

Timestamp Ordering/1

- ▶ Algorithms that are based on **timestamp-ordering** do not maintain serializability by mutual exclusion, but select a serialization order a priori (based on timestamps) and execute transactions accordingly.
 - ▶ Transaction T_i is assigned a globally unique timestamp $ts(T_i)$.
 - ▶ Conflicting operations O_{ij} and O_{kl} are resolved by timestamp order, i.e., O_{ij} must be executed before O_{kl} iff $ts(T_i) < ts(T_k)$.
- ▶ To allow for the scheduler to check whether operations arrive in correct order, each data item is assigned a write timestamp (wts) and a read timestamp (rts):
 - ▶ $rts(x)$: largest timestamp of any transaction that read x
 - ▶ $wts(x)$: largest timestamp of any transaction that wrote x

Timestamp Ordering/2

- ▶ Then the scheduler has to perform the following checks:
 - ▶ Read operation, $R_i(x)$:
 - ▶ If $ts(T_i) < wts(x)$: T_i attempts to read overwritten data; abort T_i
 - ▶ If $ts(T_i) \geq wts(x)$: the operation is allowed and $rts(x)$ is updated
 - ▶ Write operations, $W_i(x)$:
 - ▶ If $ts(T_i) < rts(x)$: x was needed before by other transaction; abort T_i
 - ▶ If $ts(T_i) < wts(x)$: T_i writes an obsolete value; abort T_i
 - ▶ Otherwise, execute $W_i(x)$
- ▶ These checks and actions mean that the older transaction is aborted if there is a conflict.

Timestamp Ordering/3

- ▶ Generation of **timestamps** (TS) in a distributed environment
 - ▶ TS needs to be locally and globally **unique** and **monotonically increasing**
 - ▶ System clock, incremental event counter at each site, or global counter are unsuitable (difficult to maintain)
 - ▶ Concatenate local timestamp/counter with a unique site identifier: $\langle \text{local timestamp}, \text{site identifier} \rangle$
 - ▶ site identifier is in the least significant position in order to distinguish only if the local timestamps are identical

Timestamp Ordering/4

- ▶ Schedules generated by the basic TO protocol have the following **properties**:
 - ▶ Serializable
 - ▶ Since transactions never wait (but are rejected), the schedules are deadlock-free
 - ▶ The price to pay for deadlock-free schedules is the potential restart of a transaction several times
 - ▶ Cascading rollbacks are possible
 - ▶ Recoverability requires additional steps that ensure that commits happen in the correct order

► Multiversion timestamp ordering

- Write operations do not modify the DB; instead, a new version of the data item is created: x_1, x_2, \dots, x_n
- $R_i(x)$ is always successful and is performed on the appropriate version of x , i.e., the version of x (say x_v) such that $wts(x_v)$ is the largest timestamp less than $ts(T_i)$
- $W_i(x)$ produces a new version x_w with $ts(x_w) = ts(T_i)$ if the scheduler has not yet processed any $R_j(x_r)$ on a version x_r such that

$$ts(T_i) < rts(x_r)$$

i.e., the write is too late.

- Otherwise, the write is rejected.

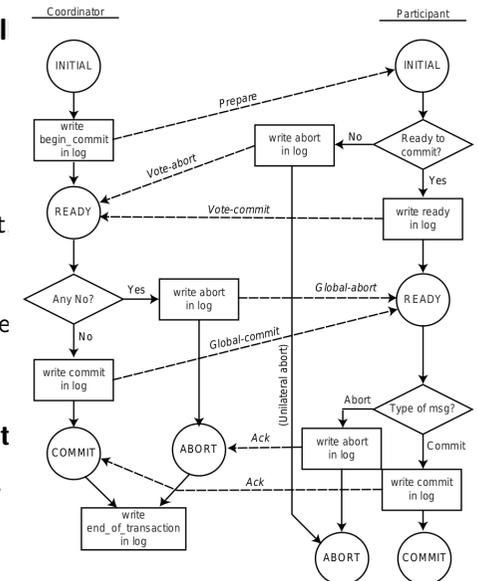
- Distributed reliability maintain is about **atomicity and durability** of distributed transactions that execute over a number of databases.
- Distributed reliability handles the distributed execution of **begin transaction, read, write, abort, commit, and recover** commands.
- The execution of the begin transaction, read, and write commands does not cause any significant new problems.
- The reliability techniques in distributed database systems consist of **commit, termination and recovery protocols**.

Termination and Recovery Protocols

- The termination and recovery protocols specify how the recover command is executed.
 - **Termination protocols** specify what active sites must do after a failure.
 - The termination protocols handle timeouts for coordinator and participants.
 - A timeout occurs if a site does not get an answer within the expected time period.
 - **Recovery protocols** specify actions that must be done when a site restarts after a failure
 - Protocol to recover when sites fail and then restart.
 - Use logs to undo and redo.
 - May have to communicate with other sites to figure out decisions.

Two Phase Commit (2PC)

- The 2PC protocol insists that **all sites agree** to commit before the changes of the transaction are made permanent.
- **Global commit rule:**
 - If even one site votes to abort the coordinator must reach a global abort decision.
 - If all sites vote to commit, the coordinator must reach a global commit decision.
- Each site can **unilaterally abort**
- Sites cannot change their votes.
- **Timers** are used to limit the waiting time.



Conclusion

- ▶ A **transaction** is a set of operations with a partial order
- ▶ The **transaction manager** ensures atomicity, consistency, isolation, and durability
- ▶ A **schedule** is some order of the operations of the given transactions. If a set of transactions is executed one after the other, we have a serial schedule.
- ▶ There are two main groups of serializable concurrency control algorithms: **lock based** and **timestamp based**
- ▶ Local properties do not ensure global properties but can be generalized.
- ▶ 2PC to commit in distributed database systems.

Course Project

- ▶ Hand in of project: December 17, 2018
- ▶ Report
 - ▶ problem definition
 - ▶ running example
 - ▶ description of solution
 - ▶ evaluation
 - ▶ strength, weaknesses, limitations
- ▶ Report (5 pages) and implementation (source code, sample data, steps to install and run) as zip/tar file
- ▶ Send by email to boehlen@ifi.uzh.ch and wellenzohn@ifi.uzh.ch

Course Exam

- ▶ Exam date: Monday 7.01.2019
- ▶ Exam time: 14:00 - 17:00
- ▶ Exam location: BIN 2.E.13
- ▶ Exam form and procedure
 - ▶ oral, 20 minutes
 - ▶ 10 minutes about project (demo, code, algorithm)
 - ▶ 10 about a topic of the course
 - ▶ Distributed Database Systems
 - ▶ Distributed Database Design
 - ▶ Distributed Query Processing
 - ▶ Distributed Query Optimization
 - ▶ Distributed Transactions and Concurrency Control
- ▶ During exam: present solutions on examples
- ▶ Prepare suitable examples beforehand