

Distributed Data Management

2007-2008

*Christophe Bobineau
Claudia Roncancio
Cyril Labbé*

References

- T. Ozsü, P. Valduriez, Principles of Distributed DB Systems, Prentice Hall, 1999
 - J. Gray, Transaction Processing: concepts and techniques, Morgan Kaufmann, 1993
 - G. Gardarin, Bases de Données: relationnel et objet", Eyrolles, 1999
 - G. Gardarin, P. Valduriez, SGBD Avancés, Bases de données objet, déductives et réparties, Eyrolles, 1990
 - E. Pitoura, G. Samaras, Data Management for Mobile Computing, Kluwer Academic Publishers, 1998
-

Part 1 : Distributed transaction management

Contents

- **Distributed Transaction Management**
 - ◆ Introduction
 - ◆ Concurrency control
 - ◆ Reliability protocols
 - ◆ Transaction models
-

Transaction definition

- **Group of database accesses**
 - **Treatment unit for applications**
 - ◆ A transaction transform a consistent database state in another consistent database state
 - **Viewed as a process in database management**
 - **Possible termination:**
 - ◆ Commit
 - ◆ Rollback (Abort)
-

Transaction properties

- **Atomicity**
 - ◆ All database accesses included in a transaction are executed, or no one
 - ◆ Use of undo-log
 - **Consistency**
 - ◆ Transaction effects must respect all defined integrity constraints
-

Transaction properties...

- **Isolation**
 - ◆ A transaction must not see intermediate results of others
 - ◆ Concurrency control problem
 - **Durability**
 - ◆ Effects of committed transactions should never be lost
 - ◆ Use of redo-log, backup, etc.
-

Distributed transaction (DT)

- **Definition**
 - ◆ Composed of several sub-transactions (ST)
 - ◆ Each sub-transaction run on a different site
 - **DT management more complicated than in centralized systems**
 - **Distribution influence on:**
 - ◆ Concurrency control
 - ◆ Reliability management
 - ◆ Transaction commitment
 - ◆ ...
-

Contents

- **Distributed Transaction Management**
 - ◆ Introduction
 - ◆ Concurrency control
 - ◆ Reliability protocols
 - ◆ Transaction models

Concurrency control

- **Objective**
 - ◆ Assure transaction isolation
 - ◆ Forbid non serializable execution of multiple transactions
 - **Global serializability**
 - ◆ Serializable execution of sub-transactions at each site
 - ◆ Identical transaction serialization order at all sites
 - **Approaches in distributed environment**
 - ◆ Timestamp management
 - ◆ Lock management
-

Example

- **Database:**
 - ◆ Object A at site S_1
 - ◆ Object B at site S_2
 - **Transactions:**
 - ◆ $T_1 = R_1(A), W_1(A), R_1(B), W_1(B)$
 - ◆ $T_2 = R_2(A), W_2(A), R_2(B), W_2(B)$
 - **Execution:**
 - ◆ At site S_1 : $R_1(A), W_1(A), R_2(A), W_2(A) \rightarrow T_1, T_2$
 - ◆ At site S_2 : $R_2(B), W_2(B), R_1(B), W_1(B) \rightarrow T_2, T_1$
 - **Not globally serializable !**
-

Timestamp management

- **Principle:**
 - ◆ Global order determined by transaction execution time (timestamp = unique order number)
 - ◆ Each concurrent access must respect this order
 - **Problem:**
 - ◆ Unique timestamp generation
-

Timestamp generation

- **Centralized**
 - ◆ Pro: simplicity
 - ◆ Cons: bottleneck, single point of failure, communication cost
 - **Distributed**
 - ◆ Each site use its own clock (or counter) and its id (timestamp <id, clk>)
 - ◆ Pro: site autonomy, efficiency
 - ◆ Cons: difficulty of clock synchronization
-

Clock synchronization approach

- 📩 Each message is timestamped by current clock value (let's say t) of emitter site
- 📩 On reception, a site synchronize itself if it is late by setting its clock to $t + 1$

Sites that do not exchange messages are not synchronized: not a problem, they don't cooperate...

Timestamp approach summary

- **Advantage**
 - ◆ Efficient method
- **Inconvenient**
 - ◆ Can imply a lot of transaction undoing (in case of transaction abortion)

Lock management is preferred...

Lock management

- **Largely used !**
 - **All transactions have to use strict two-phase-locking protocol (Strict 2PL)**
 - **Different alternatives to guarantee serializability:**
 - ◆ Centralized lock management
 - ◆ Distributed lock management
 - ◆ Locking in case of data duplication
-

Two-phase locking

- **Definition**
 - ◆ Every transaction must obtain a lock on a data before its access
 - ◆ Different types of locks (shared, exclusive, ...) and compatibility table
 - ◆ If a transaction cannot obtain a lock, it waits until lock liberation
 - ◆ 2 phases:
 - ★ Locking phase
 - ★ Liberation phase (no other lock can be reclaimed)
 - **Strict 2PL: liberation phase at commit (or abort) time**
-

Centralized lock management

- **Principle**
 - ◆ Unique lock table for all the distributed database
 - ◆ Lock table located at only one site
 - ◆ Only one lock manager
 - **Advantage**
 - ◆ Easy global serializability enforcement
 - **Inconvenient**
 - ◆ Bottleneck, single point of failure, communication cost
-

Distributed lock management

- **Principle**
 - ◆ Distribution over all sites containing data
 - ◆ Locks obtained where data are stored
 - ◆ Necessary inter-site communication
- **Preferable approach, even if it's more expensive**
- **Main difficulty: deadlock management**

Duplicated data locking

Duplication increase reliability and availability but complicate lock management

- **Locking strategies**
 - ◆ All copies locking
 - ◆ Master copy locking
 - ◆ Quorum based locking
-

All copies locking

- **Principle**
 - ◆ Shared locks are obtained locally
 - ◆ Exclusive locks must be obtained at all copy sites
- **Advantage**
 - ◆ Guarantee serializability
- **Inconvenient**
 - ◆ Costly

Master copy locking

- **Principle**
 - ◆ One of the copies is the master
 - ◆ Master copy must be locked before data access (on any site)
 - **Consequence**
 - ◆ Every site must know master copy location for all their data
-

Wait-for graph management

- **Very costly activity**
- **3 approaches**
 - ◆ Centralized management
 - ◆ Hierarchical management
 - ◆ Distributed management
 - ★ No global graph materialization

Centralized deadlock detection

- **Only one deadlock manager**
 - **Periodically, each site sends its local wait-for graph to the deadlock manager**
 - ◆ Complete
 - ◆ Containing only potential conflicts
 - ◆ Graph updates
 - **Compromise between detection delays and communication cost**
 - **Reduced autonomy (graph sharing)**
-

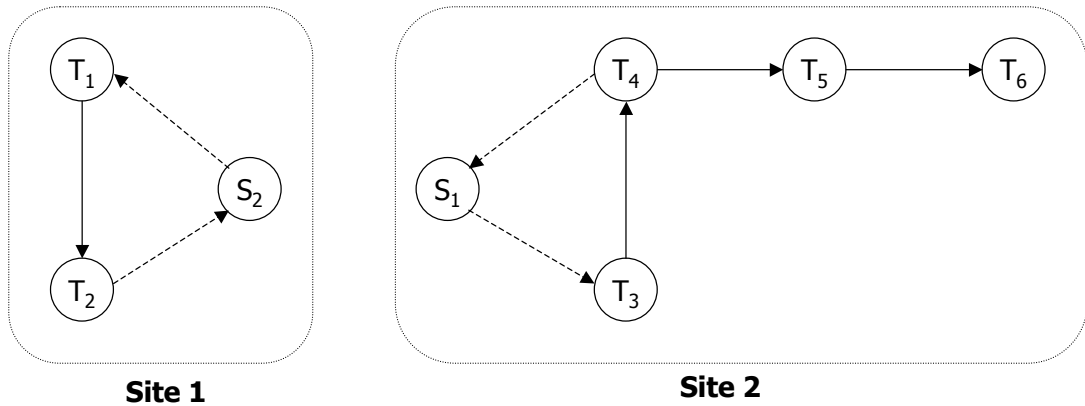
Hierarchical deadlock detection

- **Hierarchy of deadlock detectors**
 - **Local wait-for graph sent to each node parent**
 - **Advantages**
 - ◆ Detection at lower level
 - ◆ Lower communication cost
 - **Inconvenient**
 - ◆ More complicated
 - ◆ Reduced autonomy
-

Distributed deadlock detection

- **Each site maintain a global wait-for graph**
 - **Nodes:**
 - ◆ Local nodes
 - ◆ One per distant site
 - **Communication between sites to share global wait-for graphs**
 - **Local detection**
-

Distributed deadlock detection



- **Advantage**
 - ◆ Easier implementation (uniform)
- **Inconvenient**
 - ◆ Risk of multi-abortion (not the same decision for all sites participating in a deadlock cycle)

Contents

- **Distributed Transaction Management**
 - ◆ Introduction
 - ◆ Concurrency control
 - ◆ Reliability protocols
 - ◆ Transaction models
-

Reliability protocols

- **Properties guaranteed**
 - ◆ Atomicity
 - ◆ Durability
 - **Need of**
 - ◆ Atomic commitment protocol
 - ◆ Distributed failure recovery protocol
 - **Invariant: every sub-transaction participating in one global transaction must take the same termination decision (*commit* or *abort*)**
-

Two-phase commit protocol

- **Largely employed !**
 - **Principle**
 - ◆ Global transaction is committed if and only if all its sub-transaction are committed
 - ◆ If at least one of the sub-transaction is aborted, all others must be too
 - **Roles**
 - ◆ Participant sites (executing a sub-transaction)
 - ◆ Coordinator site (executing 2PC protocol)
-

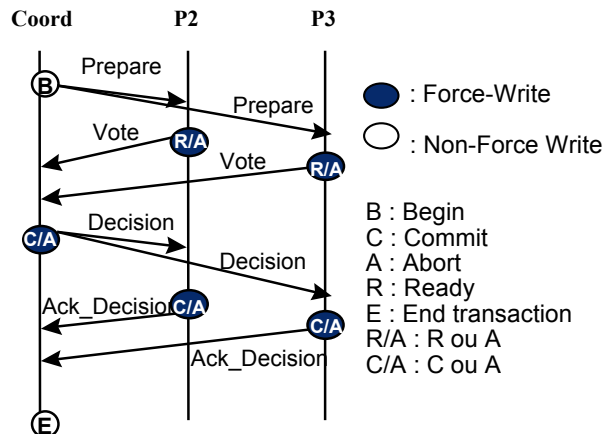
Description

- **Preparation phase**
 - ◆ The coordinator asks all participants to prepare themselves to commit their local sub-transactions
 - ◆ Each participant sends to the coordinator its vote (ready to commit or not)
 - **Validation phase**
 - ◆ The coordinator sends its decision to all participants
 - ◆ This decision takes into account each participant answer
-

Failure recovery

- **Coordinator log**
 - ◆ Answers from participants
 - ◆ Decision taken
 - ◆ Acknowledgements of participants
 - **Extended participant log**
 - ◆ Prepare order: $\langle T_i, \text{prepare} \rangle$
 - ◆ Answer: $\langle T_i, \text{ready} \rangle$ or $\langle T_i, \text{not ready} \rangle$
 - ◆ Decision: $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$
-

2PC summary

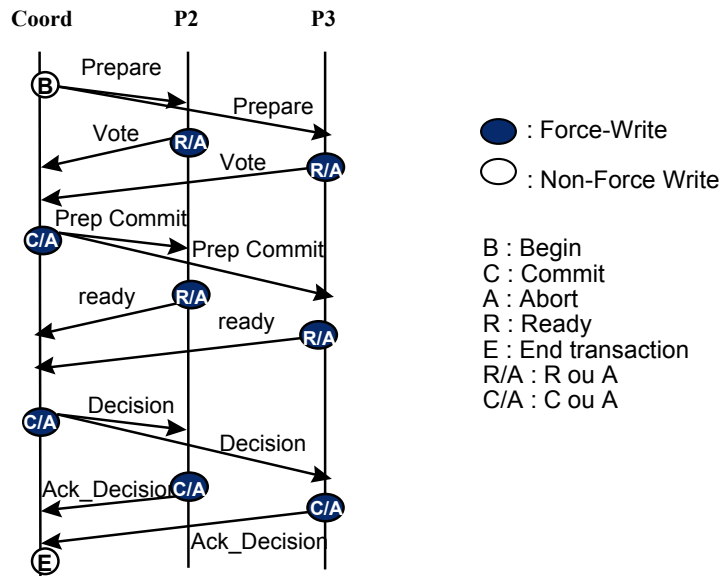


- Advantages
 - Largely employed (almost every DBMS)
 - Standardized (ex: X/open consortium XA)
- Inconvenient
 - Blocking protocol

Three-phase protocol (3PC)

- Non blocking atomic commitment protocol
- Description
 - Preparation phase
 - Prepare* message propagation
 - Votes return (*vote commit* or *vote abort*)
 - Pre-commit phase (only if all votes are *commit*)
 - Prepare to commit* message propagation
 - Participant acknowledgement (*ready to commit*)
 - Decision phase
 - Decision propagation
 - acknowledgements

3PC summary



- Advantages
 - ◆ Non blocking
 - Inconvenient
 - ◆ 3 Phases
-

Contents

- **Distributed Transaction Management**
 - ◆ Introduction
 - ◆ Concurrency control
 - ◆ Reliability protocols
 - ◆ Transaction models
-

Transaction models

- Flat transactions
- Nested transactions
- Sagas

Flat transactions

- Simplest and most frequent model
 - Unique level of control
 - Possibility of checkpoints
 - ◆ Intermediate checkpoints to permit partial rollback
 - ◆ Implies transaction status storage
 - ◆ In case of failure, entire transaction rollback
-

Nested transactions

- **Composition**
 - ◆ Top level transaction
 - ◆ Set of component transactions (sub-transactions)
 - **Properties**
 - ◆ A sub-transaction can be nested
 - ◆ Concurrent sub-transaction execution
 - ◆ Preserved serializability (no cooperation between sub-transactions)
 - **Advantages**
 - ◆ Improves parallelism
 - ◆ Better flexibility in operation failure
-

Termination conditions

- **A sub-transaction abortion do not implies top level transaction abortion**
 - **A sub-transaction begins after its mother and terminate before its mother**
 - ◆ The top-level transaction can terminate if and only if all its sub-transactions have
 - **In case of top-level transaction abortion, all its sub-transactions must be aborted**
 - ◆ Sub-transaction effects are visible for other transactions only after its top-level transaction commitment
-

Visibility rules

- **A sub-transaction can see:**
 - ◆ Last DB version accessible to its mother transaction
 - ◆ Effects of its committed sisters, or of other committed transactions
 - **A sub-transaction can access data manipulated by one of its ancestors**
 - ◆ If a sub-transaction request a lock, it is given only if all transactions possessing incompatible locks are one of its ancestors
-

Visibility rules...

- **At sub-transaction commitment time, its locks are inherited by its mother**
 - **At sub-transaction abortion time, its locks are liberated.**
 - ◆ If one of its ancestors possess a lock on the same object, the ancestor keeps it
-

Sagas

- Transaction model for long-time activities
 - Saga = set of relatively independent sub-transactions $\{T_1, T_2, \dots, T_n\}$ that can be interlaced with sub-transactions of other sagas
 - Sub-transaction execution is done in a predefined order
 - For each sub-transaction T_i , a compensation transaction CT_i is defined for undoing, in a semantic point of view, T_i effects
 - T_i and CT_i are ACID transactions
-

Termination

- A sub-transaction can commit without waiting for other sub-transaction commitment, nor saga commitment
 - The saga can commit only if all its sub-transaction have committed in the predefined order
 - If the saga is aborted, compensation transactions are rolled back in inverse order
 - A compensation transaction can commit only if its corresponding sub-transaction have and if the saga has been rolled back.
-

Isolation

	Dirty Read	Non Repeatable reads	Phantom Read	Write Skew
Uncommitted read	Possible	Possible	Possible	Possible
Committed Read	Non Possible	Possible	Possible	Possible
Repeatable Reads	Non Possible	Non Possible	Possible	Non Possible
Snapshot isolation	Non Possible	Non Possible	Non Possible	Possible
Serializable	Non Possible	Non Possible	Non Possible	Non Possible
