Transaction Management in Distributed Systems

By

Suparna Dutta

Transaction Concept

- A transaction is a unit of consistent and reliable computation.
- Thus, intuitively, a transaction takes a database, performs an action on it, and generates a new version of the database, causing a state transition.
- This is similar to what a query does, except that if the database was consistent before the execution of the transaction, we can now guarantee that it will be consistent at the end of its execution.
- Regardless of the fact that

  (1) the transaction may have been executed concurrently with others,

  (2) failures may have occurred during its execution.
- A transaction always terminates, even when there are failures. If the transaction can complete its task successfully, we say that the transaction commits. If, on the other hand, a transaction stops without completing its task, we say that it aborts.

Transaction Concept

Consider the following SQL query for increasing by 10% the budget of the CAD/CAM project that we discussed

```
UPDATE PROJ
SET BUDGET = BUDGET*1.1
WHERE PNAME= "CAD/CAM"
```

This query can be specified, using the embedded SQL notation, as a transaction by giving it a name (e.g., BUDGET UPDATE) and declaring it as follows:

```
Begin_transaction BUDGET_UPDATE
begin
  EXEC SQL          UPDATE    PROJ
            SET     BUDGET = BUDGET*1.1
            WHERE   PNAME= "CAD/CAM"
end.
```

The Begin transaction and end statements delimit a transaction. Note that the use of delimiters is not enforced in every DBMS. If delimiters are not specified, a DBMS may simply treat as a transaction the entire program that performs a database access.

Distributed Transaction

- A **distributed transaction** is a set of operations on data that is performed across two or more data repositories (especially databases). It is typically coordinated across separate nodes connected by a network, but may also span multiple databases on a single server.
- There are two possible outcomes: 1) all operations successfully complete, or 2) none of the operations are performed at all due to a failure somewhere in the system.

Goals of Distributed Transaction

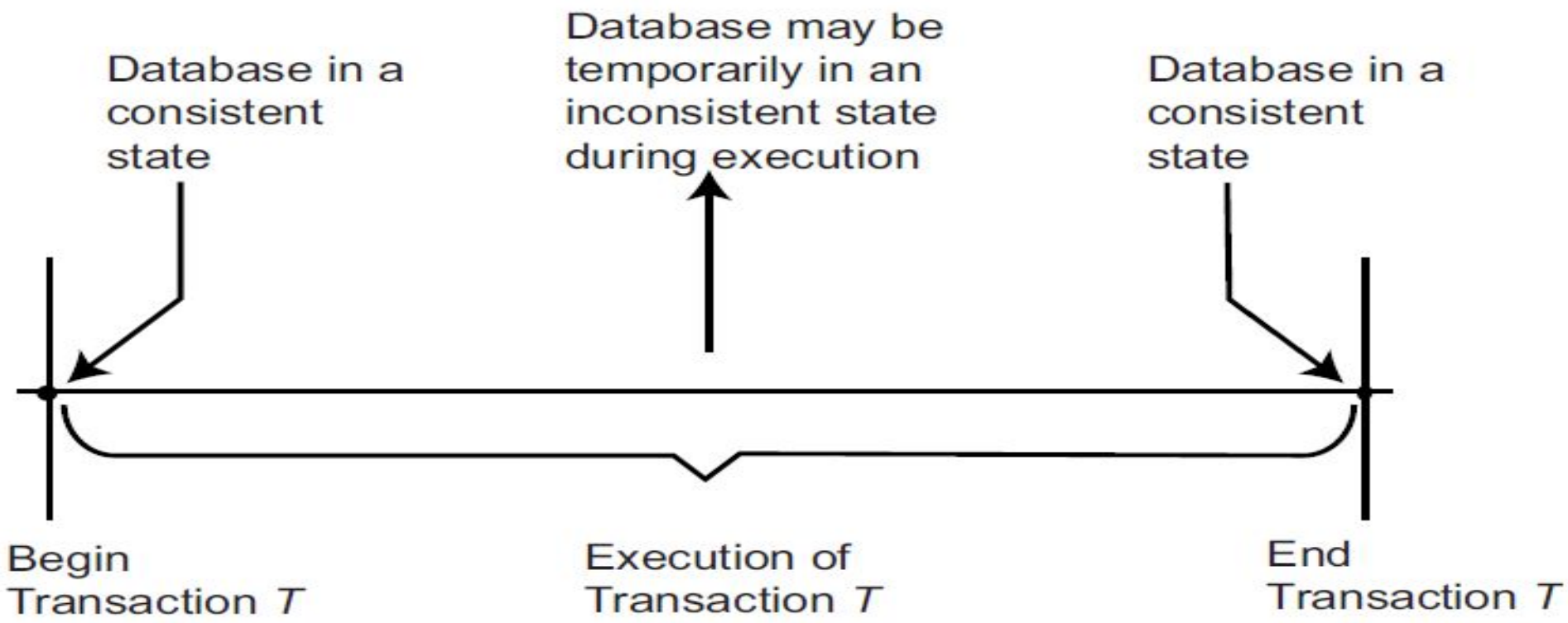
- To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure. For example, when execution prematurely and unexpectedly stops (completely or partially), many operations upon a database remain uncompleted, with unclear status.
- To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the programs' outcomes are possibly erroneous.

Characteristics of a Transaction

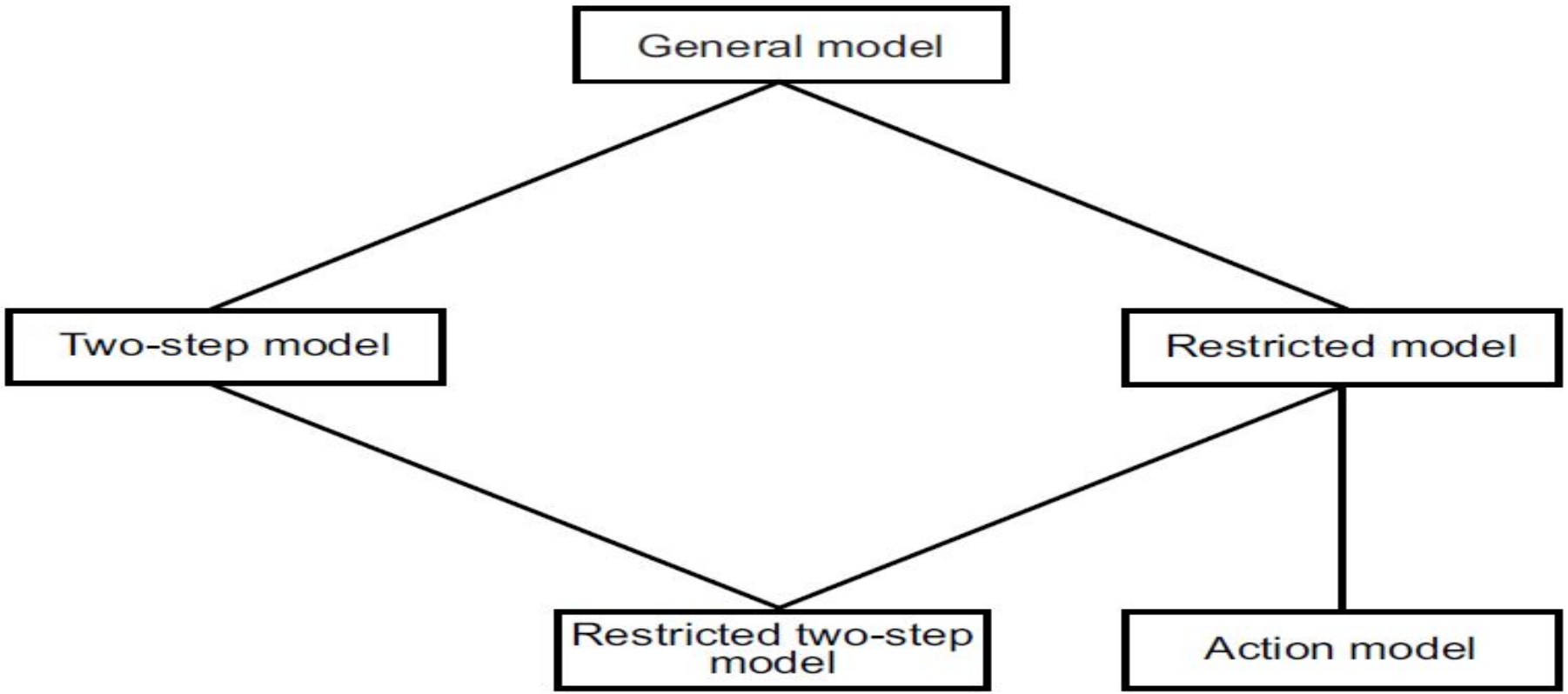
Typically, when we talk about database transactions, we are talking about ACID transactions.

- **Atomicity**: Ensures that all operations completed within the transaction either all complete or all fail. If any of the changes we're trying to make fail for some reason, then the whole operation is aborted, and it's as though no changes were ever made.
- **Consistency**: When changes are made to our database, we ensure it is left in a valid, consistent state.
- **Isolation**: Allows multiple transactions to operate at the same time without interfering.
  This is achieved by ensuring that any interim state changes made during one transaction are invisible to other transactions.
- **Durability:** Makes sure that once a transaction has been completed, we are confident the data won't get lost in the event of some system failure.

Transaction Model



Various Transaction Models



Various Transaction Models

- General: Intermix of read and write operations without any specific ordering $T_1 : \{R(x), R(y), W(y), R(z), W(x), W(z), W(w), C\}$
- Two-step: If the transactions are restricted so that all the read actions are performed before any write action

$$T_2 : \{R(x), R(y), R(z), W(x), W(z), W(y), W(w), C\}$$

- Restricted: if the transaction is restricted so that a data item has to be read before it can be updated (written)

$$T_3 : \{R(x), R(y), W(y), R(z), W(x), W(z), R(w), W(w), C\}$$

- Restricted two-step: If a transaction is both two step and restricted

$$T_4 : \{R(x), R(y), R(z), R(w), W(x), W(z), W(y), W(w), C\}$$

- Action model: consists of the restricted class with the further restriction that each (read, write) pair be executed atomically.

$$T_5 : \{[R(x), W(x)], [R(y), W(y)], [R(z), W(z)], [R(w), W(w)], C\}$$

Other Transactions based on Structures

- *Flat Transaction:* Flat transactions have a single start point (Begin transaction) and a single termination point (End transaction).
- *Nested Transaction:* a transaction that includes other transactions with their own begin and commit points.
  - ❑ Closed nested transactions: commit in a bottom-up fashion through the root. Thus, a nested subtransaction begins after its parent and finishes before it, and the commitment of the subtransactions is conditional upon the commitment of the parent.
  - ❑ Open nested transactions: allows its partial results to be observed outside the transaction such as *sagas.*
  - ❑ Two properties of sagas are:

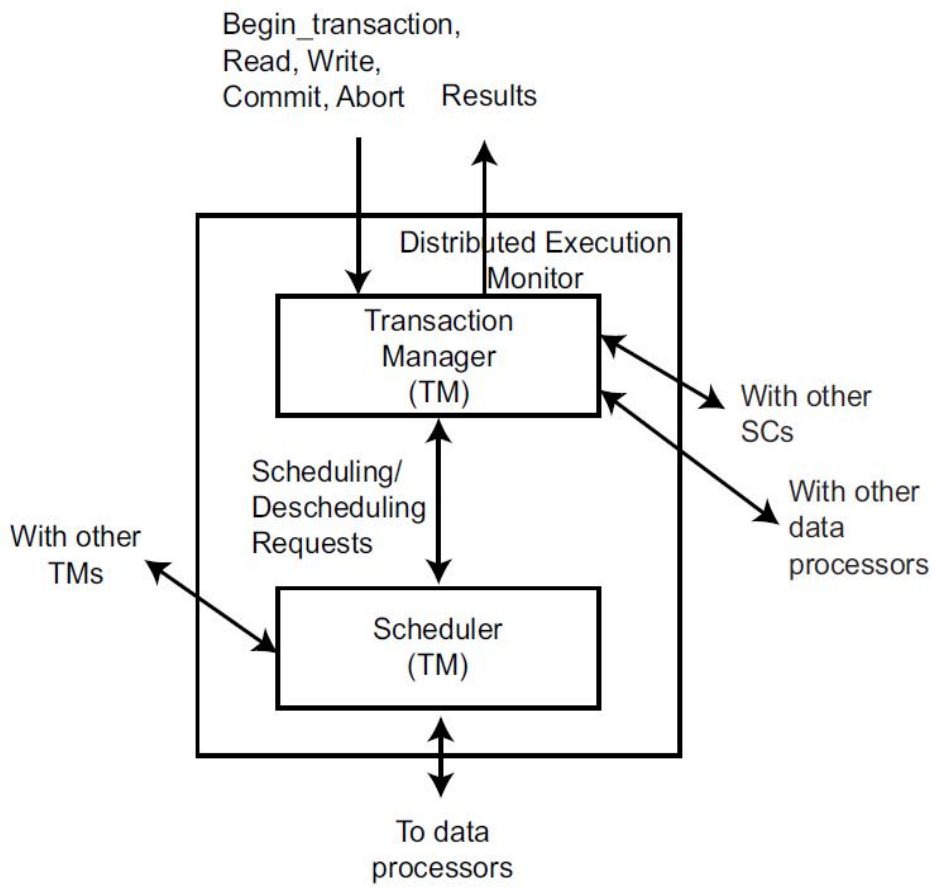
    (1) only two levels of nesting are allowed, and (2) at

    (2) and that it is open
- *Workflow:* collection of tasks organized to accomplish some business process

Architecture of Distributed Transaction

The distributed execution monitor consists of two modules:

- A Transaction Manager(TM): The transaction manager is responsible for coordinating the execution of the database operations on behalf of an application. The transaction managers implement an interface for the application programs which consists of five commands: begin transaction, read, write, commit, and abort, explained in the next slide.
- A Scheduler (SC): The scheduler, on the other hand, is responsible for the implementation of a specific concurrency control algorithm for synchronizing access to the database.



Architecture of Distributed Transaction

1. Begin transaction. This is an indicator to the TM that a new transaction is starting. The TM does some bookkeeping, such as recording the transaction's name, the originating application, and so on, in coordination with the data processor.

2. Read. If the data item to be read is stored locally, its value is read and returned to the transaction. Otherwise, the TM finds where the data item is stored and requests its value to be returned (after appropriate concurrency control measures are taken).

3. Write. If the data item is stored locally, its value is updated (in coordination with the data processor). Otherwise, the TM finds where the data item is located and requests the update to be carried out at that site after appropriate concurrency control measures are taken).

4. Commit. The TM coordinates the sites involved in updating data items on behalf of this transaction so that the updates are made permanent at every site.

5. Abort. The TM makes sure that no effects of the transaction are reflected in any of the databases at the sites where it updated data items.

In providing these services, a TM can communicate with SCs and data processors at the same or at different sites.