Advanced Databases

Distributed Transactions

Dr. George Mertzios
Michaelmas Term

george.mertzios@durham.ac.uk
Room 2066, MCS Building
Tel: 42 429

Course Outline

- Enhanced Entity-Relationship (EER) Model
- Semistructured Databases - XML
- XML Data Manipulation - XPath, XQuery
- Transactions and Concurrency Control
- **Distributed Transactions**
- Distributed Concurrency Control

Distributed databases (recap)

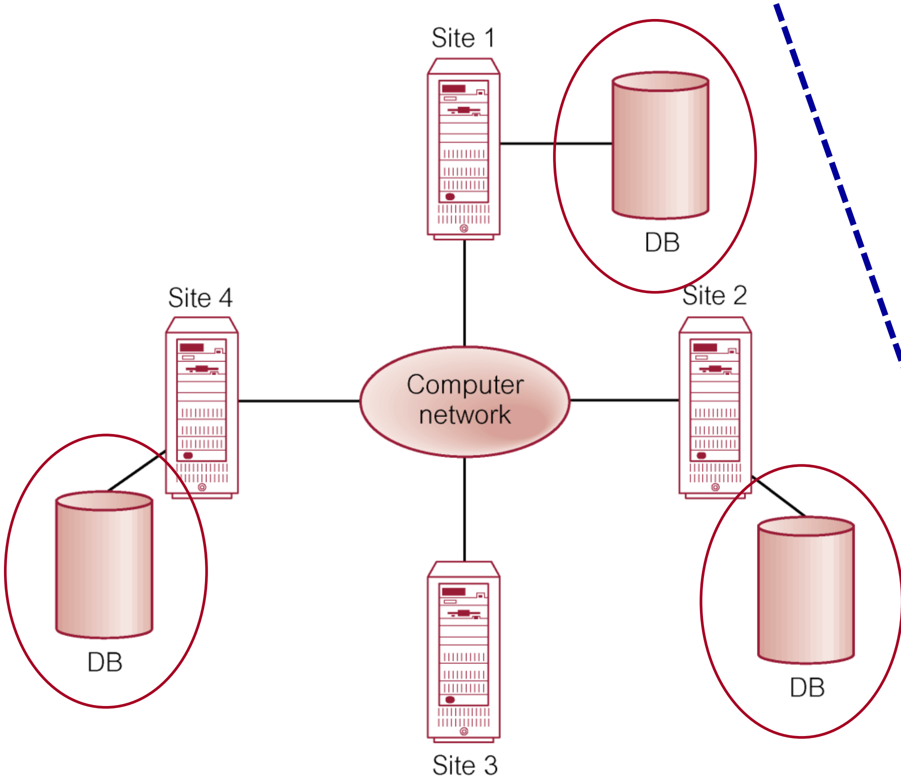
- Distributed database:
  - a collection of shared data, distributed over a network

- Distributed-DBMS (DDBMS):
  - the software system managing the distributed database

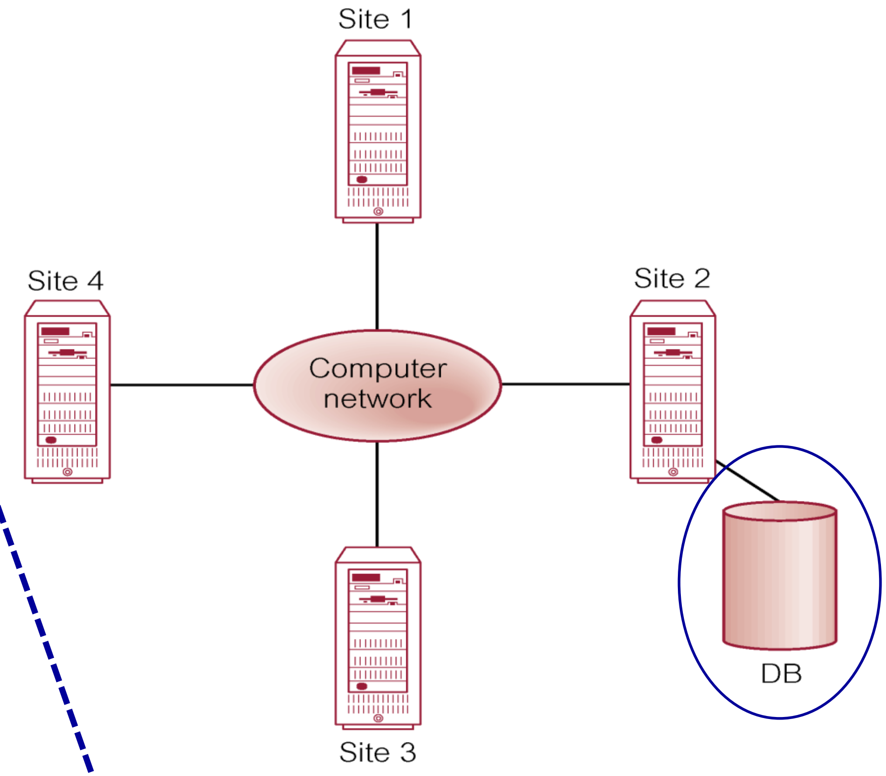
- In a DDBMS:
  - a single logical database, split into fragments
  - each fragment is stored on one (or more) sites
  - sites have local autonomy (using their own DBMS)
  - sites have access to the global database
    (using their network connections to other sites)

Distributed databases (recap)

distributed DBMS:



distributed processing
(no distributed DBMS):



Distributed databases (recap)

- A distributed database can be:
  - partitioned
    - database partitioned into disjoint fragments
    - each data item assigned to exactly one site (no replication!)
    - no data redundancy
  - completely replicated
    - complete copy of the database at each site
    - allows faster data retrieval
  - selectively replicated
    - combination of partitioning and replication

Distributed Transactions

- A transaction in a DDBMS:
  - is initiated in one of the sites
  - is divided into sub-transactions (one for each site)
- The DDBMS must ensure:
  - synchronization of sub-transactions with other local transactions
  - ACID properties of (local / global) transactions
- All "centralized" problems still exist:
  - lost update, uncommitted dependency, inconsistent analysis
- But also a new one appears:
  - multiple-copy consistency problem
  - when an item is updated, it must be updated at every site
    - otherwise inconsistent global database

Distributed Serializability

- The notions of schedule and serializability:
  - naturally extend to the distributed environment
  - local schedule (of sub-transactions) vs. global schedule
- A global schedule is serializable if:
  - each local schedule is serializable (at each site), and
  - the local serialization orders (of transactions) are identical

In other words:

- all sub-transactions appear at every site
  in the same order in the equivalent serial schedule

Distributed Serializability

That is:

- n sites $S_1, S_2, \ldots, S_n$
- Denote by T_i^x the sub-transaction of T_i at site S_x
- A global schedule of transactions is serializable if:
  - whenever $T_i^x < T_j^x$ at some site S_x,
  - we have that $T_i^y < T_j^y$ for every site S_y

In other words:

- all sub-transactions appear at every site
  in the same order in the equivalent serial schedule

Distributed Serializability

Consider a quite <u>restrictive</u> schedule of transactions:

- every site ensures local serializability
  - i.e. there exists an equivalent local serial schedule at that site
- at every time at most one transaction is active

Even then:

- global serializability is not guaranteed !

local T_1 (site 1)	local T_2 (site 2)	global G-T_1 (both sites)	global G-T_2 (both sites)
$\mathbf{write}(Y_1)$			
		$\mathbf{read}(Y_1)$	
		$\mathbf{write}(X_2)$	
	$\mathbf{read}(X_2)$		
	$\mathbf{write}(Y_2)$		
			$\mathbf{read}(Y_2)$
			$\mathbf{write}(X_1)$
$\mathbf{read}(X_1)$			

Distributed Serializability

- Local serializability at site 1:
  - $G\text{-}T_2 \rightarrow T_1 \rightarrow G\text{-}T_1$ is (unique) serialized order
- Local serializability at site 2:
  - $G\text{-}T_1 \rightarrow T_2 \rightarrow G\text{-}T_2$ is (unique) serialized order
- Thus, globally:
  - $G\text{-}T_1 \rightarrow T_2 \rightarrow G\text{-}T_2 \rightarrow T_1 \rightarrow G\text{-}T_1$ is not a serialized order

$\Longrightarrow$ the global schedule is not serializable

local T_1 (site 1)	local T_2 (site 2)	global $G\text{-}T_1$ (both sites)	global $G\text{-}T_2$ (both sites)
$\textbf{write}(Y_1)$			
		$\textbf{read}(Y_1)$	
		$\textbf{write}(X_2)$	
	$\textbf{read}(X_2)$		
	$\textbf{write}(Y_2)$		
			$\textbf{read}(Y_2)$
			$\textbf{write}(X_1)$
$\textbf{read}(X_1)$			

Distributed Serializability

Another restrictive schedule of transactions:

- every site ensures local serializability
- every global transaction is read-only

Even then:

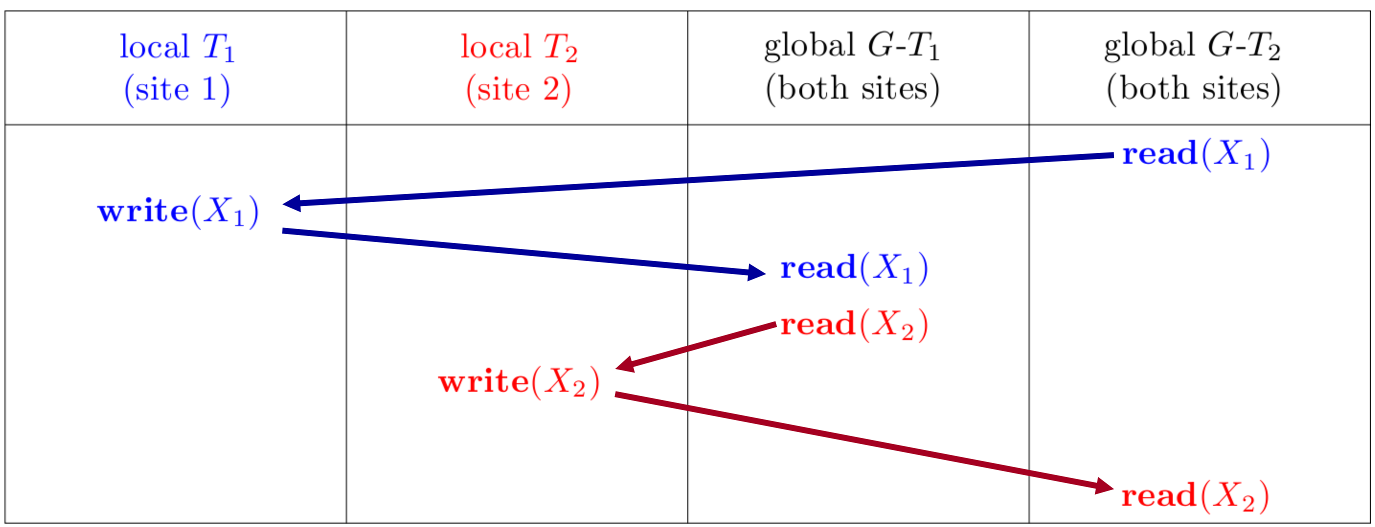
- global serializability is not guaranteed !

local T_1 (site 1)	local T_2 (site 2)	global G-T_1 (both sites)	global G-T_2 (both sites)
			$\mathbf{read}(X_1)$
$\mathbf{write}(X_1)$			
		$\mathbf{read}(X_1)$	
		$\mathbf{read}(X_2)$	
	$\mathbf{write}(X_2)$		
			$\mathbf{read}(X_2)$

Distributed Serializability

- Local serializability at site 1:
  - $G\text{-}T_2 \rightarrow T_1 \rightarrow G\text{-}T_1$ is (unique) serialized order
- Local serializability at site 2:
  - $G\text{-}T_1 \rightarrow T_2 \rightarrow G\text{-}T_2$ is (unique) serialized order
- Thus, globally:
  - $G\text{-}T_1 \rightarrow T_2 \rightarrow G\text{-}T_2 \rightarrow T_1 \rightarrow G\text{-}T_1$ is not a serialized order

$\Longrightarrow$ the global schedule is not serializable



Distributed Serializability

Given a distributed non-serial schedule:

- we can (in principle) test conflict serializability using the precedence graph
- the schedule is serializable if and only if the precedence graph has no directed cycle
  - i.e. same as in centralized schedules

What is the technical problem?

- database is distributed

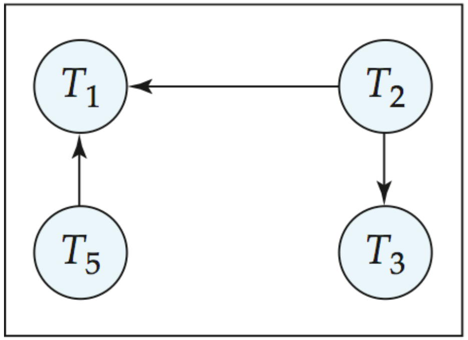
$\Rightarrow$ no site has full information about all (global) conflicts

$\Rightarrow$ even building the precedence graph is not trivial

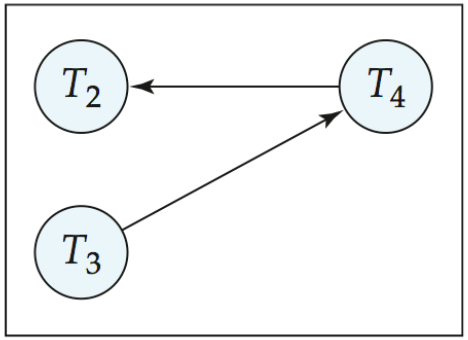
Distributed Serializability

Simple example:

- the local schedule at every site is serializable
  – no directed cycles in local precedence graphs
- but the global schedule is not serializable
  – directed cycle in global precedence graph

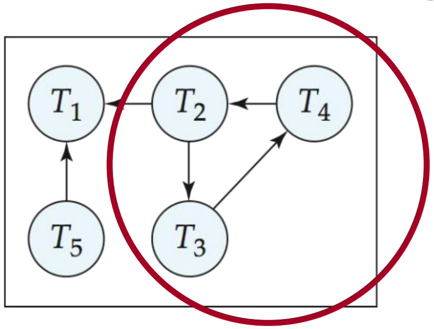


site S_1



site S_2

Local schedules



Global schedule

Distributed Concurrency Control

- How can we ensure serializability in practice?

- If all transactions are local and no replication:
  - all centralized (local) concurrency protocols work
    - two phase locking (2PL); growing / shrinking phase
    - timestamping
  - otherwise we need to extend them

- Distributed protocols for 2PL:
  - Centralized 2PL
  - Primary Copy 2PL
  - Distributed 2PL
  - Majority 2PL

Centralized 2PL

- Centralized 2PL protocol:
  - a single site has a central lock manager (LM)
  - LM maintains all locking information for the DDBMS
- When a transaction is initiated at site S_i :
  - the local transaction coordinator (TC) at S_i is responsible for ensuring consistency throughout the transaction
    - ensure that all copies of an updated item are synchronized
  - if the transaction needs to update a data item x :
    - TC requests from LM a write-lock for every copy of x
    - LM decides to grant the lock or not (standard 2PL rules)
  - similarly for reading a data item x :
    - transaction can read from any copy of x
    - LM decides to grant read-lock or not

Centralized 2PL

- Main idea of centralized 2PL:
  - treat the database as if it were centralized
- Advantages:
  - implementation is easy
    - practically no distributed considerations
  - deadlock detection is simple:
    - build the wait-for graph of DDBMS at central LM
- Disadvantages:
  - bottlenecks
    - overloaded central LM – scalability issues
  - low reliability
    - failure of central LM freezes the whole DDBMS

Primary Copy 2PL

- Primary Copy 2PL protocol:
  - straightforward extension of Centralized 2PL
  - many lock managers (LMs) across the DDBMS
    - each LM responsible for locking a different set of data items
- For every replicated data item x :
  - one copy is chosen as primary copy
  - further copies are slave copies
- When an item x is updated:
  - local TC locates the primary copy of x
  - then sends write-lock request to the appropriate LM
  - only the primary copy of x needs to be locked / updated
  - later the change propagates to slave copies

Primary Copy 2PL

- Protocol is very efficient when:
  - large updates are infrequent (high locality of reference)
  - sites do not always need a most updated version of data
- Bottleneck problems are solved:
  - load is distributed to many Lock Managers (LMs)
- Reliability problems still remain:
  - large degree of centralization
  - failure of one LM freezes some part of DDBMS
  - all primary copies of this LM are inaccessible
- Solution:
  - each LM nominates a backup site

Primary Copy 2PL – Backup site

- When an LM receives an update request:
  - LM sends a copy of request to its backup site (B-LM)

- if LM does not send a quick update notification, then the B-LM:
  - assumes that LM failed $\Longrightarrow$ acts in place of LM
  - sends a copy of request to its own backup (!)
  - notifies everybody that it is the new LM
  - performs all updates of the original LM

- when LM recovers:
  - it notifies everybody that it is again the LM
  - it receives from B-LM the log of updates made

Distributed 2PL

- Distributed 2PL protocol:
  - one lock manager (LM) at every site
    - managing the locks for the data at this site
- If data is not replicated: same as Primary Copy 2PL
- Otherwise Read-One-Write-All (ROWA) rule:
  - only a read-lock at one site that keeps the item
  - a write-lock at every site that keeps the item
    - the copies in these sites must be locked before the update
- How to check whether a write-lock can be granted?
  - not trivial: high communication cost needed
  - requesting site waits for confirmation from all sites that keep the item

Distributed 2PL

Distributed 2PL	Primary Copy 2PL
• high communication costs • always current values	• low communication costs • potentially outdated values • when needed, every value can synchronize with primary

- Majority 2PL:
  - special case of Distributed 2PL
  - write-lock is granted if at least half of sites confirm it

$\Longrightarrow$ lock holder notifies everybody that it has the lock

- A read-lock: can be simultaneously held by many users
- A write-lock: can be held by only one user each time

Why?

Summary of the Lecture

- Distributed Databases and DDBMSs
- Distributed transactions
- Distributed serializability
- Distributed 2PL concurrency control protocols
  - Centralized 2PL
  - Primary Copy 2PL
  - Distributed 2PL
  - Majority 2PL