CONCURRENCY CONTROL

CONCURRENCY CONTROL

- Concurrency control is the activity of coordinating concurrent accesses to a database in a multi-user system.
- Concurrency control allows users to access a database in a multi-programmed fashion while preserving the consistency of the data.
- The main technical difficulty in achieving this goal is the necessity to prevent database updates performed by one user from interfering with database retrievals and updates performed by other users.
- The concurrency control problem is exacerbated in a distributed DBMS (DDBMS), because in a distributed system users may access data stored at many different sites, and a concurrency control mechanism at one site cannot be aware of transactions at other sites instantaneously.
- The replication of data items in a distributed database adds extra complexity to the concurrency control mechanism.

CONCURRENCY CONTROL

- The primary goal of the concurrency control mechanism in a DDBMS is to ensure that the consistency of the data items is preserved and each atomic action will be completed in a finite time if the system has not failed.
- A good concurrency control mechanism for a DDBMS has the following objectives:
  - It must be resilient to site and communication failures.
  - It should permit parallel execution of transactions to achieve maximum concurrency, thus satisfying performance requirements.
  - Its computational methods and storage mechanisms should be modest to minimize overhead.
  - It should perform satisfactorily in a network environment taking into consideration that it involves significant communication delay.
  - It must impose few constraints on the structure of atomic actions of transactions

CONCURRENCY CONTROL

- Concurrency control is the activity of coordinating concurrent accesses to a database in a multi-user system.
- The different anomalies that can arise owing to con-current accesses of data in a multi-user centralized database environment as well as in a distribute ddatabase environment are:
  - **lost update anomaly,**
  - **uncommitted dependency or dirty read anomaly,**
  - **inconsistent analysis anomaly,**
  - **non-repeatable or fuzzy read anomaly**
  - **phantom read anomaly**

CONCURRENCY CONTROL

- **Lost update anomaly**
- This anomaly can occur when an apparently successful completed update operation made by one user (transaction) is overridden by another user (transaction).
- **Uncommitted dependency or dirty read anomaly**
- This problem occurs when one transaction allows other transactions to read its data before it has committed and then decides to abort.
- **Inconsistent analysis anomaly**
- 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.

CONCURRENCY CONTROL

- **Non-repeatable or fuzzy read anomaly**
- This occurs when one transaction reads the value of a data item that is subsequently modified or deleted by another transaction.
- If the first transaction attempts to re-read the data item either that data item may not be found or it may read a different value.
- **Phantom read anomaly**
- This anomaly occurs when a transaction performing some operation on the database based on a selection predicate, another transaction inserts new tuples satisfying that predicate into the same database. This is known as **phantom read**.

CONCURRENCY CONTROL

- **Non-repeatable or fuzzy read anomaly**
- This occurs when one transaction reads the value of a data item that is subsequently modified or deleted by another transaction.
- If the first transaction attempts to re-read the data item either that data item may not be found or it may read a different value.
- **Phantom read anomaly**
- This anomaly occurs when a transaction performing some operation on the database based on a selection predicate, another transaction inserts new tuples satisfying that predicate into the same database. This is known as **phantom read**.

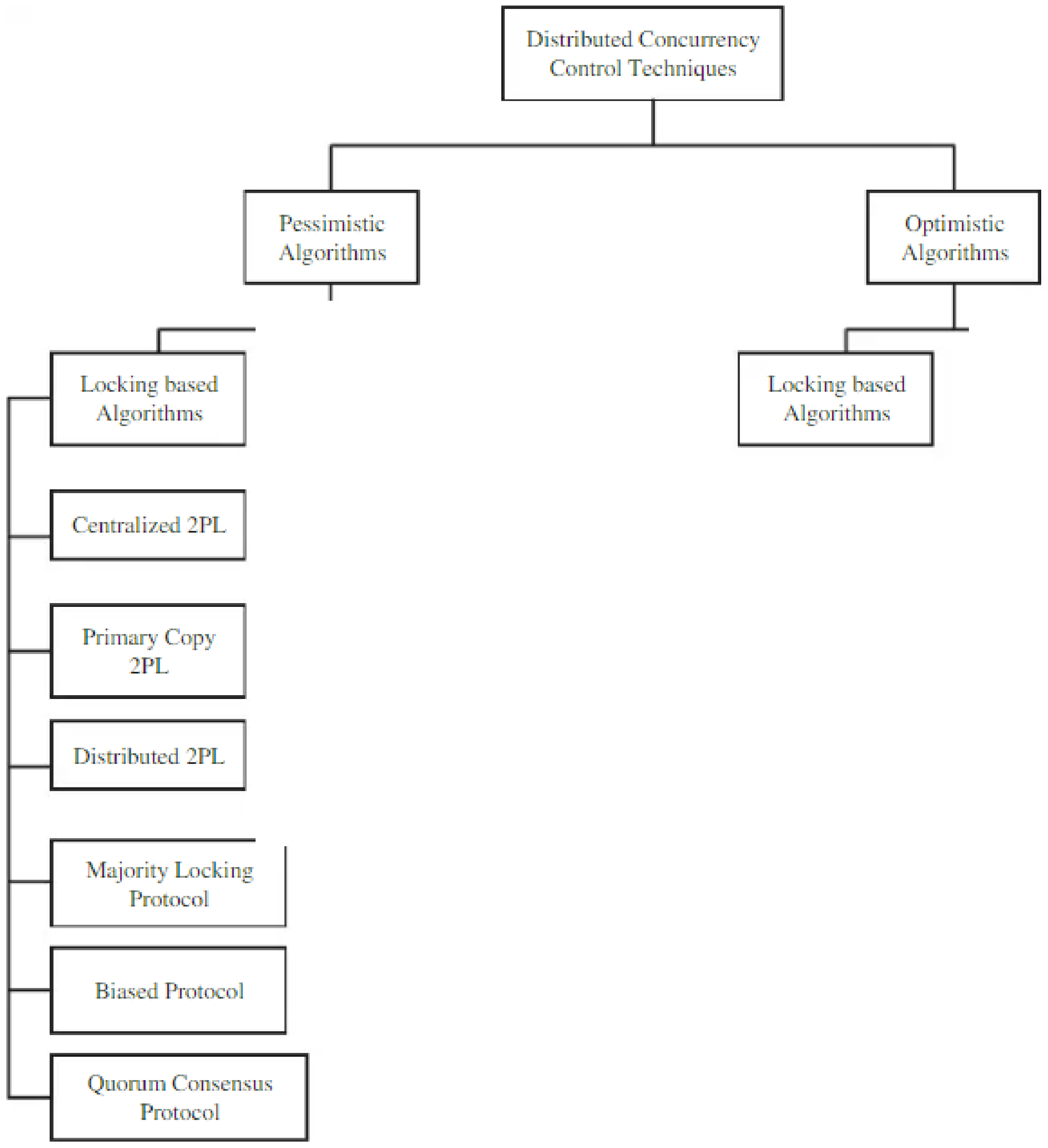
CONCURRENCY CONTROL TECHNIQUES

- Concurrency control in a distributed system involves the activity of controlling the relative order of conflicting operations and thereby ensuring database consistency.
- The most obvious classification is done based on the synchronization approaches that are used in concurrency control techniques to ensure distributed serializability.
- Thus, concurrency control techniques are divided into two broad categories based on the synchronization techniques: **pessimistic concurrency control mechanisms** and **optimistic concurrency control mechanisms.**
- Pessimistic algorithms synchronize the concurrent execution of transactions early in their execution life cycle, whereas optimistic algorithms delay the synchronization of transactions until transactions are near to their completion.
- Optimistic concurrency control algorithms are based on the assumption that a conflict is rare and it is more efficient to allow transactions to proceed without delays.

CONCURRENCY CONTROL TECHNIQUES

- When a transaction wishes to commit, a check is performed to determine whether a conflict has occurred or not.
- Therefore, optimistic concurrency control algorithms have the desirable properties of being non- blocking and deadlock free.
- In the case of pessimistic concurrency control algorithms, the synchronization of transactions is achieved earlier before the starting of transaction executions.
- The main drawback of pessimistic concurrency control algorithms is that they are neither deadlock-free nor non-blocking algorithms.
- The pessimistic concurrency control algorithms are further classified into locking-based algorithms, timestamp-based algorithms and hybrid algorithms.
- .Similarly, the optimistic concurrency control algorithms are also classified into locking-based algorithms and timestamp-based algorithms.

CONCURRENCY CONTROL TECHNIQUES



TWO-PHASE LOCKING

- A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.
- Generally, there is one lock for each data item in the database.
- Locks are used as a means of synchronizing the access by concurrent transactions to the database items.
- We should allow several transactions to access the same item X if they all access X for reading purposes only.
- This is because read operations on the same item by different transactions are not conflicting.
- However, if a transaction is to write an item X, it must have exclusive access to X.
- For this purpose, a different type of lock called a multiple-mode lock is used.
- In this scheme—called shared/exclusive or read/write locks—there are three locking operations: read_lock(X), write_lock(X), and unlock(X).
- A lock associated with an item X, LOCK(X), now has three possible states: read-locked, write-locked, or unlocked.
- A read-locked item is also called share-locked because other transactions are allowed to read the item, whereas a write-locked item is called exclusive-locked because a single transaction exclusively holds the lock on the item.

TWO-PHASE LOCKING

- When we use the shared/exclusive locking scheme, the system must enforce the following rules:
  1. A transaction T must issue the operation read_lock(X) or write_lock(X) before any read_item(X) operation is performed in T.
  2. A transaction T must issue the operation write_lock(X) before any write_item(X) operation is performed in T.
  3. A transaction T must issue the operation unlock(X) after all read_item(X) and write_item(X) operations are completed in T.3
  4. A transaction T will not issue a read_lock(X) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.

TWO-PHASE LOCKING

Initial values: X=20, Y=30

Result serial schedule T_1 followed by T_2: X=50, Y=80

Result of serial schedule T_2 followed by T_1: X=70, Y=50

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
$X := X + Y$;	$Y := X + Y$;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

a) Two transactions T1 and T2.

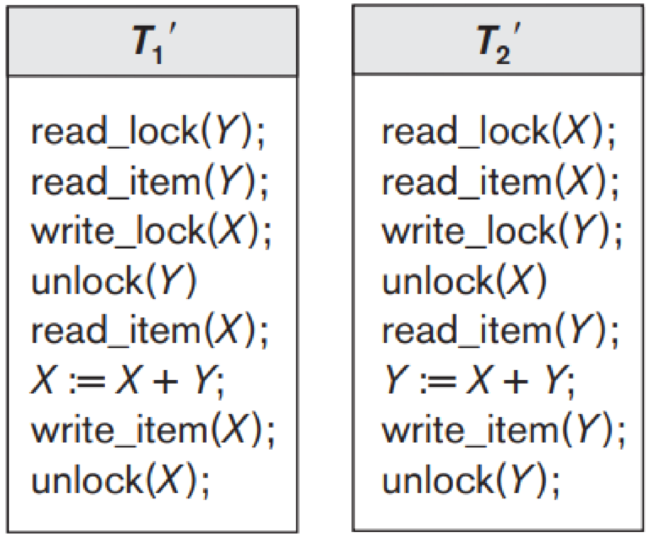
T_1	T_2
read_lock(Y);	
read_item(Y);	
unlock(Y);	
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);
write_lock(X);	
read_item(X);	
$X := X + Y$;	
write_item(X);	
unlock(X);	

b) A nonserializable schedule S that uses locks.

Result of schedule S: X=50, Y=50 (nonserializable)

TWO-PHASE LOCKING

- Transactions T1 and T2 do not follow the two-phase locking protocol because the write_lock(X) operation follows the unlock(Y) operation in T1, and similarly the write_lock(Y) operation follows the unlock(X) operation in T2.
- If we enforce two-phase locking, the transactions can be rewritten as T1' and T2', as shown below.
- Now, the schedule shown in Figure b is not permitted for T1' and T2' (with their modified order of locking and unlocking operations) under the rules because T1' will issue its write_lock(X) before it unlocks item Y; consequently, when T2' issues its read_lock(X), it is forced to wait until T1' releases the lock by issuing an unlock (X) in the schedule.



T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y)	unlock(X)
read_item(X);	read_item(Y);
$X := X + Y$;	$Y := X + Y$;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

TWO-PHASE LOCKING

- It can be proved that, if every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedules.
- The locking protocol, by enforcing two-phase locking rules, also enforces serializability.
- Two-phase locking may limit the amount of concurrency that can occur in a schedule because a transaction T may not be able to release an item X after it is through using it if T must lock an additional item Y later; or conversely, T must lock the additional item Y before it needs it so that it can release X.
- Hence, X must remain locked by T until all items that the transaction needs to read or write have been locked; only then can X be released by T.
- Meanwhile, another transaction seeking to access X may be forced to wait, even though T is done with X; conversely, if Y is locked earlier than it is needed, another transaction seeking to access Y is forced to wait even though T is not using Y yet.

TWO-PHASE LOCKING

- There are a number of variations of two-phase locking (2PL).
- A variation known as conservative 2PL (or static 2PL) requires a transaction to lock all the items it accesses before the transaction begins execution, by pre-declaring its read-set and write-set.
- The read-set of a transaction is the set of all items that the transaction reads, and the write-set is the set of all items that it writes.
- If any of the pre-declared items needed cannot be locked, the transaction does not lock any item; instead, it waits until all the items are available for locking.
- Conservative 2PL is a deadlock-free protocol.
- However, it is difficult to use in practice because of the need to pre-declare the read-set and write-set, which is not possible in many situations.
- The most popular variation of 2PL is strict 2PL, which guarantees strict schedules.

TWO-PHASE LOCKING

- In this variation, a transaction T does not release any of its exclusive (write) locks until after it commits or aborts.
- Hence, no other transaction can read or write an item that is written by T unless T has committed, leading to a strict schedule for recoverability.
- Strict 2PL is not deadlock-free.
- A more restrictive variation of strict 2PL is rigorous 2PL, which also guarantees strict schedules.
- In this variation, a transaction T does not release any of its locks (exclusive or shared) until after it commits or aborts, and so it is easier to implement than strict 2PL.
- The conservative 2PL must lock all its items before it starts, so once the transaction starts it is in its shrinking phase;
- The Rigorous 2PL does not unlock any of its items until after it terminates (by committing or aborting), so the transaction is in its expanding phase until it ends.

CONCURRENCY CONTROL TECHNIQUES

- To ensure serializability of concurrent transactions locking methods are the most widely used approach.
- In this approach, before accessing any data item, a transaction must acquire a lock on that data item.
- When a transaction acquires a lock on a particular data item, the lock prevents another transaction from modifying that data item.
- Based on the facts that read-read operation by two different transactions is non-conflicting and the main objective of the locking-based concurrency control techniques is to synchronize the conflicting operations of conflicting transactions, there are two types of locking modes: read lock (also called shared lock) and write lock (also called exclusive lock).
- If a transaction obtains a read lock on a data item, it can read, but cannot update that data item.
- On the other hand, if a transaction obtains a write lock on a data item, it can read as well as update that data item.

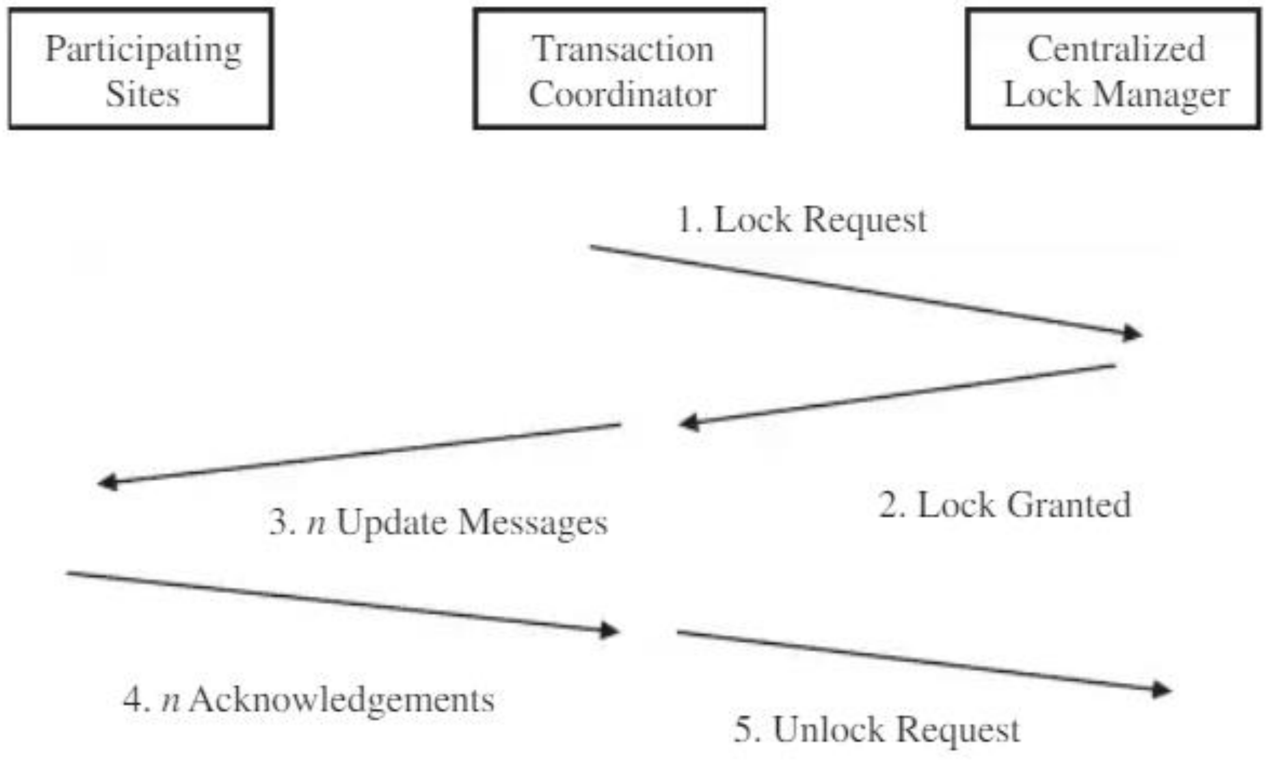
CENTRALIZED 2PL

- In the centralized 2PL method, the lock manager or scheduler is a central component, and hence a single site is responsible for managing all activities of the lock manager for the entire distributed system.
- Before accessing any data item at any site, appropriate locks must be obtained from the central lock manager.
- If a global transaction Ti is initiated at site Si of the distributed system, then the centralized 2PL for that global transaction should be implemented in the following way.
- The transaction manager at the site Si (called transaction coordinator) partitions the global transaction into several sub-transactions using the information stored in the global system catalog.
- Thus, the transaction coordinator is responsible for maintaining consistency of data.
- If the data items are replicated, then it is also the responsibility of the transaction coordinator to ensure that all replicas of the data item are updated.

CENTRALIZED 2PL

- Therefore, for a write operation, the transaction coordinator requests exclusive locks on all replicas of the data item that are stored at different sites.
- However, for a read operation, the transaction coordinator can select any one of the replicas of the data item that are available, preferably at its own site.
- The local transaction managers of the participating sites request and release locks from/to the centralized lock manager following the normal rules of the 2PL protocol.
- Participating sites are those sites that are involved in the execution of the global transaction (sub-transactions).
- After receiving the lock request, the centralized lock manager checks whether that lock request is compatible with the existing locking status or not.
- If it is compatible, the lock manager sends a message to the corresponding site that the lock has been granted; otherwise, the lock request is put in a queue until it can be granted.

CENTRALIZED 2PL



Communicating Messages in Centralized 2PL Method

CENTRALIZED 2PL

- The main advantage of the centralized 2PL method in DDBMSs is that it is a straightforward extension of the 2PL protocol for centralized DBMSs; thus, it is less complicated and implementation is relatively easy.
- The deadlock detection can be handled easily, because the centralized lock manager maintains all the locking information.
- Hence, the communication cost also is relatively low.
- In the case of DDBMSs, the disadvantages of implementing the centralized 2PL method are system bottlenecks and lower reliability.
- For example, as all lock requests go to the centralized lock manager, that central site may become a bottleneck to the system.
- Similarly, the failure of the central site causes a major failure of the system; thus, it is less reliable.
- This method also hampers the local autonomy.

PRIMARY COPY 2PL

- Primary copy 2PL method is a straightforward extension of the centralized 2PL method.
- Here the responsibilities of the centralized lock manager are distributed among a number of sites of the distributed system to overcome the disadvantages of the centralized 2PL method.
- Thus, in the primary copy 2PL method lock managers are implemented at several sites.
- Each lock manager is responsible for managing locks for a given set of data items.
- For replicated data items, one copy is chosen as the primary copy, and the other copies are called slave copies.
- The choice of the site as the primary site of a data item is flexible.
- However, it is not necessary that the primary site hold the primary copy of data items to manage the locks.

PRIMARY COPY 2PL

- Whenever a transaction is initiated at a site, the transaction coordinator will determine where the primary copy is located, and then it will send lock requests to the appropriate lock manager.
- If the transaction requires an update operation of a data item, it is necessary to exclusively lock the primary copy of that data item.
- Once the primary copy of a data item is updated, the change is propagated to the slave copies immediately to prevent other transactions reading the old value of the data item.
- However, it is not mandatory to carry out the updates on all copies of the data item as an atomic operation.
- The primary copy 2PL method ensures that the primary copy is always updated.

PRIMARY COPY 2PL

- The main advantage of primary copy 2PL method is that it overcomes the bottlenecks of centralized 2PL approach and also reliability increases.
- This approach incurs lower communication costs because less amount of remote locking is required.
- The major disadvantage of this method is that it is only suitable for those systems where the data items are selectively replicated, updates are infrequent and sites do not always require the latest version of the data.
- In this approach, deadlock handling is more complex, as the locking information is distributed among multiple lock managers.
- In the primary copy 2PL method there is still a degree of centralization in the system, as the primary copy is only handled by one site (primary site).

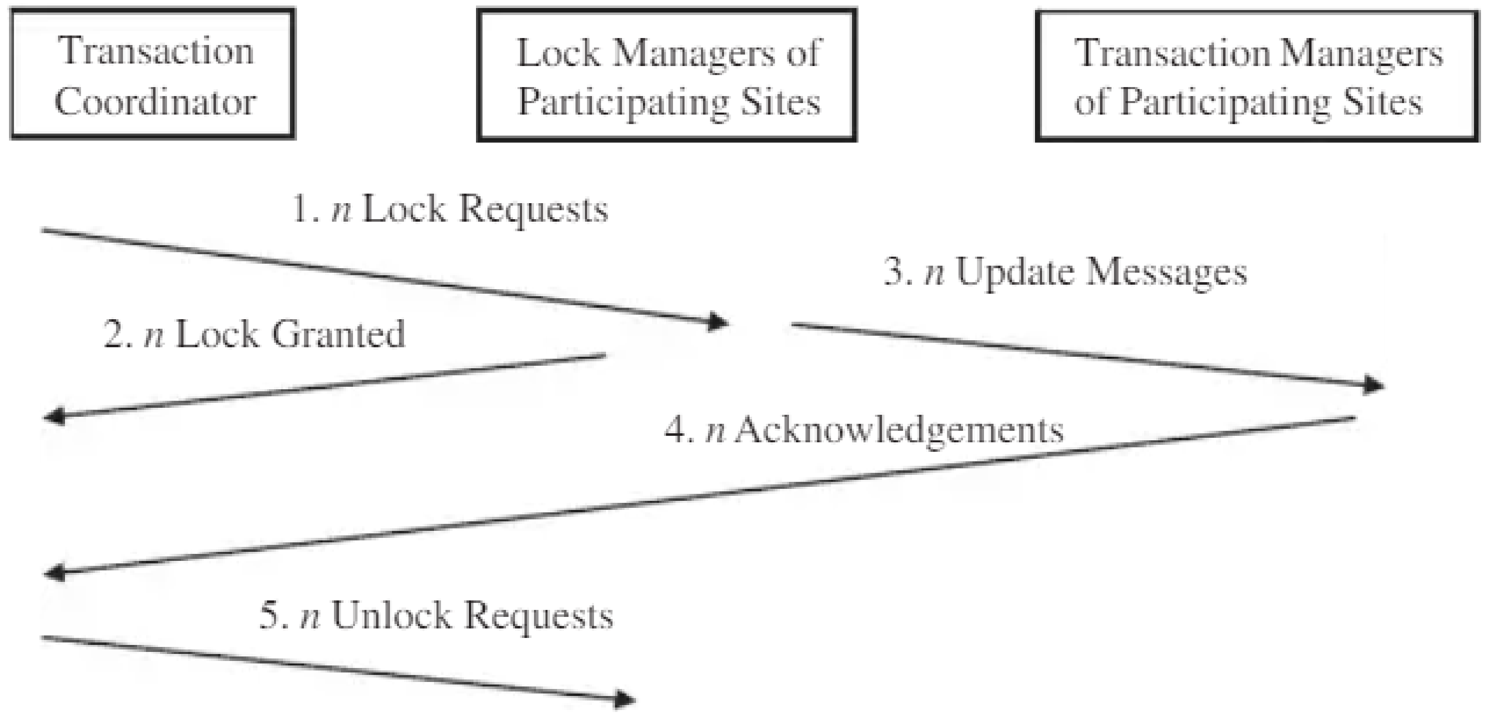
DISTRIBUTED 2PL

- Distributed 2PL method implements the lock manager at each site of a distributed system.
- Thus, the lock manager at each site is responsible for managing locks on data items that are stored locally.
- If the database is not replicated, then distributed 2PL becomes the same as primary copy 2PL.
- If the database is replicated, then distributed 2PL implements a read-one-write-all (ROWA) replica control protocol.
- In ROWA replica control protocol, any copy of a replicated data item can be used for a read operation, but for a write operation, all copies of the replicated data item must be exclusively locked before performing the update operation.

DISTRIBUTED 2PL

- In distributed 2PL, when a global transaction is initiated at a particular site, the transaction coordinator (the transaction manager of that site is called transaction coordinator) sends lock requests to lock managers of all participating sites.
- In response to the lock requests the lock manager at each participating site can send a lock granted message to the transaction coordinator.
- However, the transaction coordinator does not wait for the lock granted message in some implementations of the distributed 2PL method.
- The operations that are to be performed at a participating site are passed by the lock manager of that site to the corresponding transaction manager instead of the transaction coordinator.
- At the end of the operation the transaction manager at each participating site can send the corresponding message to the transaction coordinator.
- In an alternative approach, the transaction manager at a participating site can also pass the "end of operation" message to its own lock manager, who can then release the locks and inform the transaction coordinator.

DISTRIBUTED 2PL



Communicating Messages in Distributed 2PL Method

MAJORITY LOCKING PROTOCOL

- Majority locking protocol was designed to overcome the disadvantage of distributed 2PL method that all replicas of a data item must be locked before an update operation.
- In this protocol, a lock manager is implemented at each site to manage locks for all data items stored at that site locally.
- The majority locking protocol in a DDBMS works in the following way.
- When a transaction requires to read or write a data item that is replicated at *m* sites, the transaction coordinator must send a lock request to more than half of them sites where the data item is stored.
- Each lock manager determines whether the lock can be granted immediately or not.
- If the lock request is not compatible with the existing lock status, the response is delayed until the request is granted.

MAJORITY LOCKING PROTOCOL

- The transaction coordinator waits for a certain timeout period to receive lock granted messages from the lock mangers, and if the response is not received within that time period, it cancels its request and informs all sites about the cancellation of the lock request.
- The transaction cannot proceed until the transaction coordinator obtains locks on a majority of copies of the data item.
- In majority locking protocol, any number of transactions can simultaneously hold a shared lock on a majority of copies of a data item, but only one transaction can acquire an exclusive lock on a majority of copies.
- However, the execution of a transaction can proceed in the same manner as in distributed 2PL method after obtaining locks on a majority of copies of the data items that are required by the transaction.
- This protocol is an extension of the distributed 2PL method, and hence it avoids the bottlenecks of centralized 2PL method.
- The main disadvantage of this protocol is that it is more complicated and hence the deadlock handling also is more complex.