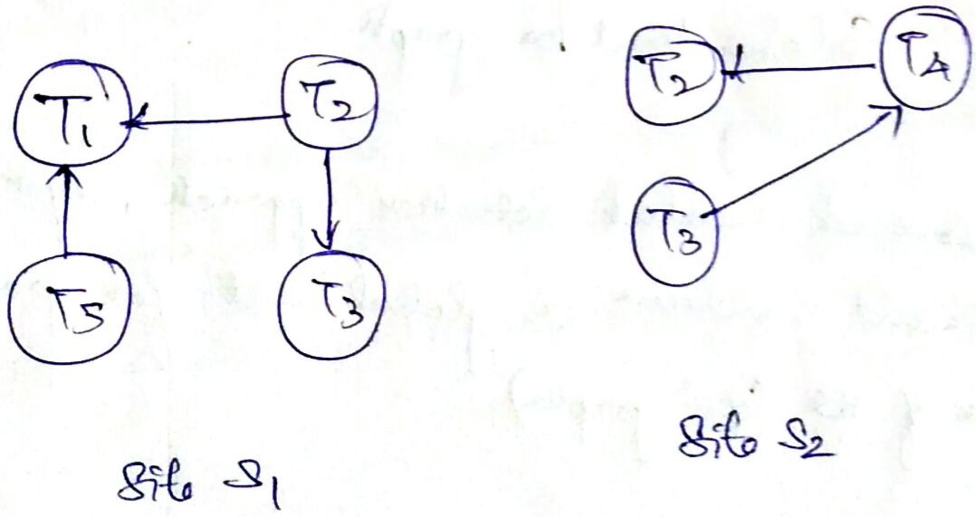
IT18303 - Database Systems

CAT-03

Rosttaan S
IT-B
2127200801072

8) (b)

* Deadlock prevention may result in unnecessary waiting and roll back
* The main problem in distributed system is deciding how to maintain wait-for-graph
* Common techniques for dealing with this issue require that each site keep a local wait-for graph
* The nodes of graph corresponds to all transactions that are currently either holding or requesting any of the items local to the site.

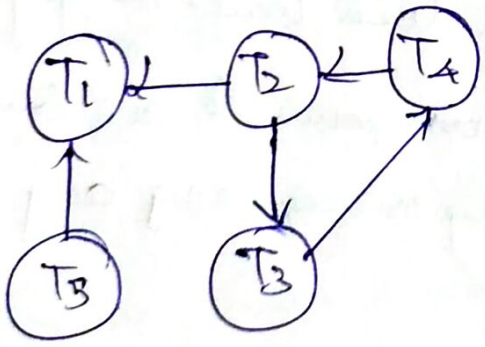


Local wait-for graphs

When a transaction T_i on site S_1 needs a resource in site S_2, it sends a request message to site S_2.

If the resource is held by transaction T_j, the system inserts an edge $T_i \rightarrow T_j$ in the local wait-for graph of site S_2

* In the above diagram, each wait-for graph is acyclic, nevertheless a deadlock exists in system because the union of local wait-for graphs contains a cycle.



Global wait for graph

* In centralized deadlock detection approach, system constructs and maintains a global wait-for graph (union of all local graphs)

The real graph describes the real but unknown state of system at any instance in time.

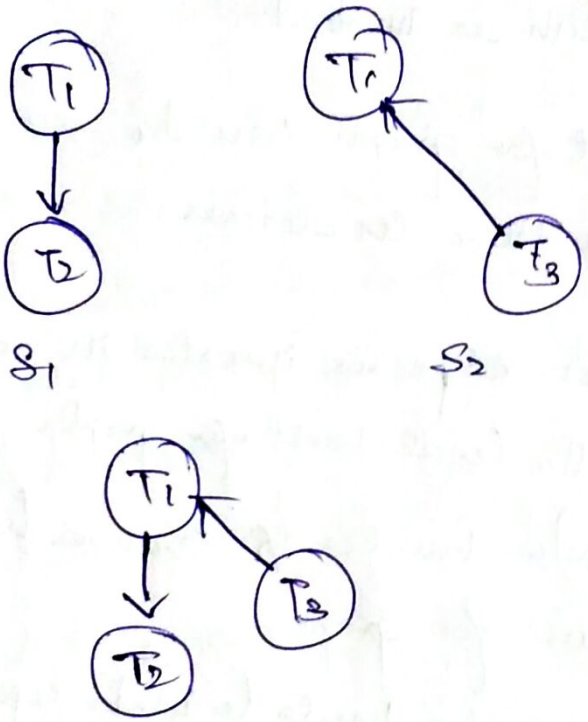
* The constructed graph is an approximation generated by the controller
* Controller must generate the constructed graph, whenever detection algorithm is invoked
* The global wait for graph can be maintained or updated based on these conditions

  * Whenever a new edge is inserted in or removed from one of the local-wait for graphs
  * Periodically, when number of changes have occured in a local-wait for graph
  * Whenever coordinator needs to invoke cycle detection algorithm

* When the coordinator invokes deadlock detection algorithm, it searches its global graph
* If it finds a cycle, selects a victim to be rolled backed.

* The Co-ordinator must modify all the sites, that particular transaction has been selected as victim.
* the sites, in turn, roll back the victim transactions.

* the scheme may produce unnecessary rollbacks if false cycles exist in global wait-for graph.



* Suppose that T_2 releases the resource that is holding in site S_1, resulting in deletion of edge $T_1 \rightarrow T_2$ in S_1.
* Transaction T_2 then requests a resource held by T_3 at site S_2, resulting in addition of edge $T_2 \rightarrow T_3$ in S_2.

→ If the insert $T_2 \rightarrow T_3$ message from S_2 arrives before the remove $T_1 \rightarrow T_2$ message from S_1, the co-ordinator may discover false cycle $T_1 \rightarrow T_2 \rightarrow T_3$ after the insert.

→ Deadlock recovery may be initiated, although no deadlock has occurred

→ A deadlock has indeed occurred and a victim as been picked, while one of transactions was aborted for reasons unrelated to deadlock.

→ Suppose that S_1 decides to abort T_2.

→ At the same time the co-ordinator has discovered a cycle, and has picked T_3 as victim.

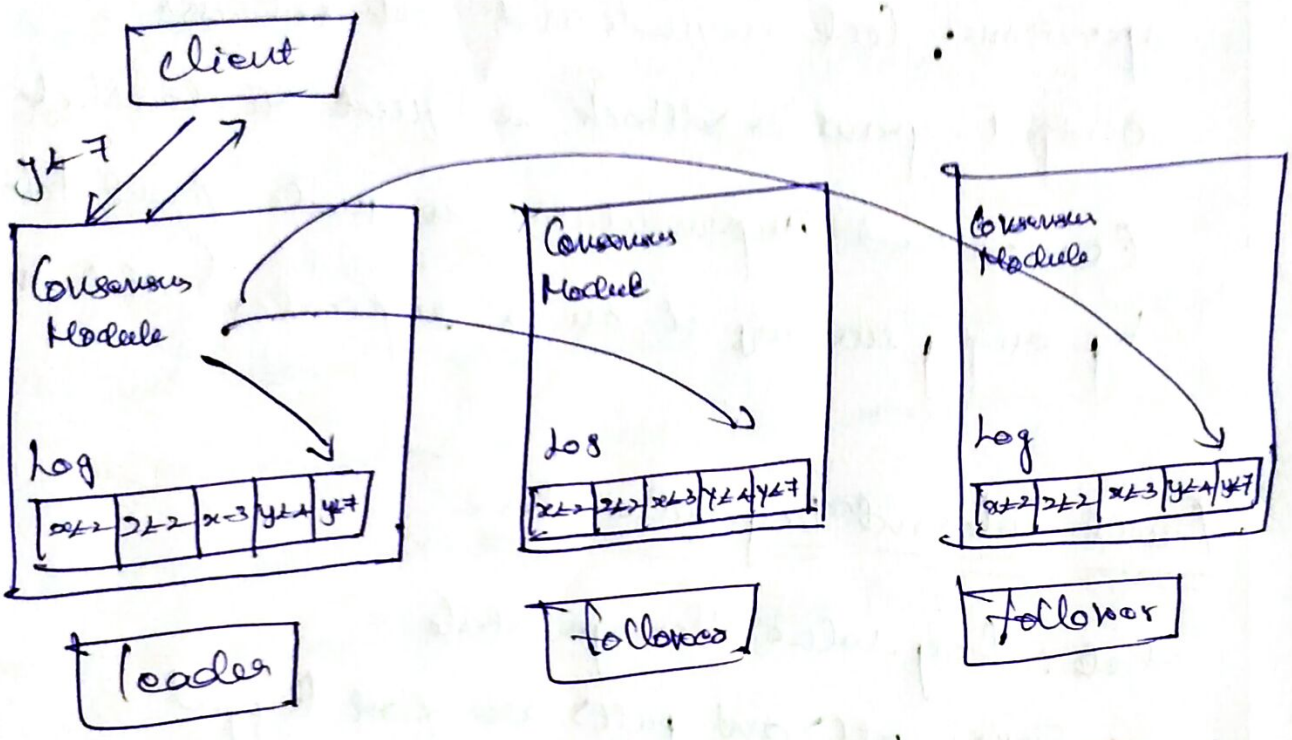
→ Both T_2 and T_3 are now rolled back although T_3 needs to wait locked.

7)
(a) Fault tolerant Services Using Replicated State Machines

* Key requirement is make a service fault tolerant.
* State machines are powerful approach to creating such services
* A state machine has a stored state and receive inputs
* Make state transitions on each input and may output some results
* Transitions and outputs must be deterministic
* A replicated state machine is a state machine that is replicated on multiple nodes.
* All replicas must get exactly same inputs.
* Even in some of the nodes fail, state and output can be obtained from other nodes.

Replicated State Machine

* Replicated state machine based on replicated log
* Example Commands assign values to variables



Leader declares a log record committed after it is replicated at a majority of nodes.

Update of state machines at each replica happens only after log record has been committed.

Uses of Replicated State Machines

Fault-tolerant lock manager

State: lock table

operations: lock requests and lock releases

output: grant or rollback requests on deadlock

Centralized implementation is made fault tolerant by simply running it on a replicated state machine.

Fault tolerant key-value store

State: key-value storage state

operations: get() and put() are first logged

* operations are executed when the log record is in committed state.

* Google spanner uses replicated state machine to implement key-value store.

* Data is partitioned, and each partition is replicated across multiple nodes

* Replicas of a partition form Paxos group with one node as leader.

* operations initiated at leader, and replicated to other nodes

6) (b) Challenges involved in ensuring data Consistency

① Distributed Concurrency control:

* When data is shared among various site, in case if data is updated or modified in one site it has to be reflected in others.
* Only then database is said to consistent.
* So, this integrity of database is maintained by specifying the synchronization of access to distributed database to manage concurrency different locking techniques uses based on mutual exclusion of access to data.

Replication Control:

* Data might be inconsistent if the whole database or a percentage of it is copied and copies are stockpiled at dissimilar sites
* Having more than one copy of a database, the issue is concerning the command uniformity of the copies ensuring with all copies are identical schema and data.
* We use replication technique for this purpose.

* Data might become inconsistent if multiple users request data at same time.
* This leads to Data Redundancy.
* To avoid this we use locking protocols and deadlock handling.

Transparent Management:

* The major problem is when data located in numerous locations and number of users are used that database
* So, transparent management of data is important to maintain the integrity of distributed database.

Replication Latency:

* In asynchronous application, there will short lag between changes to the source and delivery of those changes to target.

1) A disk is failed for 100 hours every 100,000 hours. So in order to make it to fail, it we need $(100,000)^2 / (2*100)$

2) B+ Trees can be used to locate block containing the search key.

We choose B+ tree because it is a self-balancing tree data structure that maintains data and allows searches, sequential access, insertions and deletions.

3) Advantages:

* Complex structures can be implemented through DBMS, thus increasing performance.
* Simply the DBMS for backups
* Data Retrieval, Editing

Disadvantages:

* increases the size and complexity of DBMS
* May Lead to overloading
* Data Redundancy, inconsistency

4) Spatial database is used.

Using spacial database indexing can be done and given business rules can be maintained.

5) A distributed database is a database in which data is stored across different physical locations.

& to develop map we use distributed database

Also use geospatial database