The Transaction Concept: Virtues and Limitations

Jim Gray
VLDB ’81

Shiven Mian
Prashanth R Duggirala

What is a transaction?

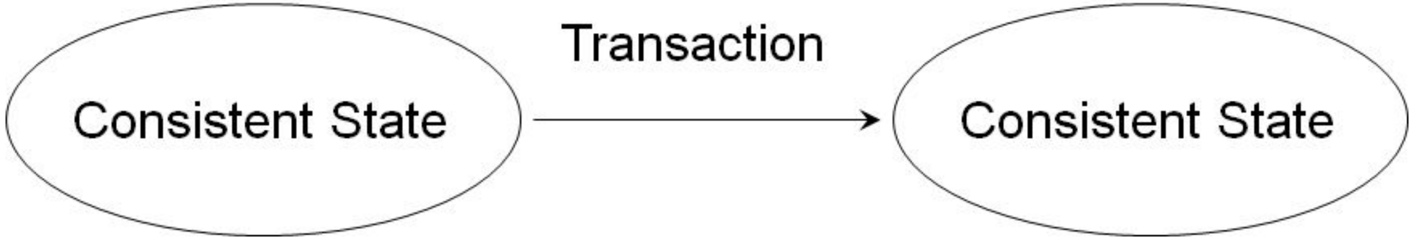
- A (set of) transformations from a consistent state to another consistent state.
- The transaction concept derives from contract law
- The transaction concept emerges with the following properties:
    - **Consistency**: The transaction must not be in violation of Laws
    - **Atomicity**: It either happens or it does not; either all are bound by the contract or none are
    - **Durability**: Once a transaction is committed, it stays - cannot be abrogated
    - **Isolation:** Transaction carried out as if it's the only one (not mentioned)

Transactions in the realm of CS

- The transaction concept was adopted to ease the programming of certain applications.
- Transformations of a system state.
- A system state consists of
    - Records and Devices with changeable values
    - Assertions about the values/allowed transformations of the values. (System consistency constraints)
    - Actions which read and transform the values

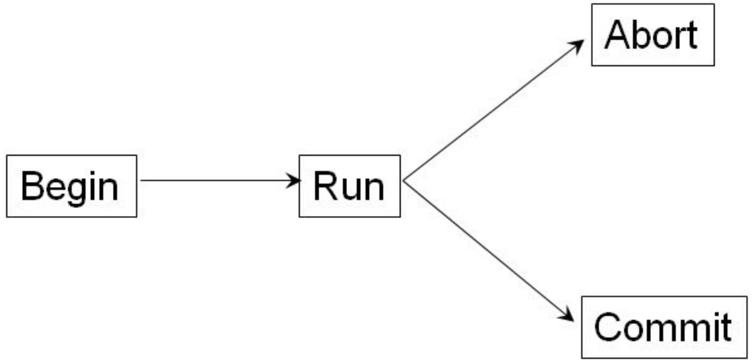
Transactions in the realm of CS

- A collection of actions may be grouped to form a transaction (Must be Consistent)
- Transactions preserve the system consistency constraints -- they obey the laws by transforming consistent states into new consistent states



Transactions in the realm of CS

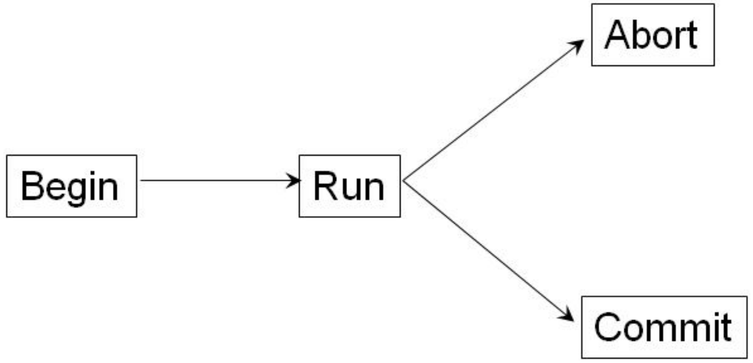
- Transactions must be atomic and durable, only two outcomes:



- All actions are done: **Commit**
- None of the effects of the transaction survive: **Abort**

Transactions in the realm of CS

- Transactions must be atomic and durable, there are only two outcomes:



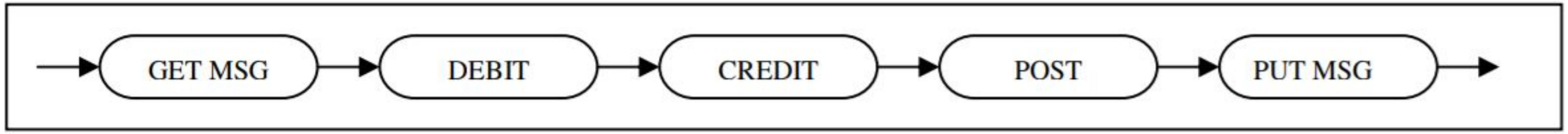
- Once a transaction commits, its effects can only be altered by running further transactions - compensating transactions.

Transactions in the realm of CS

- Actions on entities are categorized as:
    - **Unprotected**: the action need not be undone or redone if the transaction must be aborted or the entity value needs to be reconstructed.
        - Operations on temporary files during a transaction
    - **Protected**: the action can and must be undone or redone if the transaction must be aborted or if the entity value needs to be reconstructed.
        - Conventional database operations.
    - **Real**: once done, the action cannot be undone.
        - Transactions on real devices (ATM dispensing cash)

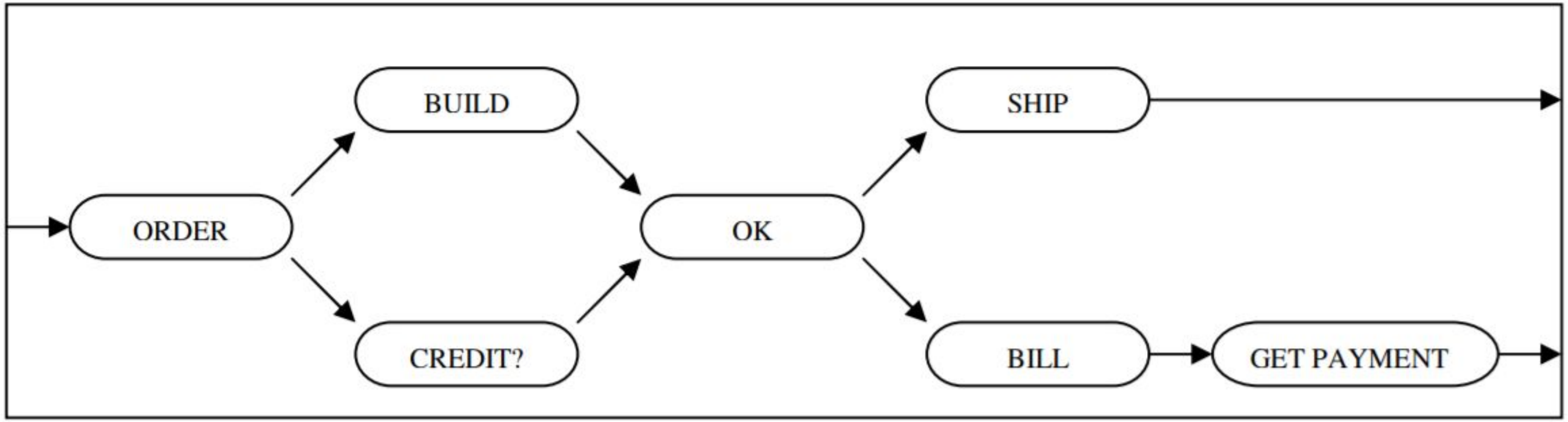
Types of Transactions

- A simple transaction is a linear sequence of actions.



Types of Transactions

- A complex transaction may have concurrency within a transaction
- The initiation of one action may depend on the outcome of a group of actions.
- The effects of the nested transactions are only visible to other parts of the transaction



Systems fail no matter how hard you try

- Even with a perfect system which never fails, People are imperfect
- People who adapted the system to their environment (Application Programming Error)
- People operating your system (Data entry/Procedural Errors/timeouts)
- A system failing once every several years is acceptable

Highly Reliable and Available Systems

- A system is unreliable if it does the wrong thing (i.e does not detect the error)
- A system is unavailable if it does not do the right thing within a specified time limit
- Can use unreliable components to build reliable systems. Need redundancy to protect from failure
- Neumann suggested redundancy and majority logic on a huge scale but can have single point of failure. Current systems have hierarchical fail-fast modules.
- Mirroring / duplexing modules, and making fail fast can increase the MTTF (mean time to failure of the system) from the failure time of one disk to the combined failure time of all the modules
- Each process may have a backup process which continues to do work of the primary process
- System is called NonStop

Writing Fault tolerant applications

- Backup process should continue computation where left off without propagating failure
- Primary process "checkpoint" its state to the backup process prior to each operation. If the primary fails, the backup process picks up where the primary left off
- Another approach: collect all the processes of a computation as a transaction
  - In the event of a failure, the transaction is undone (set to initial transaction state) and continued from that point by a new process.
  - Frees the application programmer from concerns about failures or process pairs.
  - Similar to programs in a conventional system except that they contain the verbs BEGIN-TRANSACTION, COMMIT-TRANSACTION and ABORT-TRANSACTION.
- Transaction lets user abort if system state is bad and reconstruct state using logs - provides durability.

Update in Place

- Traditional double entry bookkeeping - fail fast, and can't alter the books, use compensating entry to maintain history. Similar scenario in the first computer systems - writing a new tape is better than rewriting old.
- Direct Access Storage devices (disks / drums) followed different approach - only update parts that changed instead of copying the whole disc during change.

Implementing the Transaction concept

- Time-domain addressing
- Logging plus locking

Time-domain addressing

- An object is not “updated”; it is “evolved”
- **<Name, Time>** rather than **<Name>**
- Evolving an object consists of creating a new value and appending it as the current (as of this time) value of the object.
- The old value continues to exist and may be addressed by specifying any time within the time interval that value was current.

Time-domain addressing

- Evolving an object consists of creating a new value and appending it as the current (as of this time) value of the object.
- An entity **`E`** has a set of values **`Vi`** each of which is valid for a time period. For example the entity **`E`** and its value history might be denoted by:
    - **`E: <V0,[T0,T1)>, <V1,[T1,T2)>, <V2,[T2,*)>`**
- A transaction at time **`T3`** writing value **`V3`** to entity **`E`** starts a new time interval:
    - **`E: <V0,[T0,T1)>, <V1,[T1,T2)>, <V2,[T2,T3)>, <V3,[T3,*)>`**
- pseudo-time used in order to avoid the difficulties of implementing a global clock

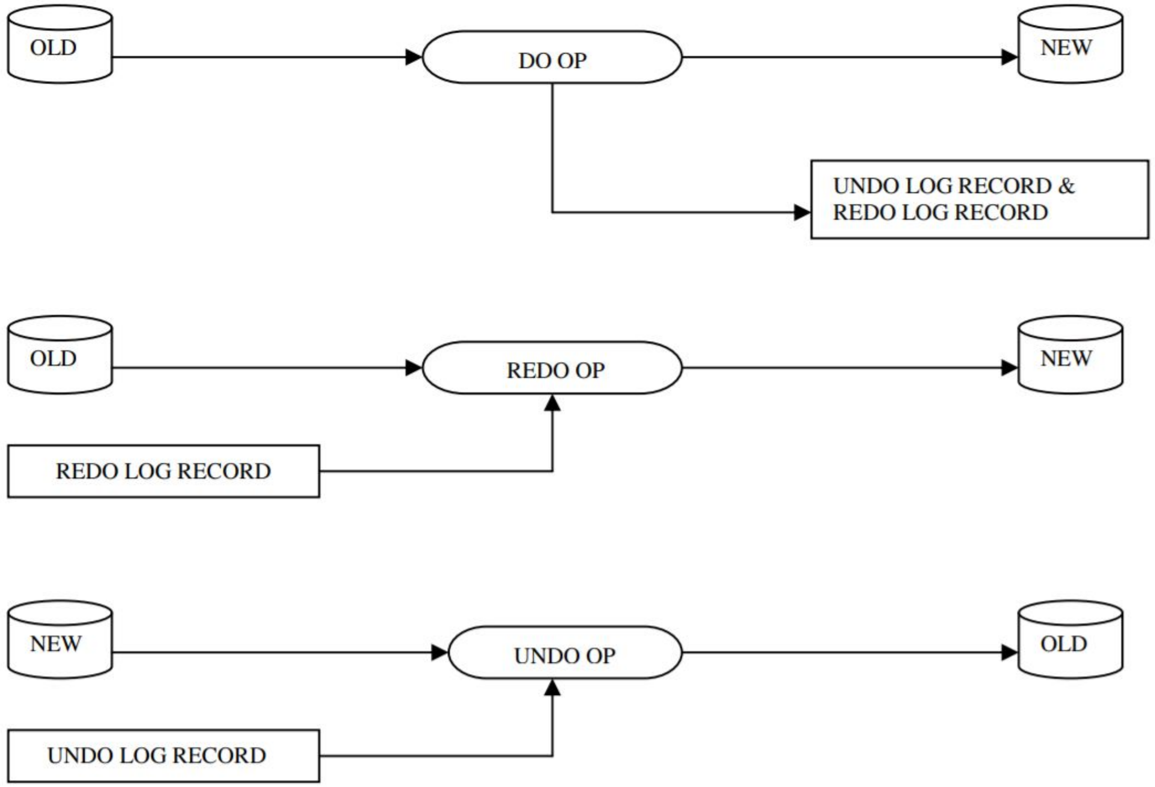
Problems with Time-domain addressing

- **Reads are writes:** reads advance the clock on an object and therefore update its header. This may increase I/O activity.
- **Waits are aborts:** In time-domain systems, conflicts abort the writer. May stop long-running “batch” transactions which do many updates.
- **Timestamps force a single granularity:** reading a million records updates a million timestamps.
- **Real operations and pseudo-time:** If one reads or writes a real device, It is unclear how real time correlates with pseudo-time and how writes to real devices are modeled as versions.

Logging and locking

- Every undoable action must both do the action but leave a trail allowing undo
- The execution of each protected action generates a log record which allows the action to be undone or redone.
- Unprotected actions need not generate log records.
- The records should be kept in stable storage – usually implemented by keeping the records on several non-volatile devices, each with independent failure modes.
- Occasionally, a stable copy of each object should be recorded so that the current state may be reconstructed from the old state. For multiple logs, transaction commit should be in all logs or none.
- Real action can't be undone so deferred till transaction commit (explained later)
- Undoing / redoing a transaction requires undoing / redoing every action in its log.
- Undo and redo must be restartable - if op is undone/redone, it must not damage obj state. Version numbers and sequence numbers used for restartability.

Logging and locking



Logging and locking

```
NAME OF TRANSACTION:
PREVIOUS LOG RECORD OF THIS TRANSACTION:
NEXT LOG RECORD OF THIS TRANSACTION:
TIME:
TYPE OF OPERATION:
OBJECT OF OPERATION:
OLD VALUE:
NEW VALUE:
```

Logging and locking

- For concurrent transactions - one transaction takes output of another, and undo requires undoing the second transaction, which may already be committed; this is why should defer / hide real actions until transaction commit.
- Repeatability: to ensure a transaction reads the same value repeatedly , i.e., it cannot read some uncommitted values of another transaction. To accomplish this, use exclusive write locks and shared read locks.
- Two phase commit protocol: A coordinator tells the nodes to commit, if a node said "yes, I will commit", it cannot say no later. The transaction commits if all nodes agreed to commit. Otherwise the transaction aborts. This is used for distributed transactions.
- Predicate locking: a predicate can covers the exact type of records that a transaction wants to lock. This is expensive.
- Hierarchical locking; pick a fixed set of predicates and group them into an acyclic graph and lock from root to leaf. Loses generality.

Limitations of known techniques

- Difficulties with current transaction models:
    - Transactions cannot be nested inside transactions.
    - Assumed to last minutes rather than weeks.
    - Not unified with programming language.

Nested Transactions

- Nested transactions differ from protected actions because their effects are visible to the outside world prior to the commit of the parent transaction.
- Compensating Transactions:
    - Transaction returns parameters to parent, which invokes if needed to be undone
    - Scratchpad – each state in database, loaded when active
- Some people also argue that nested transactions cannot be transactions
- They use the BEGIN, COMMIT and ABORT verbs. But they do not have the property of atomicity.

Long Lived Transactions

- Transactions with lifetimes in days
- Active transactions hold locks
- What if system restarted?
  - Transactions aborted – expensive
  - Salvaged with SAVE points.

Integration with Programming Languages

- Include begin-transaction, commit-transaction, and abort-transaction in PLs
- Logging also has to be included
- The performance of logging may be prohibitive

References

1. Gray, J., 1981, September. The transaction concept: Virtues and limitations. In VLDB (Vol. 81, pp. 144-154).
2. http://infolab.stanford.edu/~manku/quals/summaries/wong-transactions.htm
3. https://en.bitcoin.it/wiki/Transaction
4. http://db.cs.duke.edu/courses/cps216/fall11/Lectures/notes11.pdf
5. https://www.tutorialspoint.com/dbms/
6. http://courses.cs.vt.edu/~cs5204/fall05-gback/presentations/transactions.pdf

Thank you!