Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
- Distributed Transaction Management
  - Deadlocks
- Building Distributed Database Systems (RAID)
- Mobile Database Systems
- Privacy, Trust, and Authentication
- Peer to Peer Systems

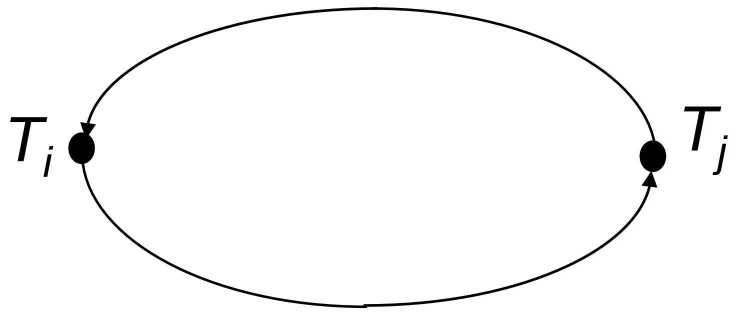
Useful References

- Textbook *Principles of Distributed Database Systems*,

  Chapter 11.6
- J. Gray, R. Lorie, G. Putzolu, I. Traiger, *Granularity of Locks and Degrees of Consistency in a Shared Data Base*, Modelling in Data Base Management Systems, G.M. Nijssen (ed). North Holland Publishing Company, 1976.

Deadlock

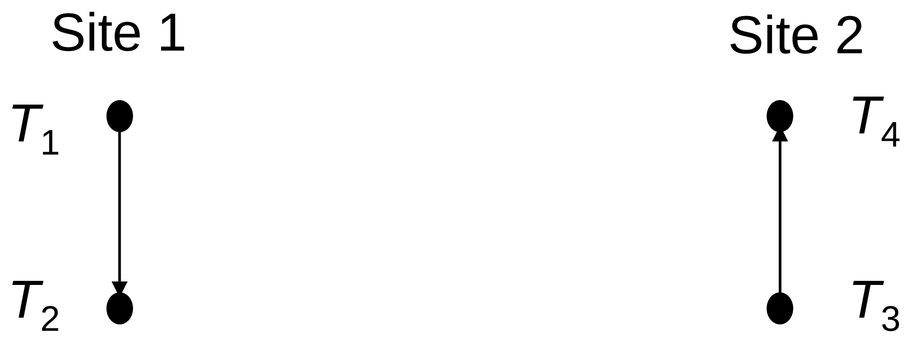
- A transaction is deadlocked if it is blocked and will remain blocked until there is intervention.
- Locking-based CC algorithms may cause deadlocks.
- TO-based algorithms that involve waiting may cause deadlocks.
- Wait-for graph
  - If transaction T_i waits for another transaction T_j to release a lock on an entity, then $T_i \rightarrow T_j$ in WFG.



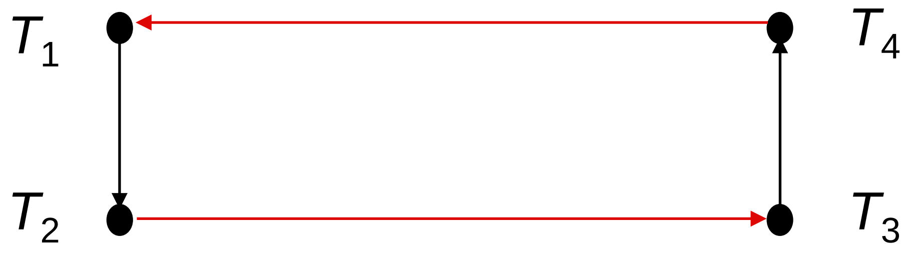
Local versus Global WFG

Assume T_1 and T_2 run at site 1, T_3 and T_4 run at site 2. Also assume T_3 waits for a lock held by T_4 which waits for a lock held by T_1 which waits for a lock held by T_2 which, in turn, waits for a lock held by T_3.

Local WFG



Global WFG



Deadlock Management

- Ignore
  - Let the application programmer deal with it, or restart the system
- Prevention
  - Guaranteeing that deadlocks can never occur in the first place. Check transaction when it is initiated. Requires no run time support.
- Avoidance
  - Detecting potential deadlocks in advance and taking action to insure that deadlock will not occur. Requires run time support.
- Detection and Recovery
  - Allowing deadlocks to form and then finding and breaking them. As in the avoidance scheme, this requires run time support.

Deadlock Prevention

- All resources which may be needed by a transaction must be predeclared.
  - The system must guarantee that none of the resources will be needed by an ongoing transaction.
  - Resources must only be reserved, but not necessarily allocated a priori
  - Unsuitability of the scheme in database environment
  - Suitable for systems that have no provisions for undoing processes.
- Evaluation:
  - – Reduced concurrency due to preallocation
  - – Evaluating whether an allocation is safe leads to added overhead.
  - – Difficult to determine (partial order)
  - + No transaction rollback or restart is involved.

Deadlock Avoidance

- Transactions are not required to request resources a priori.
- Transactions are allowed to proceed unless a requested resource is unavailable.
- In case of conflict, transactions may be allowed to wait for a fixed time interval.
- Order either the data items or the sites and always request locks in that order.
- More attractive than prevention in a database environment.

Deadlock Avoidance – Wait-Die & Wound-Wait Algorithms

WAIT-DIE Rule: If T_i requests a lock on a data item which is already locked by T_j, then T_i is permitted to wait iff $ts(T_i)<ts(T_j)$. If $ts(T_i)>ts(T_j)$, then T_i is aborted and restarted with the same timestamp.

- **if** $ts(T_i)<ts(T_j)$ **then** T_i waits **else** T_i dies
- non-preemptive: T_i never preempts T_j
- prefers younger transactions

WOUND-WAIT Rule: If T_i requests a lock on a data item which is already locked by T_j , then T_i is permitted to wait iff $ts(T_i)>ts(T_j)$. If $ts(T_i)<ts(T_j)$, then T_j is aborted and the lock is granted to T_i.

- **if** $ts(T_i)<ts(T_j)$ **then** T_j is wounded **else** T_i waits
- preemptive: T_i preempts T_j if it is younger
- prefers older transactions

Deadlock Detection

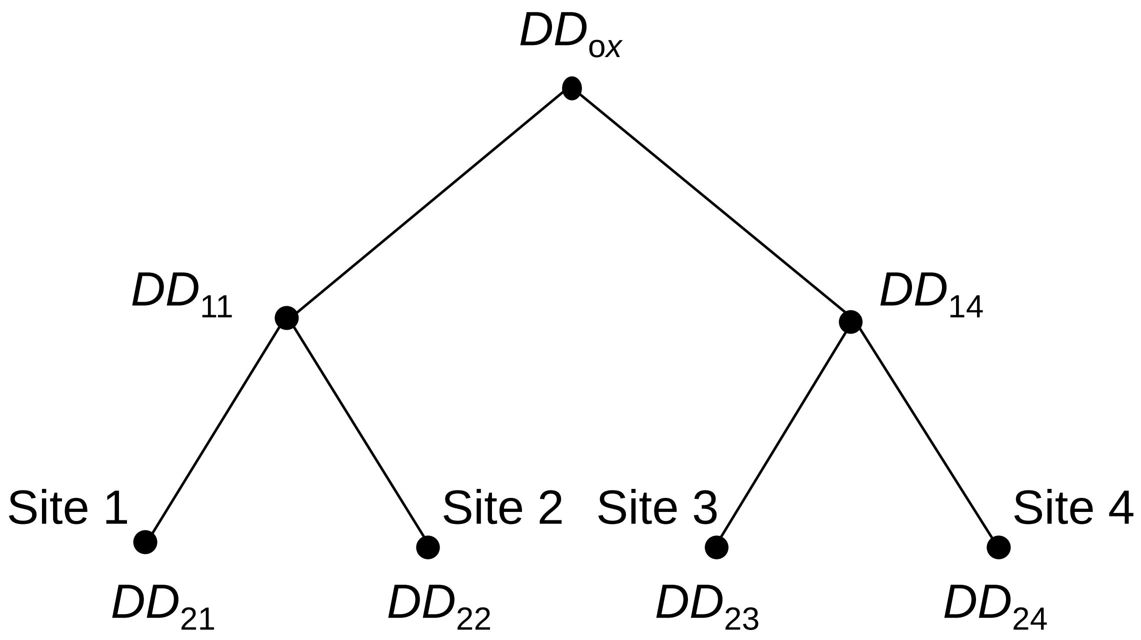
- Transactions are allowed to wait freely.
- Wait-for graphs and cycles.
- Topologies for deadlock detection algorithms
  - Centralized
  - Distributed
  - Hierarchical

Centralized Deadlock Detection

- One site is designated as the deadlock detector for the system. Each scheduler periodically sends its local WFG to the central site which merges them to a global WFG to determine cycles.
- How often to transmit?
  - Too often ⇒ higher communication cost but lower delays due to undetected deadlocks
  - Too late ⇒ higher delays due to deadlocks, but lower communication cost
- Would be a reasonable choice if the concurrency control algorithm is also centralized.
- Proposed for Distributed INGRES

Hierarchical Deadlock Detection

Build a hierarchy of detectors



Distributed Deadlock Detection

- Sites cooperate in detection of deadlocks.
- One example:
  - The local WFGs are formed at each site and passed on to other sites. Each local WFG is modified as follows:
    - Since each site receives the potential deadlock cycles from other sites, these edges are added to the local WFGs
    - The edges in the local WFG which show that local transactions are waiting for transactions at other sites are joined with edges in the local WFGs which show that remote transactions are waiting for local ones.
  - Each local deadlock detector:
    - looks for a cycle that does not involve the external edge. If it exists, there is a local deadlock which can be handled locally.
    - looks for a cycle involving the external edge. If it exists, it indicates a potential global deadlock. Pass on the information to the next site.