Concurrent events and consistency control

1 Concurrency

- **Shared Object** - May be simultaneously accessed by multiple events
- A server which manages shared objects is responsible for ensuring the objects remain consistent when accessed by concurrent events
- Such a control is called concurrency control

Extension to replication - If event T happens before event U in their conflicting access to objects at one of the servers then they must be in that order at all of the servers whose objects are accessed in a conflicting manner by both T and U

2 Locking

Grant a lock

- Locks on an object are held (in a server) locally
- Local lock manager can decide whether to grant a lock or make the requesting transaction wait

Issues with Distributed Transmission (DT)

- A DT acquires resource located at different servers
- Can't release any locks until the transaction has been committed or aborted at all servers involved in the transaction
- Objects remain locked and are unavailable for all other transactions during the atomic commit protocol

3 Timestamp ordering

Timestamp:

- Assign a timestamp to each transaction when it starts
- Serial equivalence is enforced by committing the versions of objects in the order of the timestamps of transactions that accessed them
- Requirement: Globally unique timestamps

Distributed Transaction:

- Servers of distributed transactions are jointly responsible for ensuring that they are performed in a serially equivalent manner
- To achieve the same ordering at all the servers, the coordinators must agree as to the ordering of their timestamps
- Issue - In practice, a timestamp is usually assigned by a local server, generating a timestamp, server id pair

4 Concurrent Operations

In a single machine:

- Concurrent operations (events), no matter originated from different or the same machine, are handled by the time sharing feature of an operating system
- Operations are implicitly executed one by one in a series under a single clock, i.e. order of operations is well-defined

In a distributed system:

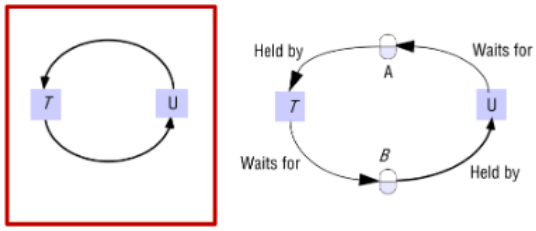
- Concurrent operations may run on different machines, i.e. distributed transactions
- Order of operations can't be easily sorted out due to overlapping operations, clock synchronisation problems, network latency and message loss etc

5 Deadlock Detection

- Local deadlocks can arise within a single server when locking is used for concurrency control
- Timeouts: clumsy solution; difficult to choose an appropriate timeout interval, and transactions may be aborted unnecessarily

Wait for graph:

- Most deadlock detections schemes operate by finding cycles in the transaction wait-for graph



5.1 Distributed Deadlock Detection

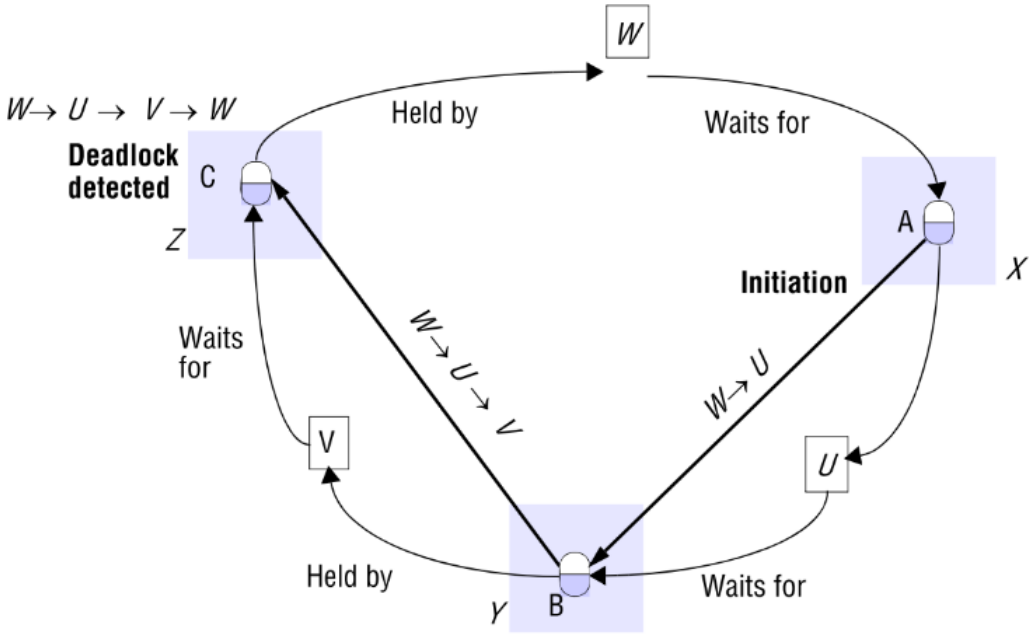
- Centralised deadlock detection is not a good idea, because it depends on a single server, suffering from poor availability, lack of fault tolerance and no ability to scale
- If the global graph is collected less frequently, deadlocks may take longer to be detected

Phantom deadlocks

- A deadlock that is ”detected” but is not really a deadlock
- Global deadlock detector requires time to receive local wait-for graphs for constructing the construct the global one. This global wait-for graph may no longer be valid by the time it is constructed

Edge chasing:

- Don’t construct a global wait-for graph
- Instead, each server involved attempts to find cycles by forwarding messages called probes
- A probe message consists of transaction wait-for relationships representing a (local) path in the global wait for graph



6 Correctness of replicated objects

- If a system only maintains a single copy for each object, object correctness simply relates to the sequence of operations applied on the object
- Correctness of replicated objects is challenging:
  - Object copies maintained by different machines, where each machine may receive operations with a different order
- Operations are linearisable if:
  - The interleaved sequence of operations meets the specification of a (single) correct copy of the objects
  - The order of operations in the interleaving is consistent with the real times at which the operations occurred in the actual execution

6.1 Linearisability

There exists a virtual canonical execution - the interleaved operations against a virtual single image of shared (and particularly replicated) objects, and that:

- Each client sees a view of the shared objects that is consistent with that single image
- i.e. results of the client's operations make sense as they occur within the interleaving

7 Consistency Control

Issues with distributed systems:

- They reply on replicating shared objects, or allowing concurrent access at many nodes
- If concurrent accesses of (shared) objects are not carefully controlled, object accesses may be executed in an order different from user expectation, generating incorrect results

Modelling of Consistency Control:

- Informally, certain object is coherent if the value returned by a read operation is always the value that the user expected
- Consistency model: Defined rules for the apparent order and visibility of updates, and it is a continuum with trade-offs

8 Types of consistency

Definition: Strict consistency

An read on a data item X returned a value corresponding to the result of the most recent write on X

Definition: Sequential Consistency

If for any transaction there is some interleaving of the series of operations issued by all the clients that satisfies the following two criteria

- The interleaved sequence of operations meets the specification of a (single) correct copy of the contents
- The order of operations in the interleaving is consistent with the program order in which each individual process executed them

Definition: Casual Consistency

If event B is caused or influenced by an earlier event A, everyone first see A and then B