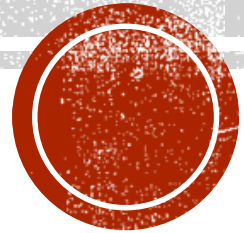
TRANSACTION MANAGEMENT



TRANSACTION MANAGEMENT

- Transaction management in a database system deals with the problems of maintaining consistency of data in spite of system failures and concurrent accesses to data.
- A **transaction** consists of a set of operations that perform a single logical unit of work in a database environment.
- If a transaction is completed successfully, then the database moves from one consistent state to another.
- This consistency must be ensured irrespective of whether transactions are successfully executed simultaneously or there are failures during execution.

TRANSACTION MANAGEMENT

Objectives of Distributed Transaction Management:

- The responsibilities of the transaction manager in a distributed DBMS is same as those of a corresponding subsystem in a centralized database system, but its task is complicated owing to fragmentation, replication and the overall distributed nature of the database.
- The consistency of the database should not be violated because of transaction executions.
- Therefore, the primary objective of a transaction manger is to execute transactions in an efficient way in order to preserve the ACID properties of transactions (local as well as global transactions) irrespective of concurrent execution of transactions and system failures.
- At the same time, the efficiency of transaction executions should be improved to meet the performance criteria.
- In addition, a good transaction management scheme must consider the following issues.

TRANSACTION MANAGEMENT

1. **CPU and main memory utilization should be improved**

- Most of the typical database applications spend much of their time waiting for I/O operations rather than on computations.
- In a large system, the concurrent execution of these I/O bound applications can turn into a bottleneck in main memory or in CPU time utilization.
- To alleviate this problem, that is, to improve CPU and main memory utilization, a transaction manager should adopt specialized techniques to deal with specific database applications.
- This aspect is common to both centralized and distributed DBMSs.

TRANSACTION MANAGEMENT

2. **Response time should be minimized**

- To improve the performance of transaction executions, the response time of each individual transaction must be considered and should be minimized.
- In the case of distributed applications, it is very difficult to achieve an acceptable response time owing to the additional time required to communicate between different sites.

TRANSACTION MANAGEMENT

3. **Availability should be maximized**

- Although the availability in a distributed system is better than that in a centralized system, it must be maximized for transaction recovery and concurrency control in distributed databases.
- The algorithms implemented by the distributed transaction manager must not block the execution of those transactions that do not strictly need to access a site that is not operational.

TRANSACTION MANAGEMENT

4. **Communication cost should be minimized**

- In a distributed system an additional communication cost is incurred for each distributed or global application, because a number of message transfers are required between sites to control the execution of a global application.
- These messages are not only used to transfer data, but are required to control the execution of the application.
- Preventative measures should be adopted by the transaction manager to minimize the communication cost.

TRANSACTION MANAGEMENT MODEL

- In a distributed system, transactions are classified into two different categories:
  - local transactions and
  - global transactions (or distributed transactions).
- If the data requirement of a transaction can be fulfilled from the local site, it is called a local transaction.
- Local transactions access data only from local sites.
- Global transactions access data from remote sites or multiple sites.
- The transaction management in a distributed DBMS is more complicated than in a centralized DBMS, as the distributed DBMS must ensure the atomicity of the global transaction as well as of each component sub-transaction executed at the local sites.
- This complicated task is performed by four high-level interconnected modules of the DBMS.

TRANSACTION MANAGEMENT MODEL

- These are:
  - transaction manager,
  - concurrency control manager,
  - recovery manager and
  - buffer manager.
- The **transaction manager** coordinates transactions on behalf of application programs by communicating with the scheduler and implements a particular strategy for concurrency control.
- The responsibility of the **concurrency control manager** is to maximize concurrency, without allowing concurrently executing transactions to interfere with one another, and thereby maintain the consistency of the database as well as the isolation property of the transactions.
- The **recovery manager** preserves the database in a consistent state in case of failures.
- The **buffer manager** manages the efficient transfer of data between disk storage and main memory.

TRANSACTION MANAGEMENT MODEL

- In a distributed DBMS, all these modules exist in each local DBMS.
- In addition, a global transaction manager or transaction coordinator is required at each site to control the execution of global transactions as well as of local transactions initiated at that site.
- An abstract model of transaction management at each site of a distributed system consists of two differ-ent submodules: the transaction manager and the transaction coordinator

TRANSACTION MANAGEMENT MODEL

Transaction manager

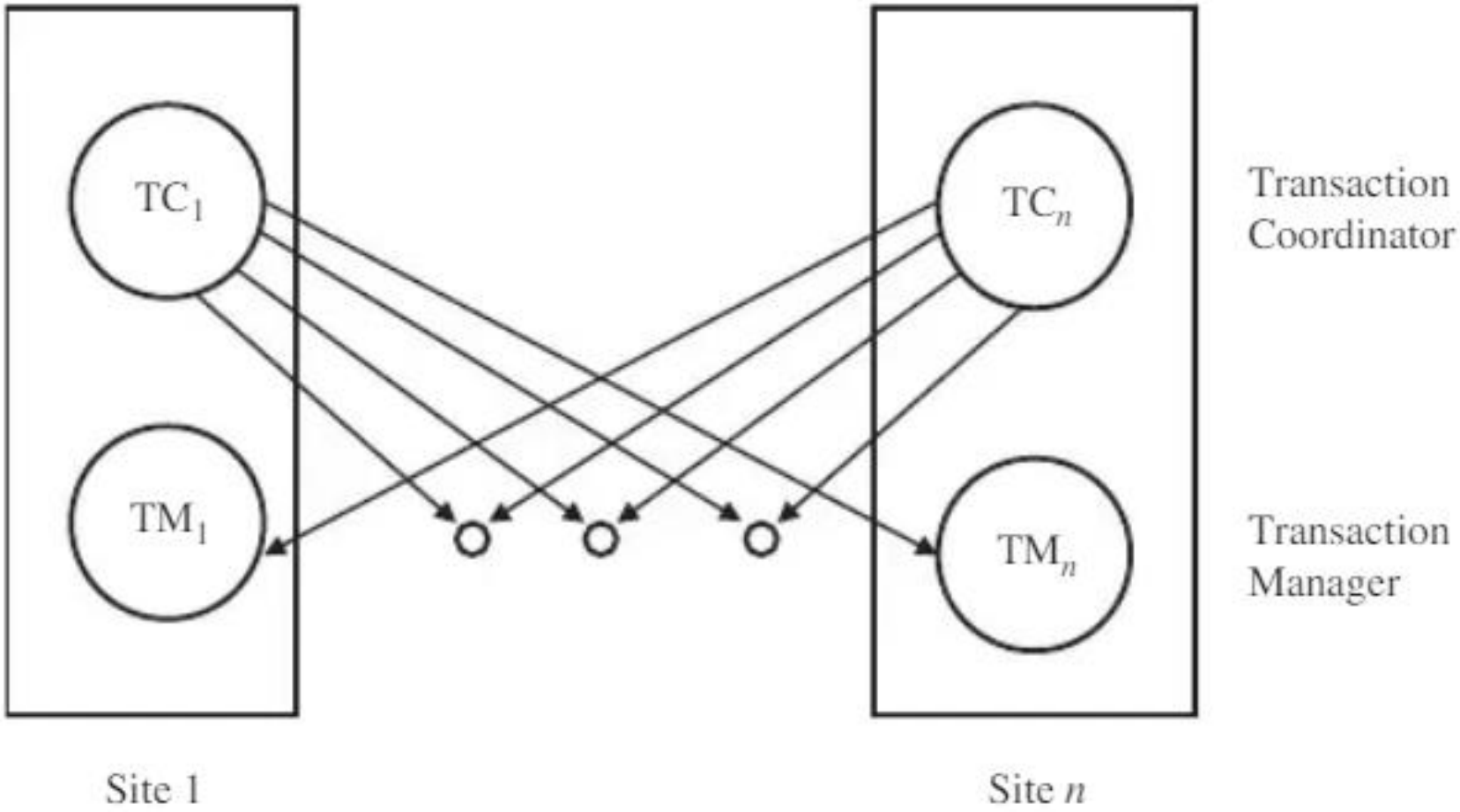
- The transaction manager at each site manages the execution of the transactions that access data stored at that local site.
- Each such transaction may be a local transaction or part of a global transaction.
- The structure of the transaction manager is similar to that of its counterpart in a centralized system, and it is responsible for the following:
  - Maintaining a log for recovery purposes in case of failures.
  - Implementing an appropriate concurrency control mechanism to coordinate the concurrent execution of transactions executing at that local site.

TRANSACTION MANAGEMENT MODEL

Transaction coordinator

- The transaction coordinator at each site in a distributed system coordinates the execution of both local and global transactions initiated at that site.
- This component or module is not required in centralized DBMSs, as there is only one site in a centralized system.
- The transaction coordinator at each site performs the following tasks to coordinate the execution of transactions initiated at that local site.
- It starts the execution of the transactions initiated at that site.
- It breaks the transactions into a number of sub-transactions and distributes these sub-transactions to the appropriate sites for execution.
- It coordinates the termination of the transactions, which may result in the transactions being committed at all sites or aborted at all sites.

TRANSACTION MANAGEMENT MODEL



TRANSACTION MANAGEMENT MODEL

- A transaction is considered as a part of an application.
- When a user requests for the execution of an application (may be local or global), the application issues a **begin_transaction primitive.**
- All actions that are performed by the application after that are to be considered part of the same transaction until a **commit** or an **abort** primitive is issued.
- In some systems, all these primitives are implicitly associated with the application; thus, it is not necessary to define all these primitives explicitly.
- To execute a global application generated at site S1, the transaction coordinator of site S1 initiates the transaction and breaks the transaction into a number of sub transactions.
- It then involves multiple sites by consulting the global system catalog for parallel execution of these sub transactions at different sites.
- When these sub transactions are distributed over multiple local sites and perform some tasks on behalf of the application, they are called agents.

TRANSACTION MANAGEMENT MODEL

- Agents reside at multiple sites and communicate with each other via message passing.
- The transaction manager at a local site maintains the log when a local transaction or part of a global transaction is executed at that local site.
- It also controls the concurrent execution of transactions executing at that site.
- The results retrieved from the parallel execution of these subtransactions at multiple sites are integrated by the transaction coordinator of site S1, and finally the transaction is terminated.

TRANSACTION MANAGEMENT MODEL

Two-Phase Commit Protocol

- Consider we have multiple distributed databases which are operated from different servers(sites) let's say $S_1, S_2, S_3, \ldots S_n$.
- Where every S_i made to maintains a separate log record of all corresponding activities and the transition T has also been divided into the sub transactions T1, T2, T3, …., T_n and each T_i are assigned to S_i.
- This all maintains by a separate transaction manager at each S_i.
- Anyone site can act as a Coordinator.
- Some points to be considered regarding this protocol:

a) In a two-phase commit, we assume that each site logs actions at that site, but there is no global log.

b) The coordinator(Ci), plays a vital role in doing confirmation whether the distributed transaction would abort or commit.

c) In this protocol messages are made to send between the coordinator(Ci) and the other sites. As each message is sent, its logs are noted at each sending site, to aid in recovery should it be necessary.

TRANSACTION MANAGEMENT MODEL

- The two phases of this protocol are as follow:
- **Phase-1–**

a) Firstly, the coordinator(Ci) places a log record <Prepare T> on the log record at its site.

b) Then, the coordinator(Ci) sends a Prepare T message to all the sites where the transaction(T) executed.

c) Transaction manager at each site on receiving this message Prepare T decides whether to commit or abort its component(portion) of T. The site can delay if the component has not yet completed its activity, but must eventually send a response.

d) If the site doesn't want to commit, so it must write on log record <no T>, and local Transaction manager sends a message abort T to Ci.

e) If the site wants to commit, it must write on log record <ready T>, and local Transaction manager sends a message ready T to Ci. Once the ready T message at Ci is sent nothing can prevent it to commit its portion of transaction T except Coordinator(Ci).

TRANSACTION MANAGEMENT MODEL

- **Phase- 2–**
- The Second phase started as the response abort T or commit T receives by the coordinator(Ci) from all the sites that are collaboratively executing the transaction T.
- However, it is possible that some site fails to respond; it may be down, or it has been disconnected by the network.
- In that case, after a suitable timeout period will be given, after that time it will treat the site as if it had sent abort T.
- The fate of the transaction depends upon the following points:

a) If the coordinator receives ready T from all the participating sites of T, then it decides to commit T. Then, the coordinator writes on its site log record <Commit T> and sends a message commit T to all the sites involved in T.

b) If a site receives a commit T message, it commits the component of T at that site, and write it in log records <Commit T>.

c) If a site receives the message abort T, it aborts T and writes the log record <Abort T>.

d) However, if the coordinator has received abort T from one or more sites, it logs <Abort T> at its site and then sends abort T messages to all sites involved in transaction T.

TRANSACTION MANAGEMENT MODEL

- Disadvantages:

a) The major disadvantage of the Two-phase commit protocol is faced when the Coordinator site failure may result in blocking, so a decision either to commit or abort Transaction(T) may have to be postponed until coordinator recovers.

b) **Blocking Problem:**
   - If a Transaction(T) holds locks on data-items of active sites, but amid the execution, if the coordinator fails and the active sites keep no additional log-record except <read T> like <abort> or <commit>.
   - So, it becomes impossible to determine what decision has been made(whether to <commit> /<abort>).
   - So, In that case, the final decision is delayed until the Coordinator is restored or fixed.
   - In some cases, this may take a day or long hours to restore and during this time period, the locked data items remain inaccessible for other transactions(T_i).
   - This problem is known as Blocking Problem.

TRANSACTION MANAGEMENT MODEL

Three-Phase Commit Protocol

- The problems of 2PC are solved by the three-phase commit (3PC) protocol, which essentially divides the second commit phase into two sub phases called prepare-to commit and commit.
- The prepare-to-commit phase is used to communicate the result of the vote phase to all participants.
- If all participants vote yes, then the coordinator instructs them to move into the prepare-to-commit state.
- The commit sub phase is identical to its two-phase counterpart.
- Now, if the coordinator crashes during this subphase, another participant can see the transaction through to completion.
- It can simply ask a crashed participant if it received a prepare-to-commit message.
- If it did not, then it safely assumes to abort.
- Thus the state of the protocol can be recovered irrespective of which participant crashes.
- Also, by limiting the time required for a transaction to commit or abort to a maximum time-out period, the protocol ensures that a transaction attempting to commit via 3PC releases locks on time-out.

TRANSACTION MANAGEMENT MODEL

Three-Phase Commit Protocol

- The main idea is to limit the wait time for participants who have committed and are waiting for a global commit or abort from the coordinator.
- When a participant receives a pre-commit message, it knows that the rest of the participants have voted to commit.
- If a pre-commit message has not been received, then the participant will abort and release all locks.