Transaction Model in Database System: a Survey

Meenu
Associate Professor
Department of Computer Science & Engineering
Madan Mohan Malaviya University of Technology,
Gorakhpur, India
myself_meenu@yahoo.co.in

ABSTRACT

A database is a structured and persistent collection of information about some aspect of the real world organized and stored in a way that facilitates efficient retrieval and modification. Transaction provides a consistent and reliable computing in database systems. This survey deals with defining transaction, modelling and synchronization of transaction. Various advanced transaction models have been compared by explaining their advantages and disadvantages.

Keywords

Transaction, advanced transaction model, concurrency control

1. Introduction

The term "database" was introduced in the 1960s. A database is a structured and persistent collection of information about some aspect of the real world organized and stored in a way that facilitates efficient retrieval and modification. Query is used to denote a retrieval statement on database expressed in a query language. In query, there is no consistent execution or reliable computation. On the other hand, the concept of a transaction is used in database systems as a basic unit of consistent and reliable computing. The consistency and reliability aspects of transactions are due to four ACID properties [4] of transactions i.e. atomicity, consistency, isolation, and durability. Thus, queries are executed as transactions once their execution strategies are determined and they are translated into primitive database operations. Database consistency is concerned with consistency of database i.e. it obeys all the consistency constraints defined over it. State of database changes due to modifications, insertions, and deletions together called updates. The database can be temporarily inconsistent during the execution of a transaction, but it should be consistent when the transaction terminates. On the other hand, transaction consistency refers to the actions of concurrent transactions. The database should be in a consistent state even if there are a number of users that are concurrently accessing (reading or updating) the database [28]. In 1981, Gray [1] indicated that the transaction concept has its roots in contract law. He states, "In making a contract, two or more parties negotiate for a while and then make a deal. The deal is made binding by the joint signature of a document or by some other act (as simple as a handshake or a nod). If the parties are rather suspicious of one another or just want to be safe, they appoint an intermediary (usually called an escrow officer) to coordinate the commitment of the transaction." The nice aspect of this historical perspective is that it does indeed encompass some of the fundamental properties of a transaction (atomicity and durability) as the term is used in database systems. It also serves to indicate the differences between a transaction and a query. In 1986, Papadimitriou [2,28] stated that a transaction may be thought of as a program with embedded database access queries. In 1988, Ullman [3] defined transaction as a single execution of a program. In 1983, Andreas Reuter and Theo Harder [4] coined the acronym ACID as shorthand for Atomicity, Consistency, Isolation, and Durability, building on earlier work by Jim Gray [1] in 1981 ,who enumerated Atomicity, Consistency, and Durability but left out Isolation when characterizing the transaction concept. These four properties describe the

major guarantees of the transaction paradigm, which has influenced many aspects of development in database systems. According to Gray and Reuter [5] , IMS supported ACID transactions as early as 1973 (although the term ACID came later).

- Atomicity. The "all or nothing" property. A transaction is an indivisible unit that is either performed in its entirety or is not performed at all.

- Consistency. A transaction must transform the database from one consistent state to another consistent state. In 1976, Gray [6] gave classification for grouping databases into four levels of consistency.

- Isolation. Transactions execute independently of one another. In other words, the partial effects of incomplete transactions should not be visible to other transactions. In 1992, ANSI [7] as part of the SQL2 (also known as SQL-92) standard specification, has defined a set of isolation levels.

- Durability The effects of a successfully completed (committed) transaction are permanently recorded in the database and must not be lost because of a subsequent failure. Atomicity and durability are the responsibility of the Recovery subsystem; Isolation and, to some extent, Consistency are the responsibility of the Concurrency control subsystem. Transaction is a set of read or write operation used to perform a unit of work. Transactions can terminate successfully (commit) or unsuccessfully (abort). Aborted transactions must be undone or rolled back. The transaction is also the unit of concurrency and the unit of recovery. A variety of commit protocols have been proposed by database researchers.

2. Transaction Model

A transaction is modelled as a fixed sequence of actions: $T = << t, A_i, N_i >| i=1\ldots n>$ where t is the transaction name, Ai are operations and Ni are entity names [49]. Transaction model access simple database objects (sets of tuples or a physical page). A number of transaction models have been proposed for a class of applications. In 1978, Gray [8] suggested, transaction model.

2.1 The Read/Write Model

In 1993, Rastogi [9] presented the best known transaction model that considers database objects to be pages in memory which can be read or written in one atomic operation; thus, transactions in this model are sequences of read and write operations, thereby abstracting from all kinds of computations that a user program might execute in main memory or in its buffer.

2.2 Model of Relational Updates

In 1988, Abiteboul [10] gave a model, in which the basic operations, to be executed atomically, are insertions of single tuples, and deletions or modifications of sets of tuples (satisfying a given condition), all applied to the relations in a given database. Transactions have been classified according to a number of criteria. One criterion is the duration of transactions. In 1987, Gray [11] classified transactions as online or batch. These two classes are also called short-life and long-life transactions, respectively. Online transactions are characterized by very short execution response times (typically, on the order of a couple of seconds) and by access to a relatively small portion of the database. Batch transactions, on the other hand, take longer to execute (response time being measured in minutes, hours, or even days) and access a larger portion of the database. Another classification is with respect to the organization of the read and write actions. If the transactions intermix their read and write actions without any specific ordering, then this type of transactions is general. In 1979, Papadimitriou [12] gave a classification in which if the transactions are restricted so that all the read actions are performed before any write action, the transaction is called a two-step transaction. In 1976 Stearns [13] gave a classification in which if the transaction is restricted so that a data item has to be read before it can be updated (written), the corresponding class is called restricted (or read-before-write).If a transaction is both two

step and restricted, it is called a restricted two-step transaction. In 1979, Kung and Papadimitriou [14] gave action model of transactions which consists of the restricted class with each <read, write> pair be executed atomically.

2.3 Transaction Model in DRTDBMS

In 1988, Michael J. Carey [15] developed a single, uniform, distributed DBMS model for studying a variety of concurrency control algorithms and performance trade-offs. Each site in the model has four components: a source, which generates transactions and also maintains transaction-level performance information for the site, a transaction manager, which models the execution behaviour of transactions, a concurrency control manager, which implements the details of concurrency control algorithm, and a resource manager, which models the CPU and I/O resources of the site. In addition to this, the model has a network manager which models the behaviour of the communications network.

2.4 Advanced Transaction Models

There are two basic features advanced models can bring along are further operational abstractions, and a departure from strict ACID. For the former, many options are available, providing more operations, higher-level operations, more execution control within and between transactions, or providing more transaction structure. Structure refers to parallelism inside a transaction, transactions inside other transactions, or it can be transactions plus other operations inside other transactions. Departure from pure ACID, means that for many applications today ACID transactions are too restrictive. The transaction protocols so far are suitable for the types of transaction for traditional business applications, such as banking system. Some of applications result in transactions that are very complex, access many data items, and are of long duration, possibly running for hours, days, or even months. The longer the transaction runs, the more likely it is that deadlock will occur if a locking-based protocol is used. In 1981, Gray [1] showed that the frequency of deadlock increases to the fourth power of the transaction size. One way to achieve cooperation among people is through the use of shared data items. However, the traditional transaction management protocols significantly restrict this type of cooperation by requiring the isolation of incomplete transactions.

Following are advanced transaction models [27,29]:

- Nested transaction model
- Sagas;
- Multilevel Transaction Model
- Dynamic Restructuring
- Workflow Models

2.4.1 Nested transaction model

In 1981, Moss [16] introduced nested transaction model. In this model, the complete transaction forms a tree, or hierarchy, of subtransactions. There is a top-level transaction that can have a number of child transactions; each child transaction can also have nested transactions. In Moss's original proposal, only the leaf-level subtransactions (the subtransactions at the lowest level of nesting) are allowed to perform the database operations.

2.4.2 Sagas

In 1987, Garcia-Molina and Salem [17] introduced the concept of Sagas. and is based on the use of compensating transactions. The DBMS guarantees that either all the transactions in a saga are successfully completed or compensating transactions are run to recover from partial execution. Unlike a nested transaction, which has an arbitrary level of nesting, a Saga has only one level of nesting. Further, for every subtransaction that is defined, there is a corresponding compensating transaction that will semantically undo the subtransaction's effect

2.4.3 Multilevel Transaction Model

The nested transaction model requires the commit process to occur in a bottom-up fashion through the top-level transaction. This is a closed nested transaction, as the semantics of these transactions enforce atomicity at the

top level. In contrast, there is open nested transactions, which relax this condition and allow the partial results of subtransactions to be observed outside the transaction. The saga model is an example of an open nested transaction. In 1991, Weikum [18] and Scheck introduced a specialization of the open nested transaction called multilevel transaction model where the tree of subtransactions is balanced. Nodes at the same depth of the tree correspond to operations of the same level of abstraction in a DBMS. The edges in the tree represent the implementation of an operation by a sequence of operations at the next lower level.

2.4.4 Dynamic Restructuring

In 1988 ,Pu [19] addressed the constraints imposed by the ACID properties of flat transactions by proposing two new operations split transaction and join-transaction .The principle behind split-transactions is to split an active transaction into two serializable transactions and divide its actions and resources (for example, locked data items) between the new transactions. This allows the partial results of a transaction to be shared with other transactions while preserving its semantics; that is, if the original transaction conformed to the ACID properties, then so will the new transactions.The split-transaction operation can be applied only when it is possible to generate two transactions that are serializable with each other and with all other concurrently executing transactions.The join-transaction performs the reverse operation of the split-transaction, merging the ongoing work of two or more independent transactions as though these transactions had always been a single transaction. A split-transaction followed by a join-transaction on one of the newly created transactions can be used to transfer resources among particular transactions without having to make the resources available to other transactions.

The main advantages of the dynamic restructuring method are:

• Adaptive recovery, which allows part of the work done by a transaction to be committed so that it will not be affected by subsequent failures.

• Reducing isolation, which allows resources to be released by committing part of the transaction.

2.4.5 Workflow Models

The models so far have been developed to overcome the limitations of the flat transaction model for transactions that may be long-lived. However, these models are still not sufficiently powerful for business activities. More complex models have been proposed that are combinations of open and nested transactions. However, as these models hardly conform to any of the ACID properties, the more appropriate name workflow model has been used instead.

In 1995, Rusinkiewicz and Sheth [20] defined the key issues in specifying a workflow.

• Task specification. The execution structure of each task is defined by providing a set of externally observable execution states and a set of transitions between these states.

• Task coordination requirements. These are usually expressed as intertask-execution dependencies and data-flow dependencies, as well as the termination conditions of the workflow.

• Execution (correctness) requirements. These restrict the execution of the workflow to meet application-specific correctness criteria. It includes failure and execution atomicity requirements and workflow concurrency control and recovery requirements.

3. Transaction processing system

In 1985, Jim Gray [21] in collaboration with 24 others from academy and industry, published an article, "A Measure of Transaction Processing Power.” in issue of Datamation. This article outlined a test for on-line transaction processing which was given the title of "DebitCredit." Transaction processing is a way of computing that divides work into individual, indivisible operations, called transactions. Transaction processing [50] is distinct from and can be contrasted with other computer processing models, such as batch processing, time-sharing, and processing. Batch processing is execution of a series of programs (jobs) on a computer without manual intervention. Real time systems attempt to guarantee an appropriate response to a stimulus or request quickly enough to affect the conditions that caused the stimulus. A transaction processing system (TPS) [22,26] is a software system, or software/hardware combination, that supports

transaction processing. A Transaction Processing System (TPS) is a type of information system that collects, stores, modifies and retrieves the data transactions of an enterprise. Transaction processing systems also attempt to provide predictable response times to requests, although this is not as critical as for real-time systems. Rather than allowing the user to run arbitrary programs as time-sharing, transaction processing allows only predefined, structured transactions.The first transaction processing system was SABRE [23] , made by IBM for American Airlines, which became operational in 1970.The Hewlett-Packard Non-stop system (formerly Tandem_Nonstop) [24] was a hardware and software system designed for Online Transaction Processing (OLTP*)* introduced in 1976.

4. Concurrency in DBMS

A major objective in developing a database is to enable many users to access shared data concurrently. Concurrent access is relatively easy if all users are only reading data, as there is no way that they can interfere with one another [25]. However, when two or more users are accessing the database simultaneously and at least one is updating data, there may be interference that can result in inconsistencies. Three potential problems caused by concurrency are the lost update problem, the uncommitted dependency problem (or dirty read), and the inconsistent analysis problem. A successfully completed update operation by one user can be overridden by another user. This is known as the lost update problem. The uncommitted dependency problem occurs when one transaction is allowed to see the intermediate results of another transaction before it has committed .The problem of inconsistent analysis occurs when a transaction reads several values from the database but a second transaction updates some of them during the execution of the first. Another problem can occur when a transaction T rereads a data item it has previously read but, in between, another transaction has modified it. Thus, T receives two different values for the same data item. This is known as a nonrepeatable (or fuzzy) read. A similar problem can occur if transaction T executes a query that retrieves a set of tuples from a relation satisfying a certain predicate, re-executes the query at a later time, but finds that the retrieved set contains an additional (phantom) tuple that has been inserted by another transaction in the meantime. This is known as a phantom read.

5. Serializability

Serial execution prevents concurrency problems. Serial execution never leaves the database in an inconsistent state, so every serial execution is considered correct, although different results may be produced. The objective of Serializability is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another, and thereby produce a database state that could be produced by a serial execution. If a set of transactions executes concurrently, the (nonserial) schedule is correct if it produces the same result as some serial execution. Such a schedule is called serializable. To prevent inconsistency from transactions interfering with one another, it is essential to guarantee serializability of concurrent transactions. Serializability identifies those executions of transactions that are guaranteed to ensure consistency [30]. Serializability can be achieved in several ways. Two main concurrency control techniques to ensure serializability of concurrent transactions are locking and timestamping.

6. SYNCHRONIZATION of TRANSACTIONS

Concurrency control is the process of managing simultaneous operations on the database without having them interfere with one another. The objective of a concurrency control protocol is to schedule transactions in such a way as to avoid any interference between them and hence prevent the types of problem caused by concurrency. There are a number of ways that the concurrency control approaches can be classified. One classification criterion is the mode of database distribution. Some algorithms require a fully replicated database, while others can operate on partially replicated or partitioned databases. The concurrency control algorithms may also be classified according to network topology, such as those requiring a communication subnet with broadcasting capability or those working in a star-type network or a circularly

connected network. The most common classification criterion is the synchronization primitive. The corresponding breakdown of the concurrency control algorithms results in two classes [31] those algorithms that are based on mutually exclusive access to shared data (locking), and those that attempt to order the execution of the transactions according to a set of rules (protocols). However, these primitives may be used in algorithms with two different viewpoints: the pessimistic view that many transactions will conflict with each other, or the optimistic view that not too many transactions will conflict with one another.

6.1 Pessimistic Locking and Timestamping Methods

The pessimistic view assumes that many transactions will conflict with each other. Pessimistic algorithms synchronize the concurrent execution of transactions early in their execution life cycle. The pessimistic group consists of locking based algorithms, ordering (or transaction ordering) based algorithms, and hybrid algorithms. Pessimistic CC do not permit a transaction to access a data item if there is a conflicting transaction that accesses that data item. The execution of any operation of a transaction follows the sequence of phases: validation (V), read (R), computation (C), write (W). This sequence is valid for an update transaction as well as for each of its operations.

6.1.1 Concurrency control using Pessimistic Locking Methods

Locking methods are the most widely used approach to ensure serializability of concurrent transactions. Locking is a procedure used to control concurrent access to data. When one transaction is accessing the database, a lock may deny access to other transactions to prevent incorrect results. If a transaction has a shared lock on a data item, it can read the item but not update it. If a transaction has an exclusive lock on a data item, it can both read and update the item. Data items of various sizes, ranging from the entire database down to a field, may be locked. The size of the item determines the fineness, or granularity, of the lock. Although locking allows greater concurrency, it permits transactions to interfere with one another, resulting in the loss of total isolation and atomicity.

- **2PL**

To guarantee serializability, we must follow an additional protocol concerning the positioning of the lock and unlock operations in every transaction. The best-known protocol is two-phase locking (2PL). The use of locks, combined with the two-phase locking protocol, guarantees serializability of schedules According to the rules of 2PL protocol, every transaction can be divided into two phases: first a growing phase, in which it acquires all the locks needed but cannot release any locks, and then a shrinking phase, in which it releases its locks but cannot acquire any new locks. There is no requirement that all locks be obtained simultaneously. Normally, the transaction acquires some locks, does some processing, and goes on to acquire additional locks as needed.2PL prevents three potential problems caused by concurrency i.e. the lost update problem, the uncommitted dependency problem (or dirty read), and the inconsistent analysis problem. It can be proved that if every transaction in a schedule follows the two-phase locking protocol, then the schedule is guaranteed to be conflict serializable [32].However, although the two-phase locking protocol guarantees serializability, problems can occur with the interpretation of when locks can be released. The situation, in which a single transaction leads to a series of rollbacks, is called cascading rollback. Cascading rollbacks are undesirable, because they lead to the undoing of a significant amount of work.

- **Rigorous 2PL**

Rigorous 2PL protocols is a variant of 2PL which prevents cascading rollbacks. It leaves the release of all locks until the end of the transaction**.**

- **Strict 2PL**

Another variant of 2PL, called strict 2PL, holds only exclusive locks (write locks) until the end of the transaction.

- **Centralized 2PL(C2PL) or Primary site 2PL**

The 2PL algorithm can easily be extended to the distributed DBMS environment. C2PL

delegates lock management responsibility to a single site only. This means that only one of the sites has a lock manager; the transaction managers at the other sites communicate with it rather than with their own lock managers. This technique is also known as the primary site 2PL algorithm [33]. Like primary copy 2PL, this approach tends to require more communication than basic2PL, since data manager (DM) reads and prewrites usually cannot implicitly request locks.

- **Primary Copy 2PL**

Primary copy 2PL is a 2PL technique that considers data redundancy [34] One copy of each logical data item is designated the primary copy; before accessing any copy of the logical data item, the appropriate lock must be obtained on the primary copy. For readlocks this technique requires more communication than basic 2PL.

- **Voting 2PL (or majority consensus 2PL)**

Voting 2PL (or majority consensus 2PL) is another 2PL implementation that considers data redundancy. Voting 2PL is derived from the majority consensus technique of Thomas [35] and is only suitable for ww synchronization.2PL is inappropriate for rw synchronization as this technique requests locks on all copies.

- **Distributed 2PL**

Distributed 2PL (D2PL) requires the availability of lock managers at each site. Distributed 2PL algorithms have been used in System R* [36] and in NonStop SQL [37,38] and [39]

6.1.2 Concurrency control using Pessimistic Timestamping Methods

A different approach that also guarantees serializability uses transaction timestamps to order transaction execution for an equivalent serial schedule. Timestamping is a concurrency control protocol that orders transactions in such a way that older transactions, transactions with smaller timestamps, get priority in the event of conflict. Timestamping methods have timestamp which is a unique identifier created by the DBMS that indicates the relative starting time of a transaction. Uniqueness is only one of the properties of timestamp generation. The second property is monotonicity. Two timestamps generated by the same transaction manager should be monotonically increasing. It is this second property that differentiates a timestamp from a transaction identifier. Timestamps can be generated by simply using the system clock at the time the transaction started, or, more normally, by incrementing a logical counter every time a new transaction starts. Locking methods may have deadlock. In Timestamp methods for concurrency control no locks are involved and therefore there can be no deadlock. Locking methods generally prevent conflicts by making transactions wait. With timestamp methods, there is no waiting, so transactions involved in conflict are simply rolled back and restarted.

- **Basic timestamp ordering**

The basic timestamp ordering protocol assumes that only one version of a data item exists, and so only one transaction can access a data item at a time. With timestamping, if a transaction attempts to read or write a data item, then the read or write is only allowed to proceed if the last update on that data item was carried out by an older transaction. Otherwise, the transaction requesting the read/write is restarted and given a new timestamp. New timestamps must be assigned to restarted transactions to prevent their being continually aborted and restarted. Without new timestamps, a transaction with an old timestamp might not be able to commit owing to younger transactions having already committed. In addition to timestamps for transactions, there are timestamps for data items. Each data item contains a read_timestamp, giving the timestamp of the last transaction to read the item, and a write_timestamp, giving the timestamp of the last transaction to write (update) the item. Basic timestamp ordering guarantees that transactions are conflict serializable, and the results are equivalent to a serial schedule in which the transactions are executed in chronological order of the timestamps. In other words, the results will be as if all of transaction

1 were executed, then all of transaction 2, and so on, with no interleaving. The basic TO algorithm tries to execute an operation as soon as it is accepted; it is therefore "aggressive" or "progressive. The basic TO algorithm never causes operations to wait, but instead, restarts them. Even though this is an advantage due to deadlock freedom, it is also a disadvantage, because numerous restarts would have adverse performance implications. However, basic timestamp ordering does not guarantee recoverable schedules.

- **Conservative TO Algorithm**

The basic TO algorithm never causes operations to wait, but instead, restarts them. Even though this is an advantage due to deadlock freedom, it is also a disadvantage, because numerous restarts would have adverse performance implications. The conservative TO algorithms attempts to lower this system overhead by reducing the number of transaction restarts. The "conservative" nature of these algorithms relates to the way they execute each operation. The basic TO algorithm tries to execute an operation as soon as it is accepted; it is therefore "aggressive" or "progressive." Conservative algorithms, on the other hand, delay each operation until there is an assurance that no operation with a smaller timestamp can arrive at that scheduler. If this condition can be guaranteed, the scheduler will never reject an operation. However, this delay introduces the possibility of deadlocks. One possible implementation of the conservative TO algorithm is Herman and Verjus algorithm [40].

- **Multiversion Timestamp Ordering**

Multiversion TO is another attempt at eliminating the restart overhead cost of transactions. The basic timestamp ordering protocol assumes that only one version of a data item exists, and so only one transaction can access a data item at a time. This restriction can be relaxed by versioning of data. Versioning of data can also be used to increase concurrency, as different users may work concurrently on different versions of the same object instead of having to wait for each other's transactions to complete. In the event that the work appears faulty at any stage, it should be possible to roll back the work to some valid state. Reed [41] proposed one concurrency control scheme that uses versions to increase concurrency based on timestamps. In multiversion concurrency control, each write operation creates a new version of a data item while retaining the old version. When a transaction attempts to read a data item, the system selects the correct version of the data item according to the timestamp of the requesting transaction. (1983). Versions have been used as an alternative to the nested and multilevel concurrency control protocols [42,43,44]

7. Concurrency control using Optimistic Locking and Timestamping Methods

In some environments, conflicts between transactions are rare and the additional processing required by locking or timestamping protocols is unnecessary for many of the transactions. Optimistic techniques are based on the assumption that conflict is rare and that it is more efficient to allow transactions to proceed without imposing delays to ensure serializability. The optimistic group can, similarly, be classified as locking-based or timestamp ordering-based. An operation submitted to an optimistic scheduler is never delayed. The read, compute, and write operations of each transaction are processed freely without updating the actual database. Locking-based optimistic concurrency control algorithms have been designed. [45]. However, the original optimistic proposals [46] are based on timestamp ordering. Thomas's write rule is a modification to the basic timestamp ordering protocol that relaxes conflict serializability can be used to provide greater concurrency by rejecting obsolete write operations [47]. When a transaction wishes to commit, a check is performed to determine whether conflict has occurred. If there has been a conflict, the transaction must be rolled back and restarted. Because the premise is that conflict occurs very infrequently, rollback will be rare. The overhead involved in restarting a transaction may be considerable, as it effectively means redoing the entire transaction. This could be tolerated only if it happened very infrequently, in which case the majority of transactions will

be processed without being subjected to any delays. These techniques allow greater concurrency than traditional protocols, as no locking is required.

There are two or three phases to an optimistic concurrency control protocol, depending on whether it is a read-only or an update transaction:

• Read phase. This extends from the start of the transaction until immediately before the commit. The transaction reads the values of all data items it needs from the database and stores them in local variables. Updates are applied to a local copy of the data, not to the database itself.

•Validation phase. This follows the read phase. Checks are performed to ensure that serializability is not violated if the transaction updates are applied to the database.

For a read-only transaction, this consists of checking whether the data values read are still the current values for the corresponding data items. If no interference occurred, the transaction is committed. If interference occurred, the transaction is aborted and restarted.

For a transaction that has updates, validation consists of determining whether the current transaction leaves the database in a consistent state, with serializability maintained. If not, the transaction is aborted and needs to be restarted.

• Write phase. This follows the successful validation phase for update transactions. During this phase, the updates made to the local copy are applied to the database.

Optimistic TO technique differs from pessimistic TO-based algorithms not only by being optimistic but also in its assignment of timestamps. Timestamps are associated only with transactions, not with data items (i.e., there are no read or write timestamps). Furthermore, timestamps are not assigned to transactions at their initiation but at the beginning of their validation step. This is because the timestamps are needed only during the validation phase as their early assignment may cause unnecessary transaction rejections. An advantage of optimistic concurrency control algorithms is their potential to allow a higher level of concurrency. It has been shown that when transaction conflicts are very rare, the optimistic mechanism performs better than locking. A major problem with optimistic algorithms is the higher storage cost. To validate a transaction, the optimistic mechanism has to store the read and the write sets of several other terminated transactions. Another problem is starvation. Consider a situation in which the validation phase of a long transaction fails. In subsequent trials it is still possible that the validation will fail repeatedly. It is possible to solve this problem by permitting the transaction exclusive access to the database after a specified number of trials. However, this reduces the level of concurrency to a single transaction.

8. Performance metric of concurrency control techniques

To evaluate the performance of concurrency control techniques, there are some evaluation metrics [48] such as:

1. Accuracy

It is the percent at which correctness of data can be achieved.

2. Serializability

It ensures that the schedule for the concurrent execution of the transactions yields consistent results and transaction can be executed in the same order of sending their request.

3. Number of Committed, Wait and Rollback Transactions

It defines number of committed transactions (transactions that executed successfully and commit their updates), number of wait transactions (transactions that still wait their order to commit their updates) and number of rollback transactions (transactions that aborted from system due to occurrence of conflict with committed one). Good algorithm provides highest number of committed transactions than wait transactions and lowest number of rollback transactions.

4. Deadlock

It occurs when two transactions wait indefinitely for each other to unlock data.

5. Storage

It is memory and RAM requirements for database storage and versions created from it.

8. FUTURE RESEARCH DIRECTIONS

An important area is to do the performance evaluation of transaction concurrency control and recovery algorithms to determine the performance metric for the transactions which are not known yet. Since performance analysis can determine the cost related with the techniques, the research in this direction can be helpful in decreasing the concurrency and recovery cost. Transactions are to be extended in distributed and parallel database environments.

9. CONCLUSION

In this paper, we have reviewed existing work in the area of basic and advanced modelling and synchronization of transaction. We have presented the research ideas in a concise way so that readers will be well informed about the type of work done on transaction management. Performance evaluation of concurrency control and recovery algorithms in database transaction is still an open area of research. Nowadays there is a high interest in using the transaction concept in contexts other than databases i.e. parallel programming.

REFERENCES

1. Gray,Jim (September1981)."The Transaction Concept: Virtues and Limitations" (PDF). Proceedings of the 7th International Conference on Very Large Databases. Cupertino, CA: Tandem Computers. pp. 144–154. Retrieved March 27, 2015.
2. Papadimitriou, C. H. (1986). The Theory of Database Concurrency Control. Computer Science Press.
3. Ullman, J. D. (1988). Principles of Database and Knowledge Base Systems, volume 1. Computer Science Press.
4. Haerder, T.; Reuter, A. (1983). "Principles of transaction-oriented database recovery". ACM Computing Surveys. 15 (4): 287. CiteSeerX 10.1.1.115.8124. doi:10.1145/289.291.
5. Gray, Jim & Andreas Reuter. Distributed Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993; ISBN 1-55860-190-2.
6. Gray, J. N., Lorie, R. A., Putzolu, G. R., and Traiger, I. L. (1976). Granularity of locks and degrees of consistency in a shared database. In Nijssen, G. M., editor, Modelling in Data Base Management Systems, pages 365–394. North-Holland.
7. ANSI (1992). Database Language SQL, ansi x3.135-1992 edition.
8. Gray, J.: Notes on data base operating systems. In R. Bayer, R.M. Graham, and G. Seegmiiller (Eds.): Operating Systems -- An Advanced Course. Lecture Notes in Computer Science, Vol. 60, Springer-Verlag, Berlin, 1978, pp. 393-481
9. Rastogi, R., Mehrotra, S., Breitbart, Y., Korth, H.F., Silberschatz, A.: On correctness of non-serializable executions. In: Proc. 12th ACM SIGACT-SIGMODSIGART Syrup. Principles of Database Systems, 1993, pp. 97-108.
10. Abiteboul, S., Vianu, V.: Equivalence and optimization of relational transactions. *J. ACM 35* (1988) 70-120.
11. Gray, J. (1987). Why do computers stop and what can be done about it. In CIPS (Canadian Information Processing Society) Edmonton ’87 Conf. Tutorial Notes, Edmonton, Canada.
12. Papadimitriou, C. H. (1979). Serializability of concurrent database updates. J. ACM,26(4):631–653.
13. Stearns, R. E., II, P. M. L., and Rosenkrantz, D. J. (1976). Concurrency controls for database systems. In Proc. 17th Symp. on Foundations of Computer Science, pages 19–32.
14. Kung, H. T. and Papadimitriou, C. H. (1979). An optimality theory of concurrency control for databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data,pages 116–125.
15. Michael J. Carey. and Miron Livny. “Distributed Concurrency Control Performance: A Study of Algorithm,

Distribution, and Replication, Proceedings of the 14th VLDB Conference, Los Angels, California Page(s): 13–25, 1988.

16. Moss J.E.B. (1981). Nested transactions: An approach to reliable distributed computing. PhD dissertation, MIT, Cambridge, MA
17. Garcia-Molina H. and Salem K. (1987). Sagas. In *Proc. ACM. Conf. on Management of Data*, 249–259
18. Weikum G. and Schek H. (1991). Multi-level transactions and open nested transactions. IEEE Data Engineering Bulletin
19. Pu C., Kaiser G., and Hutchinson N. (1988). Splittransactions for open-ended activities. In Proc. 14th Int. Conf. Very Large Data Bases
20. Rusinkiewicz M. and Sheth A. (1995). Specification and execution of transactional workflows. In*Modern Database Systems*. (Kim W., ed.), ACM Press/Addison-Wesley, 592–620
21. Anon et al., "A Measure of Transaction Processing Power," *Datamation*, Volume 31, Number 7, April 1, 1985, pages 112-118. Tandem TR at HP Labs
22. https://en.wikipedia.org/wiki/Transaction_processing_system
23. https://www.sabre.com/files/Sabre-History.pdf
24. "Tandem History: An Introduction". Center magazine, vol 6 number 1, Winter 1986, a magazine for Tandem employees.
25. Gerhard Weikum, Gottfried Vossen, Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2001.
26. Philip A. Bernstein and Eric Newcomer, Principles of Transaction Processing, Morgan Kaufmann, 1997. 3.
27. Ahmed Elmagarmid (Ed.), Database Transaction Models for Advanced Applications. Morgan Kaufmann, 1992
28. Özsu, M. Tamer & Valduriez, P & Seshadri, Dr.Sridhar. (2006). Principles of Distributed Database systems. 10.13140/RG.2.1.2514.5361.
29. Thomas Connolly, Carolyn E. Begg, Database Systems: A Practical Approach to Design, Implementation and Management ,2nd Ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1998.
30. Papadimitriou, C. H. (1979). Serializability of concurrent database updates. J. ACM,26 (4):631–653
31. Bernstein, P. A. and Goodman, N. (1981). Concurrency control in distributed database systems. ACM Comput. Surv., 13(2):185–222
32. Eswaran, K. P., Gray, J. N., Lorie, R. A., and Traiger, I. L. (1976). The notions of consistency and predicate locks in a database system. Commun. ACM, 19(11):624–633
33. Alsberg, P. A. and Day, J. D. (1976). A principle for resilient sharing of distributed resources. In Proc. 2nd Int. Conf. on Software Engineering, pages 562–570
34. STONEBRAKER, M. "Concurrency control and consistency of multiple copies of data in distributed INGRES, IEEE Trans. Softw. Eng. SE-5, 3 (May 1979),188-194.
35. THOMAS, R.H. "A solution to the concurrency control problem for multiple copy databases," in Proc. 1978 COMPCON Conf. (IEEE), New York
36. Mohan, C., Lindsay, B., and Obermarck, R. (1986). Transaction management in the r* distributed database management system. ACM Trans. Database Syst.,11(4):378–396
37. Tandem (1987). Nonstop sql – a distributed high-performance, high-availability implementation of sql. In Proc. Int. Workshop on High Performance Transaction Systems, pages 60–104.
38. Tandem (1988). A benchmark of nonstop sql on the debit credit transaction. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 337–341.
39. Borr, A. (1988). High performance SQL through low-level system integration. In Proc. ACM SIGMOD

Int. Conf. on Management of Data, pages 342–349.

40. Herman, D. and Verjus, J. P. (1979). An algorithm for maintaining the consistency of multiple copies. In Proc. 1st Int. Conf. on Distributed Computing Systems, pages 625–631

41. REED, D.P. (1978) Naming and synchronization in a decentralized computer system, Ph.D. dissertation, Dept. of Electrical Engineering, M.I.T., Cambridge, Mass.,Sept.

42. Beech D. and Mahbod B. (1988). Generalized version control in an object-oriented database. In IEEE 4th Int. Conf. Data Engineering, February

43. Chou H.T. and Kim W. (1986). A unifying framework for versions in a CAD environment. In Proc. Int. Conf. Very Large Data Bases, Kyoto, Japan, August 1986,336–344.

44. Chou H.T. and Kim W. (1988). Versions and change notification in an object-oriented database system. In Proc. Design Automation Conference, June 1988,275–281

45. Bernstein, P. A., Hadzilacos, V., and Goodman, N. (1987). Concurrency Control and Recovery in Database Systems. Addison Wesley.

46. Kung, H. T. and Robinson, J. T. (1981). On optimistic methods for concurrency control. ACM Trans. Database Syst., 6(2):213–226.

47. Thomas, R. H. (1979). A majority consensus approach to concurrency control for multiple copy databases. ACM Trans. Database Syst.

48. Sharafeldin, Marwa & Badawy, Mohammed & El-Sayed, Ayman. (2016). Survey on Concurrency Control Techniques. Communications on Applied Electronics (CAE) Journal. 05. 28-31. 10.5120/cae2016652194.

49. Jim Gray*:* A Transaction Model. ICALP 1980: 282-298

50. https://en.wikipedia.org/wiki/Transaction_processing