Concurrency control in database systems

The purpose of concurrency control is to ensure that one users work does not inappropriately influence another user's work. In some cases, these measures ensure that a user gets the same result when processing with other users as that person would have received if processing alone. In other cases, it means that the user's work Is Influenced by other users but In an anticipated way. When many transactions take place at the same time, they are called concurrent transactions.

Managing the execution of such transactions is called concurrency intro. As you can imagine, concurrency control is especially important in a multiuse database environment. Deadlock: When dealing with locks two problems can arise, the first of which being deadlock. Deadlock refers to a particular situation where two or more processes are each waiting for another to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource.

Some computers, usually those Intended for the time-sharing and/or real-time markets, are often equipped with a hardware lock, or hard lock, which guarantees exclusive access to processes, forcing serialization. Deadlocks are particularly disconcerting because there is no general solution to avoid them. A fitting analogy of the deadlock problem could be a situation like when you go to unlock your car door and your passenger pulls the handle at the exact same time, leaving the door still locked. If you have ever been in a situation where the passenger is impatient and keeps trying to open the door, It can be very frustrating.

Basically you can get stuck in an endless cycle, and nice both actions cannot be satisfied, deadlock occurs. Livestock: Livestock is a special case of resource starvation. A livestock is similar to a deadlock, except that the states of the processes involved constantly change with regard to one another wile never progressing. The general definition only states that a specific process Is not progressing. For example, the system keeps selecting the same transaction for rollback causing the transaction to never finish executing.

Another livestock situation can come about when the system is deciding which transaction gets a lock and which waits in a conflict situation. An illustration of livestock occurs when numerous people arrive at a four way stop, and are not quite sure who should proceed next. If no one makes a solid decision to go, and all the cars just keep creeping into the intersection afraid that someone else will possibly hit them, then a Basic Timestamp: Basic timestamp is a concurrency control mechanism that eliminates deadlock.

This method doesn't use locks to control concurrency, so it is impossible for deadlock to occur. According to this method a unique timestamp is assigned to each transaction, usually showing when it was started. This effectively allows an age to be assigned to transactions and an order to be assigned. Data items have both a read- timestamp and a write-timestamp. These timestamps are updated each time the data item is read or updated respectively. Problems arise in this system when a transaction tries to read a data item which has been written by a younger transaction.

This is called a late read. This means that the data item has changed since the initial transaction start time and the solution is to roll back the timestamp and acquire a new one. Another problem occurs when a transaction tries to write a ATA item which has been read by a younger transaction. This is called a late write. This means that the data item has been read by another transaction since the start time of the transaction that is altering it. The solution for this problem is the same as for the late read problem. The timestamp must be rolled back and a new one acquired.

Adhering to the rules of the basic timestamp process allows the transactions to be serialized and a chronological schedule of transactions can then be created. Timestamp may not be practical in the case of larger databases with gig levels of transactions. A large amount of storage space would have to be dedicated to storing the timestamps in these cases. Main Insights The requirement for concurrency control arose to ensure correctness when a shared database is updated by multiple transactions concurrently [Gray and Reuters 1992]. PL (Two-phase Locking) The two-phase locking rule simply states that no transaction should request a lock after it releases one of its locks. Alternatively, a transaction should not release a lock until it is certain that it will not request another lock. PL algorithms execute orientations in two phases. Each transaction has a growing phase, where it obtains locks and accesses data items, and a shrinking phase, during which it releases locks. The lock point is the moment when the transaction has achieved all its locks but has not yet started to release any of them.

Thus the lock point determines the end of the growing phase and the beginning of the shrinking phase of a transaction. It has been proven that any history generated by a concurrency control algorithm that obeys the PL rule is serialized [Cesarean et al. , 1976]. Deadlock Management Any locking-based concurrency control algorithm may result in deadlocks, since there is mutual exclusion of access to shared resources (data) and transactions may wait algorithms that require the waiting of transactions (e. G. , strict Timestamp Ordering) may also cause deadlocks.

Therefore, the distributed DB'S requires special procedures to handle them. A deadlock can occur because transactions wait for one another. Informally, a deadlock situation is a set of requests that can never be granted by the concurrency control mechanism. A deadlock is a permanent phenomenon. If one exists in a system, it will not go away unless outside intervention takes place. This outside interference may come from the user, the system operator, or the software system (the operating system or the distributed DB'S).

Pessimistic Concurrency Control Pessimistic Concurrency Control assumes that conflicts will happen Pessimistic Concurrency Control techniques detect conflicts as soon as they occur and resolve them using blocking. Locking Locking is " pessimistic" because it assumes that conflicts will happen. The concept of locking data items is one of the main techniques used for controlling the concurrent execution of transactions. A lock is a variable associated with a data item in the database. Generally there is a lock for each data item in the database.

A lock describes the status of the data item with respect to possible operations that can be applied to that item. It is used for synchronizing the access by concurrent transactions to the database items. - A transaction locks an object before using it - When an object is locked by another transaction, the requesting transaction must wait Disadvantages of locking Lock management overhead. Deadlock detection/resolution. Concurrency is significantly lowered, when congested nodes are locked.

To allow a transaction to abort itself when mistakes occur, locks can't be released until the end of transaction, thus currency is significantly lowered Conflicts are rare. (We might get better performance by not locking, and instead checking for conflicts at commit time.) Pessimistic Locking: This concurrency control strategy involves keeping an entity in a database locked the entire time it exists in the database's memory. This limits or prevents users from altering the data entity that is locked.

There are two types of socks that fall under the category of pessimistic locking: write lock and read lock. With write lock, everyone but the holder of the lock is prevented from reading, updating, or deleting the entity. With read lock, other users can read the entity, but no one except for the lock holder can update or delete it. Timestamp There is a strong relationship among the concurrency control problem, the deadlock management problem, and reliability issues. This is to be expected, since together they are usually called the transaction management problem.

The concurrency management facility is required. If a locking-based algorithm is used, deadlocks will occur, whereas they will not if timestamp is the chosen alternative. Timestamp-Based Concurrency Control Algorithms Unlike locking-based algorithms, timestamp-based concurrency control algorithms do not attempt to maintain serialization by mutual exclusion. Instead, they select, a prior', a serialization order and execute transactions accordingly. A timestamp is a simple identifier that serves to identify each transaction uniquely and is used for ordering.

Uniqueness is only one of the properties of timestamp generation. The second property is monotonic. Two timestamps generated by the same transaction manager should be monotonically increasing. Thus timestamps are values derived from a totally ordered domain. It is this second property that differentiates a timestamp from a transaction identifier. Timestamp-Based Protocols Each transaction is issued a timestamp when it enters the system. If an old transaction It has time-stamp TTS(It), a new transaction TX is assigned time-stamp TTS(TX) such that TTS(It) The protocol manages concurrent execution such that the time-stamps determine the serialization order. In order to assure such behavior, the protocol maintains for each data Q two timestamp values: W-timestamp(Q) is the largest time-stamp of any transaction that executed write(Q) successfully. R- timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Suppose a transaction It issues a read(Q) 1.

If TTS(It) 0 W-timestamp(Q), then It needs to read a value of Q that was already overwritten. Hence, the read operation is back. Rejected, and It is rolled 2. If W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to the maximum of R- timestamp(Q) and TTS(It). Suppose that transaction It issues write(Q). If TTS(It) < R-timestamp(Q), then the value of Q that Ti is producing was needed previously, and the system assumed that that value would never be produced. Hence, the write operation is rejected, and Ti is rolled back.

If TTS(It) < W-timestamp(Q), then Ti is attempting to write an obsolete value of Q. Hence, this write operation is rejected, and Ti is rolled back. Otherwise, the write operation is executed, and W- timestamp(Q) is set to TS(Ti). Another alternative is to make use of transaction timestamps to prioritize transactions and resolve deadlocks by aborting transactions with higher (or lower) priorities. To implement this type of prevention method, the lock manager is modified as follows. If a lock request of a transaction T is denied, the lock manager does not automatically force Ti I to wait.

Instead, it applies a prevention test to the requesting transaction and the transaction that currently holds the lock (say TX). If the test is aborted. Rollback A rollback is the operation of restoring a database to a previous state by canceling a pacific transaction or transaction set. Rollbacks are either performed automatically by database systems or manually by users. When a database user changes a data field but has not yet saved the change, the data is stored in a temporary state or transaction log. Users querying the unsaved data see the unchanged values.

The action of saving the data is a commit; this allows subsequent queries for this data to show the new values. However, a user may decide not to save the data. Under this condition, a rollback command manipulates the data to discard any changes made by the user, and does o without communicating this to the user. Thus, a rollback occurs when a user begins changing data, realizes the wrong record is being updated and then cancels the operation to undo any pending changes. Rollbacks also may be issued automatically after a server or database crash, e. . After a sudden power loss. When the database restarts, all logged transactions are reviewed; then all pending transactions are rolled back, allowing users to reenter and save appropriate changes. A transaction always terminates, even when there are failures. If the transaction can complete its task successfully, we say that the orientation commits. If, on the other hand, a transaction stops without completing its task, we say that it aborts. Transactions may abort for a number of reasons, which are discussed in the upcoming chapters.

In our example, a transaction aborts itself because of a condition that would prevent it from completing its task successfully. Additionally, the DB'S may abort a transaction due to, for example, deadlocks or other conditions. When a transaction is aborted, its execution is stopped and all of its already executed actions are undone by returning the database to the state before their execution. This is also known as rollback. Optimistic Concurrency Control Optimistic Concurrency Control assumes that conflicts between transactions are rare.

Does not require locking Transaction executed without restrictions Check for conflicts Just before commit Optimistic Concurrency Control (Terminology used) Readers(It): Set of objects read by Transaction It. Writes(It): Set of objects modified by Transaction It. Optimistic algorithms, on the other hand, delay the validation phase until Just before the write phase. Thus an operation submitted to an optimistic scheduler is never delayed. The read, compute, and write operations of each transaction are processed updates on local copies of data items.

The validation phase consists of checking if these updates would maintain the consistency of the database. If the answer is affirmative, the changes are made global (I. E. , written into the actual database). Otherwise, the transaction is aborted and has to restart. Types of locking Technique The two Phase locking protocol Time Stamping Protocol Validation Based protocol The two-phase locking protocol is used to ensure the serialization in Database. This protocol is implemented in two phase: Growing Phase -In this phase we put read or write lock based on need on the data.

In this phase we does not release any lock. Shrinking Phase -this phase is Just reverse of growing phase. In this phase we release read and write lock but doesn't put any lock on data. In Time stamping Protocol we select sequence of transaction in advance by using Time stamping concept. We add a special variable time stamp (a unique, fixed non decreasing) to each transaction in system. This number can be system clock value. When a new transaction entered in the system, current value of cook is assigned to orientation as time stamp value.

Value of time stamp is incremented every time after a new transaction interred in the system. Validation Based protocol Two-phase locking protocol and Time stamping Protocol are slow in working because they worked in two steps, so we use validation based protocol which is faster than Two-phase locking protocol and Time stamping Protocol. In Validation based protocol, it doesn't update entire data base in one step. It keeps local copies of all updates during transaction execution. It works in three steps Read Phase

In this step transaction is activated and it reads last committed value from DB and put these value in local variables. All updates implemented in these local variable of database. Validation Phase check consistency of data base after modification performed in database. Write Phase If validation phase say that data base is in consistent state then all update made by transaction which are in local variable are applied in database for permanent storage. If validation phase say that database is not consistent or it violate serialization then it discard/rollback all updates and it restart transactions.