Journal of Information and Operations Management
ISSN: 0976–7754 & E-ISSN: 0976–7762 , Volume 3, Issue 1, 2012, pp-289-292
Available online at http://www.bioinfo.in/contents.php?id=55

THE IMPACT OF MULTIPLE GRANULARITY ON CONCURRENCY CONTROL IN MULTI USER ENVIRONMENT

UDAI BHAN TRIVEDI AND RAMESH CHANDRA BHARTI

IMS Dehradun, India.
*Corresponding Author: Email- udaibhantrivedi@gmail.com

Received: December 12, 2011; Accepted: January 15, 2012

Abstract- In a single-user database, the user can modify data in the database without concern for other users modifying the same data at the same time. However, in a multi-user database, the statements within multiple simultaneous transactions can update the same data. Transactions executing at the same time need to produce meaningful and consistent results. Therefore, control of data concurrency and data consistency is vital in a multi-user database environment.
Data concurrency means that many users can access data at the same time.
Data consistency means that each user sees a consistent view of the data, including visible changes made by the user's own transactions and transactions of other users.To describe consistent transaction behavior when transactions execute at the same time, database researchers have defined a transaction isolation model called serializability. The serializable mode of transaction behavior tries to ensure that transactions execute in such a way that they appear to be executed one at a time, or serially, rather than concurrently. Data concurrency is very important in multi user environment and locking protocol tries to answers this problem. There are three factors affecting the performance of different lock granularities (unit of database, which can be locked by a scheduler, as a granule.). They are the lock overhead, data contention and resource contention. According to [Bern87], the finer the lock granularity adopted, the more the lock overhead involved and the higher is the degree of both the data contention and the resource contention. However, the multiprogramming level will also increase. It should be beneficial to short transactions. For long transactions, fine granularity does not help much since the portion of database to be locked will remain almost the same. [3]n [Ries77] and [Ries79], the authors concluded that coarse granularity is generally preferred except when transactions access a small part of the database randomly, in which case fine granularity is desired .
Keywords- Database sharing, Data Consistency concurrency control, locking, Multiple Granularity, Serializability, Access modes, Lock table,

Citation: Udai Bhan Trivedi and Ramesh Chandra Bharti (2012) The Impact of Multiple Granularity on Concurrency Control In Multi User Environment. Journal of Information and Operations Management ISSN: 0976–7754 & E-ISSN: 0976–7762, Volume 3, Issue 1, pp-289-292.

Introduction

Concurrency control in database systems has been a major focus of research. There are hundreds of schedulers available for concurrency control in database systems. The design of most of the schedulers are based on locking [5, 8] or timestamp ordering1 [2, 16]. When an enormous amount of data is to be managed, it is sometimes advantageous to organize the data in a hierarchical tree-like structure, especially when the hierarchy of the data is to be preserved. This helps easy management of data. For example consider a database of an educational institution. The database may be partitioned into divisions, a division into departments, a department into areas, an area into files, a file into records, and so on (Figure 3). A database is called a *tree database* if the data items are organized as nodes of a tree. In reality, a data granule could be a block of disk, a file, a record of a file, a field of a record, etc. A data granule is referred to as coarser if the granule contains relatively more ingredients than a finer granule. For example, the granularity of a file is coarser, and the granularity of a record in that file is finer. The granularity of a data item is not an important issue as far as correctness is concerned (therefore the correctness of the proposed protocol is not given here), but it is an important issue for the performance and concurrency of the system

Assumptions

1) A few assumptions were in the article. The database can be broken into some fashion of a hierarchical form, whether that is a straight hierarchy or a Directed Acyclic Graph, it uses a hierarchical form.

2) The basic principles of this locking mechanism would be possible, as long as a hierarchical structure was imposed. If a structure cannot be imposed, such as in more complex queries, this locking mechanism may lose its advantages.

This paper focus on Multiple Granularity lock based protocol and its impacts on concurrency control. This protocol has been used by major database in sharing environment for concurrency control. Before we move to the concept of multiple granularity lock base protocol the discussion of the functioning of lock manager and Transaction Spooler is very important

The Lock Manager

A global data lock tree is designed to support multigranularity locking. Each tree node is a data granule which may further represent a number of finer granules. The major components of each tree node are execution queue, wait queue and next granular level pointer. The execution queue registers those granted transaction lock accesses which are held by some not- yet-finished transactions. The wait queue is to register those not yet granted transaction lock request. For the next granular level pointer, it points to a sub tree and is null for the finest level of granule. The sub tree consists of descendant nodes of the granule. Lock compatibility table and conversion table are also designed to support multiple modes of lock and lock escalation, as shown in Table 1. The lock types supported by this prototype are intention read, intention write, read- intention-write, read and write. Figure 1 and Figure 2 describe the of Locking Process Object Diagram and Sequence Diagram

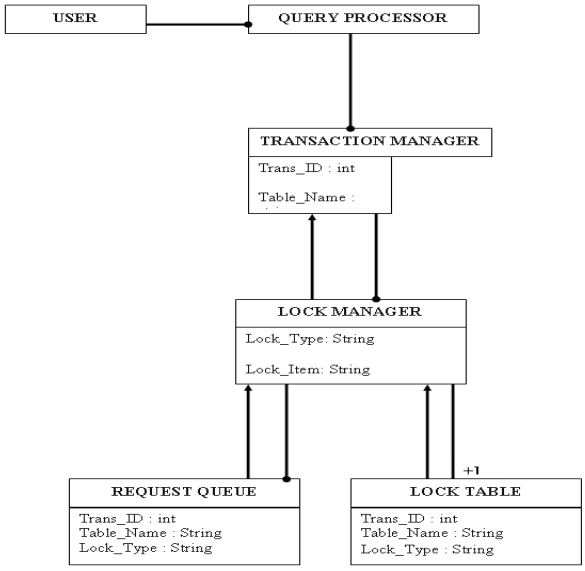

Fig.1 -Object Diagram (Indicative)

The Transaction Spooler

For transaction spooler, maintains a constant multiprogramming level by ensuring that a new transaction enters the system only after another transaction of the same class has left the system.

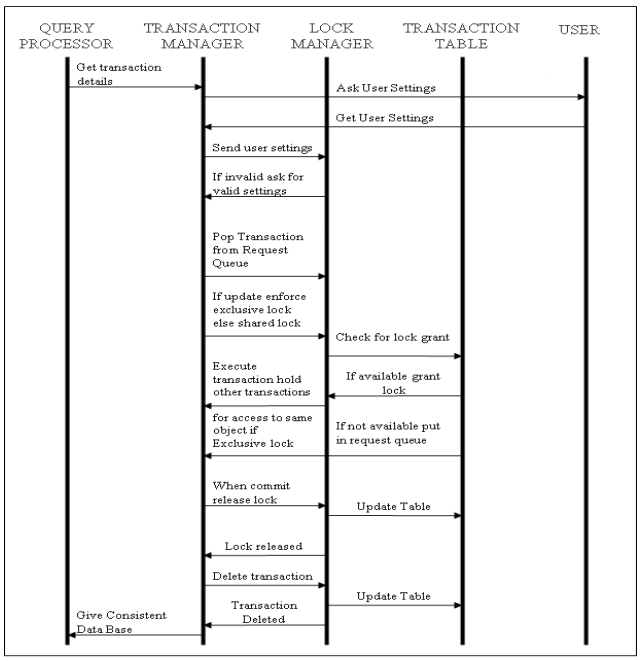

Fig. 2- Sequence Diagram (Indicative)

Hierarchical locks

We will first assume that the set of resources to be locked is organized in a hierarchy. Note that this hierarchy is used in the context of a collection of resources and has nothing to do with the data model used in a data base system. The hierarchy of Figure 3 is suggestive. We adopt the notations that each level of the hierarchy is given a node type which is a generic name for all the node instances of that type. For example, the data base has nodes of type area as its immediate descendants, each area in turn has nodes of type file as its immediate descendants and each file has nodes of type record as its immediate descendants in the hierarchy. Since it is a hierarchy, each node has a unique parent. [2]

Fig. 3- A sample locks hierarchy.

Each node of the hierarchy can be locked. If one requests exclusive access (X) to a particular node, then when the request is granted, the requestor has exclusive access to that node and implicitly to each of its descendants. If one requests shared access (S) to a particular node, then when the request is granted, the requestor has shared access to that node and implicitly to each de-

scendant of that node. These two access modes lock an entire sub tree rooted at the requested node.

Our goal is to find some technique for implicitly locking an entire sub tree. In order to lock a sub tree rooted at node R in share or exclusive mode it is important to prevent share or exclusive locks on the ancestors of R which would implicitly lock R and its descendants. Hence a new access mode, intention mode (I), is introduced. Intention mode is used to "tag" (lock) all ancestors of a node to be locked in share or exclusive mode. These tags signal the fact that locking is being done at a "finer" level and thereby prevents implicit or explicit exclusive or share locks on the ancestors. The protocol to lock a sub tree rooted at node R in exclusive or share mode is to first lock all ancestors of R in intention mode and then to lock node R in exclusive or share mode. For example, using Table 1, to lock a particular file one should obtain intention access to the data base, to the area containing the file and then request exclusive (or share) access to the file itself. This implicitly locks all records of the file in exclusive (or share) mode [4]

Multiple Granularity locking protocol

All the concurrency control protocols operate on individual data items to achieve synchronization of transactions. It is sometimes desirable, however, to able to access a set of data items as a single unit, e.g., to effectively lock each item in the set in one operation rather than having to lock each item individually. J.N Gray presented a *multiple granularity locking* protocol, which aims to minimize the number of locks used while accessing sets of objects in a database [Gray et al. 75]. In their model, Gray *et al.* organize data items in a tree where items of small granularity are nested within larger ones. Each non-leaf item represents the data associated with its descendants. This is different from the tree protocol presented above in that the nodes of the tree (or graph) do not represent the order of access of individual data items but rather the organization of data objects. The root of the tree represents the whole database. Transactions can lock nodes explicitly, which in turn locks descendants implicitly. Two modes of locks were defined: *exclusive* and *shared*. An exclusive (X) lock excludes any other transaction from accessing (reading or writing) the node; a shared (S) lock permits other transaction to read the same node concurrently, but prevents any updating of the node.

To determine whether to grant a transaction a lock on a node (given these two modes), the transaction manager would have to follow the path from the root to the node to find out if any other transaction has explicitly locked any of the ancestors of the node. This is clearly inefficient. To solve this problem, a third kind of lock mode called *intention* lock mode was introduced [Gray 78]. All the ancestors of a node must be locked in intention mode before an explicit lock can be put on the node. In particular, nodes can be locked in five different modes. A non-leaf node is locked in intention -shared (IS) mode to specify that descendant nodes will be explicitly locked in shared (S) mode. Similarly, an intention-exclusive (IX) lock implies that explicit locking is being done at a lower level in an exclusive (X) mode. A shared and intention exclusive (SIX) lock on a non-leaf node implies that the whole sub tree rooted at the node is being locked in shared mode, and that explicit locking will be done at a lower level with exclusive-mode locks. A compatibility matrix for the five kinds of locks is defined as shown in Table 1. The matrix is used to determine when to grant lock requests and when to deny them.

Access modes and compatibility:

Two lock requests for the same node by two different transactions are *compatible* if they can be granted concurrently. The mode of the request determines its compatibility with requests made by other transactions. The three modes X, S and I are incompatible with one another but distinct S requests may be granted together and distinct I requests may be granted together.

Table1. *Compatibilities among access modes*

	IS	IX	S	SIX	X
IS	Y	Y	Y	Y	N
IX	Y	Y	N	N	N
S	Y	N	Y	N	N
SIX	Y	N	N	N	N
X	N	N	N	N	N

To summarize, we recognize five modes of access to a resource:

IS: Gives intention share access to the requested node and allows the requestor to lock descendant nodes in S or IS mode. (It does no implicit locking.)

IX: Gives intention exclusive access to the requested node and allows the requestor to explicitly lock descendants in X, S, SIX, IX or IS mode. (It does no implicit locking.)

S: Gives share access to the requested node and to all descendants of the requested node without setting further locks. (It implicitly sets S locks on all descendants of the requested node.)

SIX: Gives share and intention exclusive access to the requested node. (In particular it implicitly locks all descendants of the node in share mode and allows the requestor to explicitly lock descendant nodes in X, SIX or IX mode.)

X: Gives exclusive access to the requested node and to all descendants of the requested node without setting further locks. (It implicitly sets X locks on all descendants. Locking lower nodes in S or IS mode would give no increased access.)

The multiple granularity locking protocol increases concurrency and decreases overhead especially when there is a combination of short transactions with a few accesses and transactions that last for a long time accessing a large number of objects such as audit transactions that access every item in the database. The Orion object-oriented database system provides a concurrency control mechanism based on the multi-granularity mechanism described above [Kim et al. 88; Garza and Kim 88].

Concluding Remarks

A dynamic granularity locking protocol for tree structured databases shares the advantages exhibited by both fine and coarse granularity locking protocols, and retains the power of multi-granularity locking protocol. The protocol dynamically changes the granule size of the data items to be locked depending on the transaction-requirements, the current system load condition and the conflict status of the transactions. The strategy of the protocol is to follow the principles of coarse granularity locking protocol at light system load or when conflicts are less, and to follow the principal of fine granularity locking

Protocol at heavy system load or when conflicts are more

References

[1] Gray J.N., Lorie R.A., Putzolu G.R., Traiger I.L. *IBM research Laboratory San Jose, California*.

[2] Fundamental of *Data base Systems by Korth*

[3] Imasri Ramez and Navathe Shamkant B., *Fundamental of Data base Systems*.
[4] Tamer M. OZSU and Valduriez Patric, *Principal of Distributed Database Systems*.
[5] DATE C.J. (1981) *An Introduction to Database Systems Addison-Wesley, Reading, Mass*.
[6] Gray J.N., Lorie R.A. and Putzolu G.R. (1975) *Very Large Database,* Vol. 1, pp. 428-451.
[7] Gray J.N. et al. (1976) *IFIP Modeling of Database Management System,* pp. 695-723.
[8] Korth H.F. (1981) *Distributed Datamanagement and Computing Networks*.
[9] Korth H.F. (1983) *Journal of the ACM,* Vol. 30, No. 1.
[10] Lehamn T.J. et al.(1986) *ACM SIGMOD*.
[11] Lehmn T.J. and Caray M.J.(1987) *ACM SIGMOD,* pp.104-117, 7.
[12]Ries D.R. and Stonebraker M. (1977) *ACM Tran. on Database Systems,* Vol. 2, pp. 233-246, 1977.