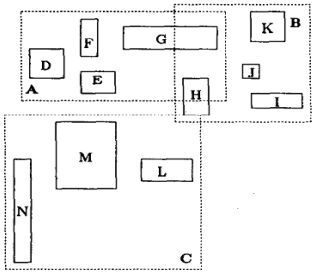


Fig 2.a Rectangles organized on to R Tree

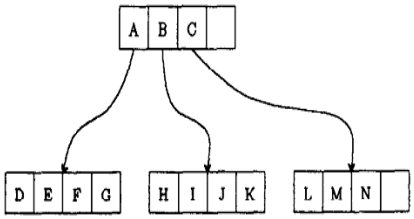


Fig 2.b. R Tree for Rectangles of Fig 2.a.

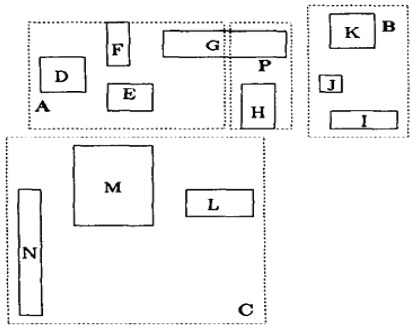


Fig 2.c Rectangles organized on to R Plus Tree

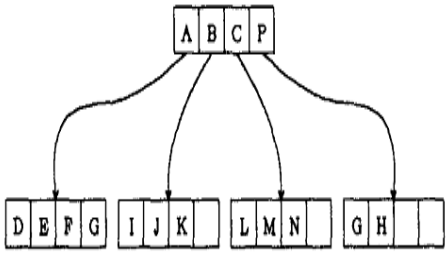


Fig 2.d. R Plus Tree for Rectangles of Fig 2.c.

2.3 Concurrency Controls:

Several concurrency control algorithms have been proposed to support concurrent operations on multidimensional index structures, and they can be categorized into lock-coupling-based and link-based algorithms. The lock coupling- based algorithms [20], [21] release the lock on the current node only when the next node to be visited has been locked while processing search operations. During node splitting and MBR updating, these approaches must hold multiple locks on several nodes simultaneously, which may deteriorate the system throughput.

The link-based algorithms [29], [30], [31], [32], [26] were proposed to reduce the number of locks required by lock coupling- based algorithms. These methods lock one node most of the time during search operations, only employing lock coupling when splitting a node or propagating MBR changes. The link-based approach requires all nodes at the same level be linked together with right or bidirectional links. This method reaches high concurrency by using only one lock simultaneously for most operations on the B-tree. The link-based approach cannot be used directly in multidimensional data access methods as there is no linear ordering for multidimensional objects. To overcome this problem, a right-link style algorithm (R-link tree) [30] has been proposed to provide high concurrency control by assigning logical sequence numbers (LSNs) on R-trees. However, when a node splitting propagates and its MBR updates, this algorithm still applies lock coupling. Also, in this algorithm, additional storage is required to retain extra information for the LSNs of associated child nodes. To solve this extra storage problem, Concurrency on Generalized Search Tree (CGiST) [31] applies a global sequence number, the Node Sequence Number (NSN). The counter for NSN is incremented for each node split, with the original node receiving the new value and the new sibling node inheriting the previous NSN and its right-link pointer. In order for the algorithm to work correctly, multiple locks on two or more levels must be held by a single insert operation, which increases the blocking time for search operations.

Several mechanisms, such as top-down index region modification (TDIM), copy-based concurrent update (CCU), CCU with non blocking queries (CCUNQ) [29], and partial lock coupling (PLC) [26], have been proposed to improve the concurrency based on the above linking techniques. However, the link-based approach with these improvements is still not sufficient to provide phantom update protection. Phantom updating refers to updates that occur before the commitment, in the range of a search (or a following update), and are not reflected in the results of that search (or the following update). Concurrent data access through multidimensional indexes introduces the problem of protecting a query range from phantom updates. The dynamic granular locking approach (DGL) has been proposed to provide phantom update protection in the R-tree [5] and GiST [5].The DGL method dynamically partitions an embedded space into lockable granules that adapt to the distribution of objects. The leaf nodes and external granules of internal nodes are defined as lockable granules. External granules are additional structures that partition the non covered space in each internal node to provide protection. According to the principles of granular locking, each operation requests locks on sufficient

granules such that any two conflicting operations will request conflicting locks on at least one common granule. Although the DGL approach provides phantom update protection for multidimensional access methods and granular locks can be efficiently implemented, the complexity of DGL may impact the degree of concurrency.

2.4 Isolation Levels in Data Base Engine:

Transactions specify an isolation level that defines the degree to which one transaction must be isolated from resource or data modifications made by other transactions. Isolation levels are described in terms of which concurrency side-effects, such as dirty reads or phantom reads, are allowed.

Transaction isolation levels control:

- Whether locks are taken when data is read, and what type of locks are requested.
- How long the read locks are held.
- Whether a read operation referencing rows modified by another transaction:
- Blocks until the exclusive lock on the row is freed.
- Retrieves the committed version of the row that existed at the time the statement or transaction started.
- Reads the uncommitted data modification.

Choosing a transaction isolation level does not affect the locks acquired to protect data modifications. A transaction always gets an exclusive lock on any data it modifies, and holds that lock until the transaction completes, regardless of the isolation level set for that transaction. For read operations, transaction isolation levels primarily define the level of protection from the effects of modifications made by other transactions.

A lower isolation level increases the ability of many users to access data at the same time, but increases the number of concurrency effects (such as dirty reads or lost updates) users might encounter. Conversely, a higher isolation level reduces the types of concurrency effects that users may encounter, but requires more system resources and increases the chances that one transaction will block another. Choosing the appropriate isolation level depends on balancing the data integrity requirements of the application against the overhead of each isolation level. The highest isolation level, serializable, guarantees that a transaction will retrieve exactly the same data every time it repeats a read operation, but it does this by performing a level of locking that is likely to impact other users in multi-user systems. The lowest isolation level, read uncommitted, may retrieve data that has been modified but not committed by other transactions. All of the concurrency side effects can happen in read uncommitted, but there is no read locking or versioning, so overhead is minimized.

Database Engine Isolation Levels:

The ISO standard defines the following isolation levels, all of which are supported by the SQL Server Database Engine:

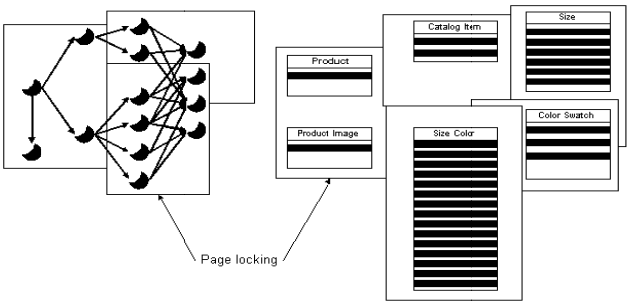
- Read uncommitted (the lowest level where transactions are isolated only enough to ensure that physically corrupt data is not read)
- Read committed (Database Engine default level)
- Repeatable read
- Serializable (the highest level, where transactions are completely isolated from one another).

The amount of data that can be locked with the single instance or groups of instances defines the granularity of the lock. The types of granularity are illustrated here are:

- Page locking
- Cluster locking
- Class or table locking
- Object or instance locking

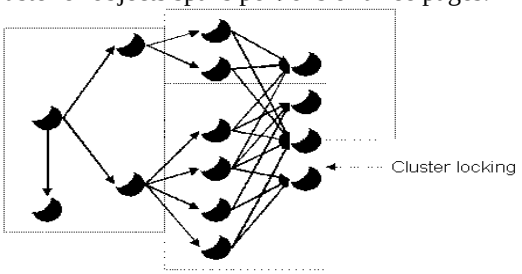
Page locking

Page locking (or page-level locking) concurrency control is shown in the figure below. In this situation, all the data on a specific page are locked. A page is a common unit of storage in computer systems and is used by all types of DBMSs. In this figure, each rectangle represents a page. Locking for objects is on the left and page locking for relational tuple's is on the right. If the concept of pages is new to you, just think of a page as a unit of space on the disk where multiple data instances are stored.



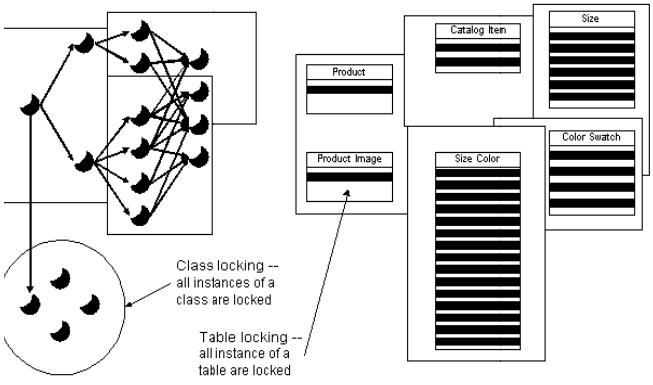
Cluster locking

Cluster locking or container locking for concurrency control is illustrated in the figure below. In this form of locking, all data clustered together (on a page or multiple pages) will be locked simultaneously. This applies only to clusters of objects in ODBMSs. Note that in this example, the cluster of objects spans portions of three pages.



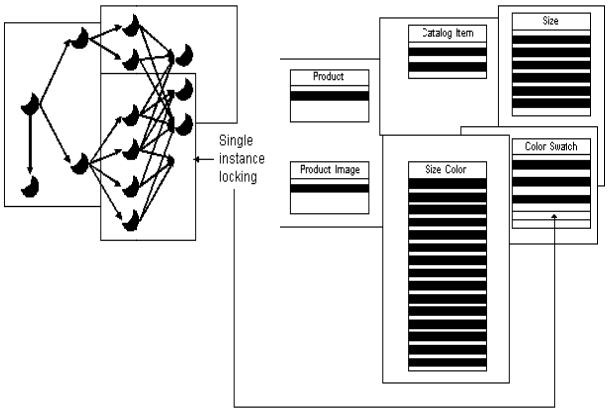
Class or table locking

Class or table locking means that all instances of either a class or table are locked, as is illustrated below. This shows one form of concurrency control. Note the circle at the lower left. It represents all instances of a class, regardless of the page where they are stored.



Object or instance locking

Instance locking locks a single relational tuple in an RDBMS or a single object in an ODBMS. This type of concurrency control is illustrated below.



3. EXPERIMENTAL SETUP:

The experiments [1] were conducted on a Pentium 4 desktop with 512 Mbytes memory, running a Java2 platform under windows XP. The implementations of the R-tree, the R+-tree, and the ZR+-tree were all based on the Java source package for R-tree obtained from the R-tree portal [26]. The first set of experiments evaluated the construction and query performance of the ZR+-tree. In these experiments, different data sizes were selected to construct the ZR+-trees, R-trees, and R+-trees. In evaluating the query performance, I/O cost is the determining factor, because the query process on the ZR+-tree does not introduce extra computation compared to the R+-tree. The disk accesses of the point queries were recorded by varying the number of rectangles. Additionally, the standard deviations of the number of disk accesses were calculated to compare the stability of the ZR+-tree and the R+-tree. Consequently, queries with different window sizes were executed on the constructed trees in order to record the execution cost. From the analysis of the algorithm given in the previous section, both the point query and window query performances of ZR+-trees are expected to be better than those of the R-trees. The number of disk accesses in this set of experiments was computed to be the average value for 1,000 random queries in order to reduce the impact of uneven data distribution.

Throughput of Concurrency Control

The performance for concurrent query execution was evaluated both for the R-tree with granular locking and the ZR+-tree with the proposed GLIP protocol. In order to compare these two multidimensional access frameworks, two parameters, namely, concurrency level and write probability were applied to simulate different application environments on the three data sets. Here, concurrency level is defined as the number of queries to be executed simultaneously, and write probability describes how many queries in the whole simultaneous query set are update queries. The execution time measured in milliseconds was used to represent the throughput of each of the approaches. According to the algorithm analysis in the previous section, the ZR+-tree with concurrency control should perform better than the R-tree with granular locking when the write probability is low. This performance gain comes from not only the outstanding query performance of the ZR+-tree but also the finer granules of the leaf nodes in the ZR+-tree. The size of the queries executed was tunable in this set of experiments. The data sets used in these experiments were the same as those used in the query performance experiments, except that the size of the synthetic data set was reduced to 5,000 in order to assess the throughput in relatively small data sets compared to the real data sets.

Fig 3.0 and Fig 3.1 shows the execution time costs for the three data sets with a fixed concurrency level and changing write probabilities when the query range is 1 percent of the data space. The concurrency level was fixed at two levels 30 and 50 as representative levels, while the write probability varied from 5 percent to 40 percent. The y-axis in these figures shows the time taken to finish these concurrent operations, and the x-axis indicates the portions of update operations in all the concurrent operations in terms of percentages. Both approaches degrade the throughput when the write probability increases. Comparing the performance from the different write probabilities, GLIP on the ZR+-tree performs better than granular locking on the R-tree when the write probability is small. When the write probability increases, the throughput of the concurrency control on the R-tree comes close to and exceeds that of the ZR+-tree. Specifically, when the concurrency level is 30, the throughput of the ZR+- tree is better with a write probability lower than 30 percent in real data sets. When the concurrency level is raised to 50, the concurrency control on the ZR+-tree outperforms the R-tree in cases where the write probability is less than 35 percent. From this set of figures, it can be concluded that in reading predominant environments, GLIP on the ZR+-tree provided better throughput than dynamic granular locking on the Rtree, although this advantage tended to decrease as the write probability increased.

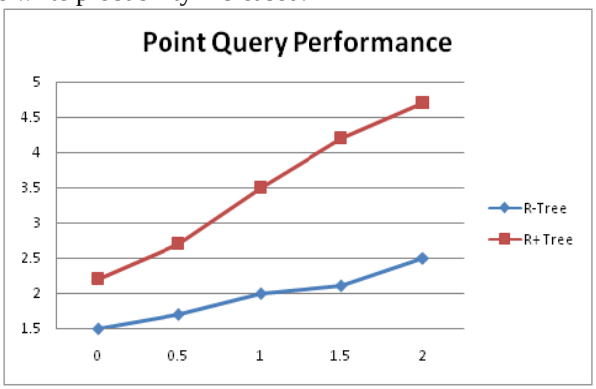


Fig. 3.0. Point Query Performance of R-tree, R+-tree,

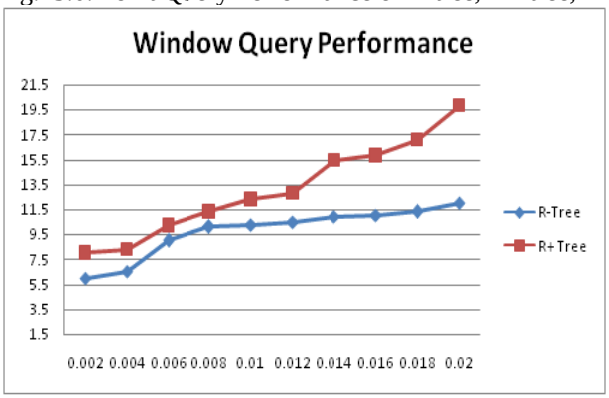


Fig. 3.1. Window Query Performance of R-tree, R+-tree, on Synthetic Data

CONCLUSION

This paper proposes a new concurrency control protocol, GLIP, with an improved spatial indexing approach, the ZR+-tree. GLIP is the first concurrency control mechanism designed specifically for the R+-tree and its variants. It assures serializable isolation, consistency, and deadlock free for indexing trees with object clipping. The ZR+-tree segments the objects to ensure every fragment is fully covered by a leaf node. This clipping-object design provides a better indexing structure. Furthermore, several structural limitations of the R+-tree are overcome in the ZR+-tree by the use of a non-overlap clipping and a clustering-based reinsert procedure. Experiments on tree construction, query, and concurrent execution were conducted on both real and synthetic data sets, and the results validated the soundness and comprehensive nature of the new design. In particular, the GLIP and the ZR+-tree excel at range queries in search-dominant applications. Extending GLIP and the ZR+-tree to perform spatial joins, KNN-queries, and range aggregation offer further attractive possibilities.

REFERENCES:

[1].GLIP: A Concurrency Control Protocol for Clipping Indexing by Chang-Tien Lu, Member, IEEE , JingDai, Student Member, IEEE, Ying Jin, and Janak Mathuria, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 21, NO. 5, MAY 2009

[2]R-TREES. A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING Antomn Guttman University of Cahforma Berkeley

[3]THE R+-TREE: A DYNAMIC INDEX FOR MULTI-DIMENSIONAL OBJECTS Timos Sellis, Nick Roussopoulosand Christos Faloutsos2 Department of Computer Science University of Maryland College Park, MD 20742Conf. Principles of Database Systems (PODS ’87), pp. 159-169, 1987.

[4]K. Chakrabarti and S. Mehrotra, “Dynamic Granular Locking Approach to Phantom Protection in R-Trees,” Proc. 14th IEEE Int’l Conf. Data Eng. (ICDE ’98), pp. 446-454, 1998.

[5]K. Chakrabarti and S. Mehrotra, “Efficient Concurrency Control in Multi-Dimensional Access Methods,” Proc. ACM SIGMOD ’99, pp. 25-36, 1999.

[6] J.K. Chen, Y.F. Huang, and Y.H. Chin, “A Study of Concurrent Operations on R-Trees,” Information Sciences, vol. 98, nos. 1-4, pp. 263-300, May 1997.

[7] V. Gaede and O. Gunther, “Multidimensional Access Methods,” ACM Computing Surveys, vol. 30, no. 2, pp. 170-231, June 1998.

[8] D. Greene, “An Implementation and Performance Analysis of Spatial Data Access Methods,” Proc. Fifth IEEE Int’l Conf. Data Eng. (ICDE ’89), pp. 606-615, 1989.

[9] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees in database systems. In Proceeding of VLDB, pages 562573, September 1995.

[10] M. Kornacker, C. Mohan, and J. Hellerstein. Concurrency and recovery in generalized search trees. In Proc. of SIGMOD, 1997.

[11]D. Lomet. Key range locking strategies for improved concurrency. In VLDB Proceedings, August 1993.

[12].J. Melton and A. R. Simon. Understanding the new sql: A complete guide. Morgan Kauffman, 1993.

[13].C. Mohan. ARIES/KVL: A key value locking method for concurrency control of multiaction transactions operating on btree indexes. In *Proceeding of VLDB*, August 1990.

[14]A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM TODS, Vol. 17, No. 1:94–162, March 1992.[15] B. Nichols, D. Buttlar, and J. P. Farrell. Pthreads Programming. O’Reilly & Associates, 1996

[15] M. Kornacker, C. Mohan, and J. Hellerstein, “Concurrency and Recovery in Generalized Search Trees,” Proc. ACM SIGMOD ’97, pp. 62-72, 1997.

[16] P. Lehman and S. Yao, “Efficient Locking for Concurrent Operations on B-trees,” ACM Trans. Database Systems, vol. 6, no. 4, pp. 650-670, Dec. 1981.

[17] D. Lomet, “Key Range Locking Strategies for Improved Concurrency,” Proc. 19th Int’l Conf. Very Large Data Bases (VLDB ’93), pp. 655-664, 1993.

[18] C. Mohan and F. Levin, “ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging,” Proc. ACM SIGMOD ’92, pp. 371-380, 1992.

[19] V. Ng and T. Kamada, “Concurrent Accesses to R-Trees,” Proc. Third Symp. Advances in Spatial Databases (SSD ’93), pp. 142-161, 1993.

[20] J. Nievergelt, H. Hinterberger, and K.C. Sevcik, “The Grid File: An Adaptable, Symmetric Multikey File Structure,” ACM Trans. Database Systems, vol. 9, no. 1, pp. 38-71, Mar. 1984.

[21] J.A. Orenstein and T.H. Merrett, “A Class of Data Structures for Associative Searching,” Proc. Third Symp. Principles of Database Systems (PODS ’84), pp. 181-190, 1984.

[22] J.T. Robinson, “The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes,” Proc. ACM SIGMOD ’81, pp. 10-18, 1981. Conf. Very Large Data Bases (VLDB ’87), pp. 507-518, 1987.

[23] M. Abdelguerfi, J. Givaudan, K. Shaw, and R. Ladner, “The 2-3TRTree, a Trajectory-Oriented Index Structure for Fully Evolving Valid-Time Spatio-Temporal Datasets,” Proc. 10th ACM Int’l Symp. Advances in Geographic Information System (ACMGIS ’02), pp. 29-34, 2002.

[24] L. Shou, Z. Huang, and K.-L. Tan, “The Hierarchical Degree-ofVisibility Tree,” IEEE Trans. Knowledge Data Eng., vol. 16, no. 11, pp. 1357-1369, Nov. 2004.[25] S.I. Song, Y.H. Kim, and J.S. Yoo, “An Enhanced Concurrency Control Scheme for Multidimensional Index Structure,” IEEE Trans. Knowledge Data Eng., vol. 16, no. 1, pp. 97-111, Jan. 2004.

[26] Y. Theodoridis, “The R-Tree Portal,” http://www.rtreeportal.org, 2005.

[27] P.S. Yu, K.-L. Wu, K.-J. Lin, and S.H. Son, “On Real-Time Databases: Concurrency Control and Scheduling,” Proc. IEEE, vol. 82, no. 1, pp. 140-157, Jan. 1994.

[28] D. Zhang and T. Xia, “A Novel Improvement to the R-Tree Spatial Index Using Gain/Loss Metrics,” Proc. 12th ACM Int’l Symp. Advances in Geographic Information Systems (ACMGIS ’04), pp. 204-213, 2004.

[29] K.V.R. Kanth, D. Serena, and A.K. Singh, “Improved Concurrency Control Techniques for Multi-Dimensional Index Structures,” Proc. Ninth Symp. Parallel and Distributed Processing (SPDP ’98), pp. 580-586, 1998.

[30] M. Kornacker and D. Banks, “High-Concurrency Locking in R-Trees,” Proc. 21st Int’l Conf. Very Large Data Bases (VLDB ’95), pp. 134-145, 1995.

[31] M. Kornacker, C. Mohan, and J. Hellerstein, “Concurrency and Recovery in Generalized Search Trees,” Proc. ACM SIGMOD ’97, pp. 62-72, 1997.

[32] P. Lehman and S. Yao, “Efficient Locking for Concurrent Operations on B-trees,” ACM Trans. Database Systems, vol. 6,no. 4, pp. 650-670, Dec. 1981.

[33] D. Lomet, “Key Range Locking Strategies for Improved Concurrency,” Proc. 19th Int’l Conf. Very Large Data Bases (VLDB ’93), pp. 655-664, 1993.

Authors Biography

N Krishna Kumari, pursuing M.Tech. (I.T.) in Dept. of Information Technology, University College of Engineering, JNTUK - Vizianagaram Campus, VIZIANAGARAM, Andhra Pradesh, India. Participated and Presented papers at National Level, Seminars and Technical Symposiums.

Dr MHM Krishna Prasad is working as an Associate Professor and Head, Dept. of Information Technology, University College of Engineering, JNTUK- Vizianagaram Campus, VIZIANAGARAM, Andhra Pradesh, India. Received PhD from JNTU Hyderabad and M. Tech from SIT, JNTU, Hyderabad. Had vast Experience in industry and academics, and published papers in various International Conferences and Journals