Concurrency Control is an integral part of database system. This chapter deals with a detailed survey of various concerning concurrency control problems and their solutions have been formalized and implemented.

3.1 CONCURRENCY CONTROL APPROACHES

There are basically three generic approaches: Locking, Timestamp and Optimistic that can be used to design concurrency control algorithms which are used to synchronize concurrent transactions [MJC1984] [CWE2004]. Figure 3.1 shows the classification of concurrency control approaches.

The synchronization can be accomplished by utilizing:

- **Wait**: If two transactions conflict, conflicting actions of one transaction must wait until the actions of the other transactions are completed.

- **Timestamp**: The order in which transactions are executed is selected based on a time stamp. Each transaction is assigned a unique timestamp by the system and conflicting actions of two transactions are processed in timestamp order.

- **Rollback**: If two transactions conflict, some actions of a transaction are undone or rolled back or else one of the transactions is restarted. This approach is also called optimistic because it is expected that conflicts are such that only a few transactions would rollback.

The main concern in designing a concurrency control algorithm is to correctly process transactions that are in conflict. Each transaction has a read set and a write set. 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. Transactions T1 and T2 can conflict only if both are executing at the same time. If, for example, T1 has finished before T2 was submitted to the system, even if their read and write sets intersect, they are not considered to be in conflict [VAT1994].
3.2 LITERATURE REVIEW

Concurrency control in distributed database has been the focus of research for many years. One research uses divergence control lock model based on prudent order sharing and a check-out / check-in protocol supporting transactions disconnections [GQL2003]. A recent work considering three states of mobile devices: good connection, reasonable connection and no connection takes the advantage of the capabilities of the different connection alternatives to send large data and small data respectively [NGY2006]. By using the notion of planned disconnection, Multi-Check-out Timestamp Order Technique (MCTO), a framework has been designed which makes the replicated data of mobile nodes available to access and update in low cost for reading and writing [JTA2006]. Fixed end timeout technique for planned disconnection in mobile database over fixed network with weighted data distribution is a milestone [MAK2002]. Mobile clients
Chapter 3: Concurrency Control Algorithms and Its Variants: A survey

encounter wide variations in connectivity ranging from high-bandwidth, low latency communications through wired networks to total lack of connectivity [JHO2000]. These attributes can be supported by using weak read and write operations which access only local and potentially inconsistent copies and perform updates using bounded inconsistency which is defined by allowing controlled variation among copies located at weakly connected sites [EPI1997].

In dynamically configurable environments, the lengthy transmission delay of some networks, frequent and unpredictable disconnections and long inactivity periods of users could affect transaction duration generating long running transactions which cause low concurrency rate, deadlocks and starvation etc. [ACH2008]. Mobile transaction concurrency can be increased using semantic knowledge to relax the notion of absolute transaction atomicity by providing a high degree of inter-transaction parallelism [ABR2005]. A weaker form of consistency can also work in a number of situations in mobile environment and use $\varepsilon$-Serializability (ESR) which tolerates a limited amount of inconsistency specified by $\varepsilon$ [NPR2004].

In mobile client/server database environments, a service for replication is needed to guarantee availability of data in disconnected mode also. One such replication service provides an interface to mobile applications and administrators. Using this interface, applications can select data from server database for replication into a local database on demand [CGO2003]. Blocking problem can be reduced by attaching multiple back-up sites to the coordinator site [KMI2004]. Blocking caused by complex transaction structure can be reduced by using a method called ASGT (Active Serialization Graph Technique) which first normalizes the structure of the transaction to a tree structure with operations as its nodes and then applies a pruning method to cut the problems caused by complex structure of transactions [LXI2006]. Data allocation based on Ontology Schema for minimizing transmission cost also has been proposed which uses data fragmentation instead of data replication at different sites based on user requirement [JJS2003]. System performance can be improved by using the Data Caching. Caching stores desired data in the local storage of data processing node to improve data availability and data access time [PKR2003]. In mobile applications, synchronization architectures are required to provide
Chapter 3: Concurrency Control Algorithms and Its Variants: A survey

the best speed, network traffic and storage. ERDA (Efficient Remote Data Access) provides efficient pull based replication and synchronization for mobile databases using optimistic methods. Another approach for synchronization is deferred updates i.e. each client synchronizes with the server when an update is required. In other words, a view is updated only when a query is updated against it and the client is connected to the server [VKU2005]. But if database sites are interconnected in a rooted tree fashion, then for concurrency among global transactions, a non-two-phase locking protocol is used which guarantees deadlock freedom in addition to serializability [KVI1991].

3.3 CLASSIFICATION OF CONCURRENCY CONTROL ALGORITHMS

3.3.1 Algorithms Based on Locking Mechanism

When two transactions conflict, one solution is to make one transaction wait until the other transaction has released the entities common to both. To implement this, the system can provide locks on the database entities. Transactions can get a lock [SAM2007] on an entity from the system, keep it as long as the particular entity is being operated upon and then give the lock back. If a transaction requests the system for a lock on an entity and the lock has been given to some other transaction, the requesting transaction must wait [ECH2007]. To reduce the waiting time when a transaction wants to read, there are two types of locks that can be employed, based on whether the transaction wants to do a read operation or a write operation on an entity:

- **Read lock**: The transaction locks the entity in a shared mode. Any other transaction waiting to read the same entity can also obtain a read lock.
- **Write lock**: The transaction locks the entity in an exclusive mode. If one transaction wants to write on an Entity, no other transaction may get either a read lock or a write lock.

After a transaction has finished operations on an entity, the transaction can do an unlock operation. After an unlock operation, either type of lock is released and the entity is made available to other transactions that may be waiting.
Chapter 3: Concurrency Control Algorithms and Its Variants: A survey

Since the correctness criterion for concurrently processing several transactions is serializability, locking must be done correctly to assure the above property. One simple protocol that all transactions can obey to ensure serializability is called Two-phase Locking (2PL). The protocol simply requires that in any transaction, all locks must precede all unlocks. A transaction operates in two phases: The first phase is the locking phase, and the second phase is the unlocking phase [ZTA2006].

3.3.1.1 Variations of Locking Algorithm

- Writing in the backup site (using BC protocol): In [PKR2003], one backup site is attached to each coordinator site. After receiving responses from all participants in the first phase, the coordinator communicates its decision only to its backup site in the backup phase. Afterwards, it sends final decision to participants. When blocking occurs due to the failure of the coordinator site, the participant sites consult coordinator’s backup site and follow termination protocols. In this way, BC protocol achieves non-blocking property in most of the coordinator site failures. If both the coordinator and its backup site fail simultaneously, the participants wait until the recovery of either the coordinator site or the backup site. BC protocol has following merits. First, it eliminates the blocking of transactions in most of the coordinator failures. Second, it ensures consistency of the database in case of partitioning failures. And third, the performance of BC protocol is close to 2PC (Two Phase Concurrency) protocol. With these merits, BC protocol becomes a good choice for commit processing in DDBS environments where frequent site failures occur and messages take longer delivery time.

- Locking at Central Node, Execution at all Nodes (LCN Method): Instead of executing the transaction at the central node, we can only assign the locks at the central node and send the transaction back to node X. The transaction Ti is executed at node X. The values of the read set are read and the values of the write set are obtained at node X. Node X sends the values of the write set and obtains acknowledgments from all other nodes. It then knows that transaction Ti has been completed. The node X sends
a message to unlock entities referenced by Ti. The central node after receiving this message releases the locks and starts assigning locks to waiting transactions.

- **Global Two-phase Locking (G2PL):** In G2PL, instead of a transaction getting all locks in the beginning and releasing all locks in the end, the policy of two-phase locking is employed. Each transaction obtains the necessary locks as they are needed and then releases locks on entities that are no longer needed. A transaction cannot get a lock after it has released any lock.

- **Divergence control lock model based on prudent ordered sharing and a Check-Out/Check-In protocol supporting disconnections (DC/POS-PAI-2PL):** A divergence control lock model based on prudent ordered sharing reduces blocking in concurrency control. A Check-Out/Check-In protocol is given which can support disconnected transactions to continue their executions. DC/POS-PAI-2PL, a two-phase lock strategy integrating these methods for mobile real-time transactions is implemented [GQL2003]. This protocol is meant for the flat transaction model and based on the serializability. Basic locking model could be extended as follows. Mobile real-time transactions are divided into query transactions (QT) and updated transactions (UT). QT lock data objects with query lock and UTs lock data objects with read lock (R) or write lock (W) according to its operation types. The relations between QTs and UTs include compatible (Y), non-compatible (N) and limited compatible (LY) which may be LY-1 representing that the QTs is allowed to read the dirty data modified by the UTs; or LY-2 representing that the UTs is allowed to modify the data read by QTs. The relations among UTs include compatible (Y), non-compatible (N) and ordered compatible (notated OY).

Ordered compatibility only occurs when transactions with lower priority are blocked by transactions with higher priority. In addition, in order to avoid “cascade abort”, it is not allowed that R lock is ordered compatible with W lock. The lock model extended as above is called the divergence control lock model based on prudent ordered sharing.

A Check-Out/Check-In protocol is given which can support disconnected transactions to continue their executions after a fixed server has received a “Check Out” message
from a mobile host, it creates an “image transaction” to replace the transaction in the mobile host to check out data. Its lock compatible matrix is shown as in Table 3.1.

**Table 3.1 Compatible Matrix for Divergence Control Locking Model for Prudent Ordered Sharing**

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>Q</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>Q</td>
<td>Y</td>
<td>Y</td>
<td>LY-1</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>LY-2</td>
<td>N/OY</td>
<td>N/OY</td>
<td></td>
</tr>
</tbody>
</table>

The image transaction can not end (commit or abort) until receiving a “Check In” message or its deadline is expired. The image transactions have the same deadlines as their original transactions respectively. The transaction managers in the fixed servers can independently abort the expired image transactions and it is not necessary to wait “Check In” message from corresponding transactions in mobile hosts. Therefore, the database consistency in the fixed servers is ensured by the image transaction mechanism. And the unnecessary blockings are also eliminated, since the conflicts between the fixed transactions and the disconnected transactions are actually transformed into the conflicts between the fixed transactions and the “image transactions” of the disconnected transactions.

- **S2PL**: A Secure multilevel database based on secure 2PC protocol in which two security levels: high and low are used [SME1997]. A primary concern in multilevel security is information leakage by a high security level transaction to a transaction executing at a low security level. Leakage can occur in two ways: directly through an overt operation such as reading a data item or indirectly through a covert channel [NKA2005]. Direct leakage can be prevented by mandatory access control policies such as the Bell-La Padula (BL) model [DEB2005] but handling of covert channel needs modifications in conventional concurrency control schemes such as Two-Phase Locking (2PL) and Timestamp Ordering (TO).

- **Primary Copy Locking (PCL)**: In this variant, instead of selecting a node as the central controller, a copy of each entity on any node is designated as the primary copy
of the entity. A transaction must obtain the lock on the primary copy of all entities referenced by it. At any given time, the primary copy contains the most up-to-date value for that entity.

- **Weaker Consistency:** Mobile computing introduces a new form of distributed computation in which communication is most often intermittent with low bandwidth availability thus providing only weak connectivity [LXI2006]. Evaggelia et al. [EPI1997] proposed a replication scheme that supports weak connectivity and disconnected operation by balancing network availability against consistency guarantees. A dual database interface is proposed that in addition to read and write operations with the usual semantics, supports weak read and write operations. In contrast to the usual read and write operations that read consistent values and perform permanent updates, weak operations access only local and potentially inconsistent copies and perform updates that are only conditionally committed. Exploiting weak operations supports disconnected operation since mobile clients can employ them to continue to operate even while disconnected. The extended database interface coupled with bounded inconsistency offers a flexible mechanism for adapting replica consistency to the networking conditions by appropriately balancing the use of weak and normal operations. Bounded inconsistency is defined by allowing controlled deviation among copies located at weakly connected sites. Adjusting the degree of divergence among copies provides additional support for adaptability. In the proposed scheme, data located at strongly connected sites are grouped together to form clusters. Mutual consistency is required for copies located at the same cluster, while degrees of inconsistency are tolerated for copies at different clusters. The interface offered by the database management system is enhanced with operations providing weaker consistency guarantees. Such weak operations allow access to locally, i.e., in a cluster, available data. Weak reads access bounded inconsistent copies and weak writes make conditional updates. The usual operations called “strict” are also supported. They offer access to consistent data and perform permanent updates. The scheme supports disconnected operation since users can operate even when disconnected by using only weak operations. In case of weak connectivity, a balanced
use of both weak and strict operations provides for better bandwidth utilization, latency and cost. Core copies are copies that have permanent values, while quasi copies are copies that have only conditionally committed values. When connectivity is restored, the values of core and quasi copies of each data item are reconciled to attain a system-wide consistent value.

### 3.3.2 Algorithms Based on Time-Stamp Mechanism

Timestamp is a mechanism in which the serialization order is selected a priori; the transaction execution is obliged to obey this order. In timestamp ordering, each transaction is assigned a unique timestamp by the scheduler or concurrency controller. Obviously, to achieve unique timestamps for transactions arriving at different nodes of a distributed system, all clocks at all nodes must be synchronized or else two identical timestamps must be resolved.

- **Clock Synchronization by message passing (CSM Method):** Lamport [LLA1978] has described an algorithm to synchronize distributed clocks via message passing. If a message arrives at a local node from a remote node with a higher timestamp, it is assumed that the local clock is slow or behind. The local clock is incremented to the timestamp of the recently received message. In this way, all clocks are advanced until they are synchronized. In the other scheme where two identical timestamps must not be assigned to two transactions, each node assigns a timestamp to only one transaction at each tick of the clock. In addition, the local clock time is stored in higher-order bits and the node identifiers are stored in the lower-order bits. Because node identifiers are different, this procedure will ensure unique timestamps. When the operations of two transactions conflict, they are required to be processed in timestamp order. It is easy to prove that timestamp ordering (TSO) produces Serializable histories.

- **CSM:** Thomas [RHT1979] has studied the correctness and implementation of [LLA1978] approach and described it. Essentially each node processes conflicting operations in timestamp about its direct predecessor only. Each read-write conflict relation and write-write conflict relation is resolved by timestamp order.
Consequently all paths in the relation are in timestamp order and since all transactions have unique timestamps, it follows that no cycles are possible in a graph representing transaction histories.

- **OSN Method**: A method for concurrency control in distributed DBMS’s which increases the level of concurrent execution of transactions called ordering by serialization numbers (OSN) is proposed [UHA1989]. The OSN method works in the certifier mode and uses time interval technique in conjunction with short-term locks to provide serializability. Deadlocks are prevented by the method. The scheduler is distributed and the standard transaction execution policy is assumed, that is, the read and writes operations are issued continuously during transaction execution. However, the write operations are copied into the database only when the transaction commits. The amount of concurrency provided by the OSN method is demonstrated by log classification. It is shown that the OSN method provides more concurrency than basic timestamp ordering and two-phase locking methods and handles successfully some logs which cannot be handled by any of the past methods. The complexity analysis of the algorithm indicates that the method will work in a reasonable amount of time. Furthermore, it is shown that the communication cost of the OSN method is comparable to the communication cost of the past methods. A performance analysis realized through simulation showed that the OSN method performs better than the basic timestamp ordering and there is a reasonable operational region where it controls distributed database systems, distributed schedulers, serializability and also performs better than the two-phase locking technique.

- **Deferred Sync**: In mobile databases, Deferred Sync [KMI2004], unlike existing synchronization techniques, supports multiple file types and applications, allows for a large network capacity, supports intermittently connected devices and most importantly, minimizes data transmission load. Deferred sync converts relational data into XML tree structure and then makes use of deferred views in order to minimize bandwidth and storage space for the client by using a conflict detection and resolution algorithm based on priority selection to synchronize the data. Using a view, the client can retrieve only the data relevant to them instead of transferring the entire database.
In addition, deferred updating allows a series of updates to be batched together. In order to resolve conflicts, a combination of priority selection and latest timestamp ordering is used. Deferred sync makes use of the view in XML to coordinate the synchronization between devices. Each view is contained in XML whereby each device sees a portion of the entire database view stored on a central server. A device can see a portion of the database and modify any records within that view. Each modified field in a record is assigned a status flag which keeps track of which device has modified which record. If a device modifies a record, its corresponding device ID will be stored in the flag. When the device is ready to synchronize, an XSLT (Extensible Style sheet Language Transformations) will be applied to make sure only records that have been modified will be committed. Along with synchronizing, a log file is kept with a date time stamp to keep track of when the device was last submitted. When a server receives an update from a device, a check is needed to tell when the device last synchronization has occurred. This is to ensure that the device has not missed any previous updates or structure changes within the database. For example, if a device has been disconnected for quite long time, many updates from many other devices may have affected the disconnected device view. When the disconnected device finally synchronizes, a “catch-up” package must be made whereby the reconnected device view is synchronized with each entry contained in the catch-up package. The catch-up package contains numerous time stamped views that match the reconnected device view in the synchronization. This will result in a most up-to-date result set which is then committed to the server view. To perform synchronization, the server checks each field from the device request and compares with the corresponding field in the server view. If there are no conflicts, the field is committed and stored. If a conflict occurs, conflict resolution must be applied. In this algorithm, a rule-based approach is applied whereby based on priority of devices, if the connecting device has higher priority over an update already completed by another device, then the connecting device change is committed, otherwise the synchronization is ignored. After the synchronization is completed, a log file entry is created to note when the device was last connected with the server. Then the updated
view can be returned to the device in order to replace its view and after done it may disconnect from the network, if needed. Deferred sync operates in a unique 3-step process. Whenever a mobile device requests synchronization, the log-files are used to perform an Updates Catch-Up.

- **SDD-1**: In this approach, it is assumed that the read set and the write set of every transaction is known in advance. This information is used to group transactions into predefined classes. A transaction class is defined by a read set and a write set. A transaction \( T \) is a member of a class \( C \) if the read set of \( T \) is a subset of the read set of class \( C \) and the write set of \( T \) is a subset of the write set of class \( C \). Class definitions are used to provide concurrency control. This mechanism was used in the development of a prototype distributed database management system called SDD-1, developed by the Computer Corporation of America. SDD-1 (System for Distributed Database) permits data to be stored redundantly at several database sites in order to enhance the reliability and responsiveness of the system and to facilitate upward scaling of system. Redundant updating can be costly because it may potentially involve extensive inter-computer communication overhead in order to lock all copies of data being updated. The method described in this approach avoids this overhead by identifying cases in which it is not necessary to perform this global database locking. The identification of transactions that do not require global locking is based on transaction classes performed by the SDD-1. The classes defined are used at run time to decide what level of synchronization is needed for a given transaction. The system employs a series of four synchronization protocols which vary in cost and provide varying levels of synchronization control. The protocol selection procedure is driven by two tables established at database design time. One of these tables, called the transaction class table, is defined by the database administrator based on his knowledge of the kinds of transactions that are expected for each database application. The other table is protocol table that tells the system what protocol it must use when a transaction of a given type (or class) is entered into the system. The protocol table embodies a detailed analysis of all the possible ways, the transactions in the defined classes can interfere with each other. At run time, SDD-1 consults this
Chapter 3: Concurrency Control Algorithms and Its Variants: A survey

table to determine very quickly and concisely what level of synchronization is needed to run any given transaction. It is important to note that this predefinition and preanalysis activity in no way limits the types of transactions the system can accept; it merely permits more efficient execution of the transaction types that were anticipated. Transactions that were not anticipated ahead of time and thus not included in the preanalysis are simply handled via the strongest protocol, protocol P4. The effectiveness of the methodology is certainly application dependent: its effectiveness depends on the relative number of transactions for the application that may run under each of the protocols.

3.3.3 Algorithms Based on Optimistic Control Mechanisms

Concurrency control schemes are typically categorized as follows: pessimistic control and optimistic control, where multiple worker processors handle transactions and require I/O to disk servers. Each worker processor is composed of CPU, memories, clocks, etc. Disk servers are dedicated to store database files and data blocks. In general pessimistic concurrency controls use lock mechanism in order to maintain serialization. Under lock based pessimistic concurrency control, a transaction locks the object that will be accessed. After it finishes operations on the object, it unlocks the object. The pessimistic concurrency control is vulnerable to deadlock.

Optimistic concurrency control \[\text{[ABO2002][DAG1987]}\] highly takes advantage of parallelism because there is no delay by lock.

- **Optimistic Concurrency Control (OCC Method):** In optimistic concurrency control, validity of write operation is known not in time when it invoked but in time when transaction of it is validated. Thus write operation must be delayed and cache \[\text{[VKU2005]}\] operation for the write operation also be delayed. In order to reduce such delay, Tae-Young Cheo \[\text{[TYC2008]}\] combined cache coherency control and optimistic concurrency control rather than they operate in separate. Optimistic concurrency control which includes cache control in parallel database system is introduced. Two basic concurrency control schemes, direct validation scheme and reduced validation scheme are devised according to the degree of cache activity.
Chapter 3: Concurrency Control Algorithms and Its Variants: A survey

Based on how many caches retain its activity, concurrency controls are classified as direct validation scheme and reduced validation scheme. Direct validation scheme preserves activity of cache and it invalidates transaction which tries to accesses invalidated object again. Reduced validation scheme uses 'reduction of I/O frequency' of cache and reduces the number of access for an object to one or two. In the case that the access number is one, the transaction is valid.

Table 3.2 Variants and their respective Properties

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Protocol</th>
<th>Eliminating blocking</th>
<th>Consistency assured</th>
<th>Load Balancing</th>
<th>Increased Efficiency</th>
<th>Reduced Overhead</th>
<th>Flexibility</th>
<th>Security</th>
<th>Deadlock prevention</th>
<th>Lesser comm. cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>BC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>2.</td>
<td>LCN</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>3.</td>
<td>G2PL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>4.</td>
<td>DC/POS -PAI-2PL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>5.</td>
<td>S2PL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>6.</td>
<td>PCL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>7.</td>
<td>Weaker consistency</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>8.</td>
<td>CSM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>9.</td>
<td>CSM Correctness</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>10.</td>
<td>OSN Method</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>11.</td>
<td>Deferred Sync</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>12.</td>
<td>SDD-1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>13.</td>
<td>OCC Method</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
3.4 CONCLUSION

Concurrency Control is a problem that arises when multiple processes are involved in any part of the system. In most commercial systems, the most popular mechanism for concurrency control is two-phase locking. In this chapter, comparative study among different variants of lock algorithm, time stamp ordering and optimistic concurrency control algorithms which have been implemented over last 20 years in distributed, mobile databases have been done based on various chosen parameters like reduced blocking, consistency, load balancing, efficiency, security etc. as mentioned in Table 3.2. Depending upon application’s requirement and resources available for a system, suitability of a particular variant to be used, can be decided for a specific environment.

Due to the many interesting ideas that come into play in distributed database systems in the context of replication and reliability, research in concurrency control is continuing. Some studies are being done for object-oriented systems [SYO1992] while others are dealing with semantics of transactions and weaker form of consistency.