Correctness Criteria for Transaction Processing: A Survey and Analysis

Authors
1 S. Meenakshi, 2 Dr. V. Thiagarasu

Address for correspondence:
1 Associate Professor of Computer Science, Gobi Arts & Science College, Gobichettipalayam
2 Professor of Computer Science, Gobi Arts & Science College, Gobichettipalayam, Tamilnadu

Abstract—Transaction processing and concurrency control issues have played a major role in databases and hence have been an important area of research for many years. A main objective in developing a database is to enable users to access shared data concurrently. When a database is shared and updated by multiple transactions concurrently, the requirement for concurrency control is needed to ensure the correctness of the database. In general, serializability is used as a correctness criterion for concurrency control in database applications. However serializability is not adequate for advanced database applications characterized by long duration transactions with cooperative access to shared data. Hence there is a need for flexible correctness criteria for processing those transactions more efficiently and this has motivated to survey the various correctness criteria proposed in the literature for transaction processing in database systems. Also this paper analysis the requirements for effective transaction processing in advanced applications that function in distributed, cooperative and heterogeneous environments.

Keywords—Database; concurrency control; correctness criteria; serializability; transaction processing

I. INTRODUCTION

Database systems are essential for many applications ranging from space station operations to commercial problems. A main objective in developing a database is to enable many users to access shared data concurrently. When a database is shared and updated by multiple transactions concurrently, the requirement for concurrency control is needed to ensure the correctness of the database. There are two criteria for defining the correctness of the database: database integrity and serializability [6]. The database integrity is satisfied by assigning a set of constraints that must be satisfied for a database to be correct. The serializability ensures that database transitions from one state to the other are based on an interleaved execution of a set of concurrent transactions.

In general serializability is used as a correctness criterion for concurrency control in database applications [6, 20]. Serializability requires that the execution of each transaction must appear to every other transaction as a single atomic step. (i.e.) the execution of each transaction cannot be interrupted by other transactions. While serializability with atomicity property (atomic transaction model) has been successfully used in traditional applications(commercial/financial applications such as banking, air line reservation etc) characterized by short duration transactions with competitive access to shared data, it is restrictive and hardly applicable in nontraditional applications characterized by long duration transactions with cooperative access to shared data such as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Multimedia systems, Digital Publishing, Geographic Information Systems(GIS), Interactive dynamic websites and network management[3]. These applications require the use of serializability in a relaxed way and this has motivated the introduction of flexible correctness criteria that go beyond traditional serializability.

The remaining sections of this paper are organized as follows. Section 2 describes the concept of schedules based on serializability. Section 3 study the various proposed flexible correctness criteria for processing concurrent transactions in database applications. Section 4 outlines the techniques used for achieving concurrency control. Section 5 shows the requirements for effective transaction processing in modern database applications and Section 6 concludes the paper.

II. SCHEDULES BASED ON SERIALIZABILITY

A sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transaction is called a schedule [8]. A transaction comprises a sequence of operations consisting of read and/or writes action to the database, followed by a commit or abort action. A schedule S consists of sequence of the operations from a set of ‘n’ transactions T1, T2..., Tn, subject to the constraint, that the order of operations for each transaction is preserved in the schedule. Thus for each transaction Tj in schedule S, the order of the operations in Tj must be same in the schedule S. When concurrent transactions are executing the
types of schedules that are considered are serial and nonserial schedules. A schedule where the operations of each transaction are executed consecutively without any interleaved operations from other transactions is called a serial schedule. A schedule where the operations from a set of concurrent transaction are interleaved is called a nonserial schedule. In a serial schedule there is no interference between transactions, so it leaves the database in a consistent state but database system may make poor use of their resources and hence resulting in smaller transaction execution rate. In a nonserial schedule, interleaving of transactions improves transaction execution rate but it may produce concurrency problems such as lost update, dirty read, and unrepeatable read. Hence the concept of serializability has been proposed as the correctness criterion for concurrency control in transactions. The objective of serializability is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another and also produce a database state that could be produced by a serial execution. Serializability ensures that a concurrent schedule of transactions is equivalent to the serial schedule of the same transactions [11]. Serializability in database systems has been guaranteed through the designing of concurrency control algorithms [11, 8].

Serializability has been successfully used in traditional database applications and is not appropriate in advanced applications because of different kinds of consistency constraints [7]. If enough information is known about the transactions and operations, a nonserializable but consistent schedule can be constructed. In fact, equating the notions of consistency with serializability causes the significant loss of concurrency in advanced applications [3]. Serializability is too restrictive criterion for long running cooperative applications since it prevents a transaction from seeing the intermediate results of another transaction [3, 4, and 7].

III. CORRECTNESS CRITERIA FOR CONCURRENCY CONTROL

To overcome the limitations of the serializability concept, a number of flexible correctness criteria have been proposed in the literature [7, 17, 19, 5, 12, 10, 1, 11a, 14] to serve the different application requirements. This section describes those concepts in brief and Table I summarizes it.

Cooperative Serializability (COSR) [7] has been proposed as an alternative correctness criterion for cooperative databases and it is defined with respect to a set of transactions which maintain the consistency properties. Transactions form cooperative transaction sets and a cooperative transaction set could be formed by the components of an extended transaction or transactions collaborating over the objects while maintaining the consistency of the objects. In such cases, consistency can be maintained if other transactions which do not belong to the set are serialized with respect to all the transactions in the set. In other words, a set of cooperative transactions becomes the unit of concurrency with respect to serializability.

Epsilon-Serializability (ESR) [17] has been introduced as a generalization of serializability where a limited amount of inconsistency is permitted and the concurrency is enhanced by permitting the nonserializable schedules. ESR defines correctness for both consistent and inconsistent database states.

ESR introduces the notion of epsilon transactions (ETs) by attaching a specification of the amount of permitted inconsistency to each transaction. ESR aims at bounding the amount of imported and exported inconsistency for each ET. ESR distinguishes between transactions that contain only read operation, called query epsilon transaction and transactions with at least one update operation called update epsilon transaction. Query ETs may view uncommitted, possibly inconsistent, data being updated by update ETs. Thus updates ETs are seen as exporting the inconsistencies while query ETs are importing these inconsistencies.

Eventual Consistency has been proposed as an alternative correctness criterion for distributed databases with replicated or interdependent data [19]. A correctness criterion that ensures eventual consistencies is the current copy serializability. Each update occurs on a current copy and is asynchronously propagated to other replicas. Eventual consistency requires that duplicate copies are consistent at certain times but may be inconsistent in the interim intervals. The basic idea is that duplicates are allowed to diverge as long as the copies are made consistent periodically. The times where these copies are made consistent could depend on the application.

Multiversion Serializability (MVS) in multiversion databases [5] each write on a data item ‘x’ produces a new version (or copy) of ‘x’. For each read on ‘x’ the database selects one of the versions of ‘x’ to be read and write do not overwrite each other and since reads can read any version of the database. Hence concurrency is increased by having transactions read older versions while other concurrent transactions are creating newer versions. In a multiversion database only one type of conflict is possible; when a transaction reads a version of a data item that was written by another transaction. Based on the assumption that users expect their transactions to behave as if there were just one copy of each data item, the notion of one-copy serial schedule is defined. A schedule is one-copy serial (1-serial) if for all i, j, and x, if a transaction Ti reads ‘x’ from a transaction Ti, then either i ≤ j or Ti is the last transaction preceding Ti that writes into any version of ‘x’. Hence a schedule is defined as one copy serializability (1-SR) if it is equivalent to a 1-serial schedule.

Predicatewise Serializability (PWSR) has been proposed as a correctness criterion for long duration transaction processing environments such as CAD and Office information systems [12]. PWSR focus on database consistency constraints that can be expressed in conjunctive normal form. If database consistency constraints can be expressed in a conjunctive normal form, a schedule is said to be PWSR if all projections of that schedule on each group of data items that share a disjunctive clause are serializable. The restrictions that must be enforced on PWSR schedules to preserve database consistency are: force the transactions to be of fixed structure, i.e., they are independent of the database state from which they execute, force the schedules to be delayed read, i.e., a transaction Ti cannot read a data item written by a transaction Tj until after Tj has completed all of its operations, and the conjuncts of the integrity constraints can be ordered in a way that no transaction
Quasi Serializability (QSR) is a correctness criterion for global concurrency control in heterogeneous distributed database systems (HDDBS) [10]. HDDBS consists of a set of local databases (LDBS) and a set of local and global transactions: Local transactions that access only one LDBS and global transactions that access more than one LDBS. A global schedule in an HDDBS is a set of local schedules where the local schedule is defined over global and local transactions. The basic idea of QSR is that in order to preserve global database consistency, global transactions should be executed in a serializable way with proper consideration of the effects of local transactions. A quasi serial schedule is a schedule where global transactions are required to execute serially and local schedules are required to be serializable. A global schedule in an HDDBS is said to be quasi serializable if it is (conflict) equivalent to a quasi serial schedule.

Relative serializability (RSR) has been introduced as a correctness criterion for collaborative databases [1]. RSR introduces the notion of relative atomicity of coactions as a relaxation of the traditional atomicity property. It is used to specify how coactions can be interleaved relative to other coactions without breaking the overall atomicity requirement for the collaboration activity. The main idea is that before a collaborative activity takes place first a collaborative channel is established and then correctness is checked against RSR. That is any execution obeying the RSR criterion would preserve the consistency of the database even it is not serializable.

Semantic Serializability is an approach for controlling concurrency that exploits the semantic information available about transactions to allow controlled nonserializable interleavings [11a]. The semantic information about transaction takes the form of transaction types, transaction steps and transaction break points. An approach used to define the notion of consistency called relative consistency (RC) that describes the action of allowable interleavings among transactions that known to be correct and also to ensure that each schedule produced by the system is equivalent to a correct schedule. Enforcing relative consistency requires knowledge about the application which must be provided by the user and the transaction failure can be handled by using the method of counter steps. In particular, users will need to group actions of the transactions into steps and specify which steps of a transaction of a given type can be interleaved with the steps of another type of transactions without violating consistency.

Two-level Serializability (2LSR) is a correctness criterion for multibase database systems (MDBS) to relax serializability requirements and allow higher degree of concurrency [14]. MDBS consists of set of heterogeneous autonomous pre-existing local databases and a set of local and global transactions. 2LSR schedules preserve consistency by exploiting the nature of integrity constraints and transactions in multibase database environments. A global schedule consisting of both local and global transactions is 2LSR if all local schedules are serializable and the projection of that schedule on global transactions is serializable. Local schedules consist of all operations, from global and local transactions, that access the same local database.

IV. CONCURRENCY CONTROL TECHNIQUES

The universally accepted correctness criterion for processing transactions against a database is serializability [20] and it can be achieved through various concurrency control algorithms. The objective of concurrency control protocol is to schedule transactions in such a way to avoid any interference between them and also to correctly process transactions that are in conflict [6]. Two transactions conflict if the read set of one transaction intersects with the write set of the other transaction and/or the write set of one transaction conflicts with the write set of the other transaction. It must be noted the transactions for example T_1 and T_2 can conflict, only if both are executing at the same time. If T_1 has finished before T_2 was submitted to the system, even if their read and write sets intersect, they are not considered to be in conflict.

There are basically three generic techniques that can be used to design concurrency control algorithms [4, 6 and 8]. They are locking, timestamp ordering (pessimistic) and optimistic. Pessimistic approaches cause transactions to be delayed (wait or rollback) in case they conflict with other transactions at some time in the future. Optimistic methods are based on the premise that conflict is rare so they allow transactions to continue unsynchronized and only check for conflicts at the end when a transaction commits. Based on the above approaches the five main concurrency control algorithms used in the literature are two-phase locking (2PL), multi-granularity locking, timestamp ordering, multiversion timestamp ordering and optimistic concurrency control [3, 8]. Two-phase locking (2PL) is the most commonly implemented concurrency control algorithm in commercial database systems [2, 20].

V. TRENDS IN ADVANCED APPLICATIONS

Transaction processing in different application contexts is still a challenging task. The nature of concurrent behavior in advanced applications have arrived the following requirements for concurrency control [3, 8]: (i) Supporting long transactions-Long transactions need different support than short transactions. In particular blocking a transaction until another transaction commits (2PL) is not suitable for long transactions. (ii) Supporting synergistic cooperation-Cooperation among transactions is essential to exchange and sharing knowledge in cooperative environments. (iii) Support for complex objects-The complexity of the structure and size of the objects in advanced applications require the need for object oriented technology.(iv) Supporting user control-In order to support tasks that are nondeterministic and interactive in nature, the concurrency control mechanism should provide the user with interactively execute operations, commit or abort at any time.

Today, with the growth in e-commerce and the blurring of enterprise boundaries, there is a renewed interest in business processes (BPs) coordination, especially for inter-organizational processes [9]. A recent trend in transaction management focuses on adding transactional properties to BPs [9]. Since BPs contains activities that access shared and
persistent data resources, they have to be subject to transational semantics. However, it is not adequate to treat an entire BP as a single traditional transaction mainly because BPs: (i) are of long running and treating an entire process as a transaction would require locking resources for a long period of time, (ii) involve many independent database and application systems and enforcing transactional properties across the entire process would require expensive coordination among these systems.

Recently the major databases and applications behind the need for flexible concurrency control for transaction processing include object oriented databases, active databases, mobile databases, web databases, collaborative applications, medical applications, business process applications and software development environments. These advanced applications are characterized by long and cooperative transactions in which concurrency must be optimized for adequate overall performance.

The various correctness criteria presented in section 3 aim at achieving one or several of the following goals: a) accept nonserializable but correct executions by using the semantics of transactions, their structure, and integrity constraints b) allow inconsistencies to appear in a controlled manner which may be acceptable for the transactions, c) limit conflicts by creating a new version of the data for each update, d) treat transactions accessing more than one database, in the case of multidatabases differently from those accessing one single database and maintain over all correctness.

Also the above surveyed extended serializability criteria, relax the serializability and atomicity properties and capture database consistency requirements and transaction correctness properties via a single notion. But these goals and properties are highly undesirable consequences for handling transactions in advanced applications because of its above requirements. Recent researches have attempted to come up with correctness criteria that view these requirements effectively by the introduction of advanced transaction models that extend the atomic transaction model.

VI. CONCLUSION

This paper has given a survey of correctness criteria for handling concurrent transaction processing in database systems. The various correctness criteria studied in this paper supports the concept of extended serializability or relaxing the atomicity properties for ensuring the correctness of transaction processing in specific application domains. Section 5 reveals the need for flexible concurrency control with suitable transaction model that meet the requirements of today’s modern applications.

<table>
<thead>
<tr>
<th>Correctness criterion</th>
<th>Concept</th>
<th>Area of application</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooperative Serializability</td>
<td>A cooperative transaction set could be formed by the components of an</td>
<td>Cooperative databases.</td>
<td>[7]</td>
</tr>
<tr>
<td>Epsilon Serializability</td>
<td>extended transaction or transactions collaborating over the objects while maintaining the consistency of objects.</td>
<td>Applications that tolerate the inconsistencies.</td>
<td>[17]</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>Requires that duplicate copies are consistent at certain times but may be inconsistent in the interim intervals</td>
<td>Distributed databases with replicated or interdependent data.</td>
<td>[19]</td>
</tr>
<tr>
<td>Multiversion Serializability</td>
<td>Allows the schedules as 1-SR serializable if it is equivalent to 1-serial schedule. Concurrency control correctness depends on translation as well as order of operations in a transaction.</td>
<td>Multiversion database systems.</td>
<td>[5]</td>
</tr>
<tr>
<td>Predicatewise Serializability</td>
<td>Focuses on data integrity constraints.</td>
<td>CAD database and office information system.</td>
<td>[12]</td>
</tr>
<tr>
<td>Quasi Serializability</td>
<td>Executes global transactions in a serializable way while taking into account the effect of local transactions.</td>
<td>Multidatabase systems.</td>
<td>[10]</td>
</tr>
<tr>
<td>Relative Serializability</td>
<td>Co-actions can be interleaved relating to other co-actions without breaking the overall atomicity requirement for the collaboration activity.</td>
<td>Collaborative databases.</td>
<td>[1]</td>
</tr>
<tr>
<td>Semantic Serializability</td>
<td>Uses semantic information about transactions to accept the nonserializable but correct schedules.</td>
<td>Applications that can provide the semantic knowledge.</td>
<td>[11a]</td>
</tr>
<tr>
<td>Two-level Serializability</td>
<td>Ensures consistency by exploiting the nature of integrity constraints and the nature of transactions in multidatabase systems.</td>
<td>Multidatabase systems.</td>
<td>[14]</td>
</tr>
</tbody>
</table>

REFERENCES


