SANTHOSH.M
IT-B
212720080 1078

IT 18305 - DATABASE SYSTEMS

CAT-3 (III rd Semester)

11.02.22

Part - A

3) Advantages for storing multiple relations in a single file:

* Complex structures can be implemented through the DBMS, thus increasing performance.

Disadvantages of storing multiple relations in a single file:

* Increases the size and complexity of the DBMS.

D) For the two disk mirrored case, we assume A disk and B disk. In order to lose data, A and B need to be failed at the same time. If A is already failed and within 1000000 hours B disk will fail, then data will be lost. The other case is B is already failed and within 100,000 hours A will fail, and then data will be lost.

If the mean time to failure of a single disk is 100000 hours and the mean time to repair is 10 hours then the mean time to data loss is $(100,000^2)/2*100$.

5) Map database.

Map database management systems are software programs designed to efficiently store and recall spatial information.

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.

Part-B

6)b) Challenges in Maintaining Data Consistency

Data discrepancy occurs when the data in the target database deviates from the source database. The extent to which the data deviates depends on various factors. Even when using products that replicate ~~data~~ data reliably, such as Oracle, there remains potential causes of data discrepancy. ~~Of the good of the~~

Migration Errors

Different kinds of migration tools are employed to facilitate the initial load of the target databases before replication can begin. Differences in configuration for handling data by the migration tools and replication products can ~~exist~~ result in data discrepancies.

Lift & Shift workload ~~on~~ to Cloud

Since the world is moving towards the cloud, the lift & shift of database from on premises to cloud is the need of today's IT world.

Differences in Source and Target

Diff in source and target database configuration, for example different encodings, locales, endianess, of database versions can cause subtle discrepancies to happen during migration and replications.

Instantiation Errors

Before migration or replication can begin, the target databases will need to be instantized with the correct schema and constraints.

Configuration Errors

Improper and unintended configuration of replication products can cause discrepancies.

Gaps in Replication

Although replication is enabled between source and target databases and is working perfectly well, there are instances where data inserted on the source will not be replaced.

Replication latency

With asynchronous replication, there will be a short lag in between changes to the source database and delivery of those changes to the target.

User errors

Often target databases are created to offload query processing from the soure database. This enables rich operational reporting without impacting the applications running.

Application errors

Applications that use target ~~dat~~ data bases can potentially change data due to faulty logic as well as, application upgrades.

The technology requirements for manging data consistency are:

* High speed, low impact data comparisons.
* Support for heterogenous databases.
* Capability for handling large data volumes.
* Flexible options for manging data comparisons.
* Support for live databases with contantly changing data.
* Minimally intrusive.
* Comparison of only changed data in continuous replication
* Comparison of huge table through automated & manual partitioning.
* Data comparison reports for auditing purposes
* Zero downtime of source and target systems.
* Capability to identify data ~~inconsist~~ inconsistency.
* Low impact on hardware and network resources.
* Flexible reporting for varying & roles and access levels.
* Data Security.
* Easy to use, understand, configure, deploy and diagnose.

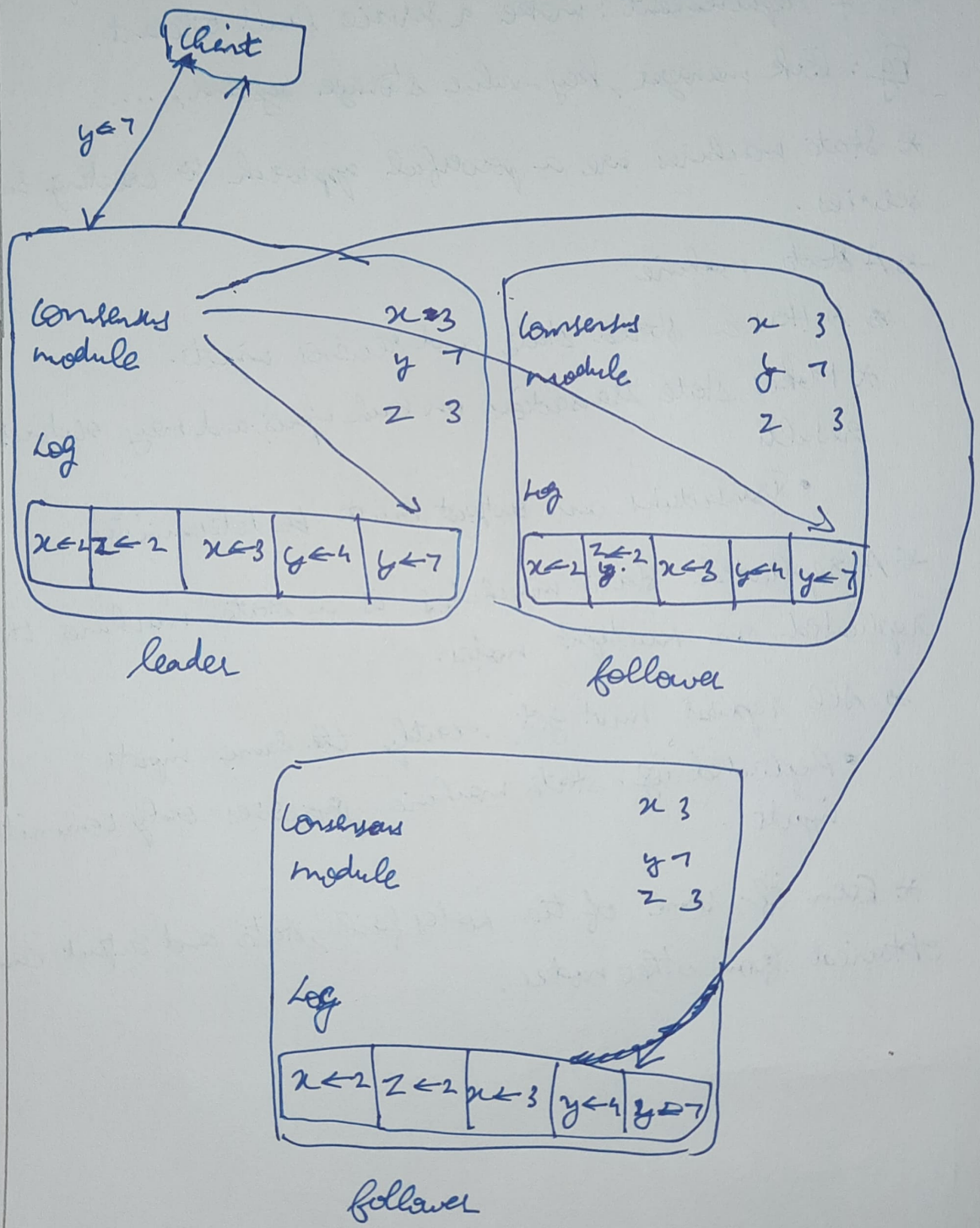
Part-C

7a) **Fault Tolerant Services using Replicated State Machines**

* Key requirement: make a service fault tolerant.
  Eg: lock manager, key-value storage system, ...
* State machines are a powerful approach to creating such services.
* A state machine
  * Has a stored state, and receives inputs.
  * Makes state transactions on each input, and may outputs some results
    - Transactions and output must be deterministic
* A replicated state machines is a state machine that is replicated on multiple nodes.
  * All replicas must get exactly the same inputs
    - Replicated log. State machine processes only committed inputs.
  * Even if 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 decides log record committed after it is replicated on a majority of nodes.

Uses of Replicated State Machines

* Replicated state machines can be used to implement wide variety of services.
  * Inputs can specify operations with parameters.
  * But operations must be deterministic.
  * Results of operation can be sent to from any replica.
    - Gets executed only when log record is committed in replicated log.

* Example : 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.

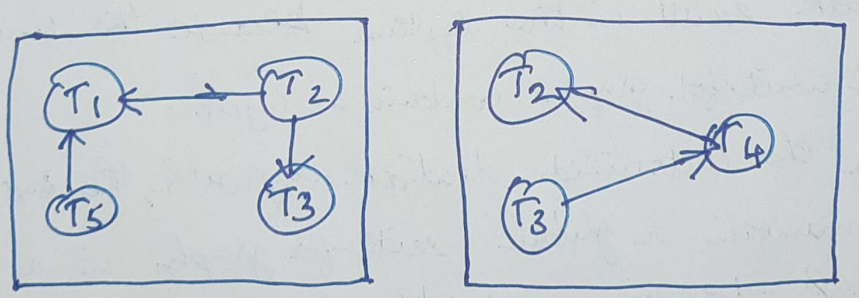
* Fault tolerant key value-store
  * State : key value storage state
  * Operations : get() and put() are first logged.

* Google Spanner uses replicated state machine to implement key value store.
  * Data is partitioned, and each partition is replicated across multiple nodes.

8)b) Deadlock Handling

The deadlock ~~handling~~ prevention and deadlock-detection algorithms can be used in a distributed system, provided that modifications are made. For example, we can use the tree protocol by defining a global tree among the system data items. Similarly, the timestamp-ordering approach could be directly applied to a distributed environment.

Deadlock prevention may result in unnecessary waiting and rollback. Furthermore, certain deadlock-prevention techniques may require more sites to be involved in the execution of a transaction than would otherwise be the case. If we allow deadlocks to occur and rely on deadlock detection, the main problem in a distributed system is deciding how to maintain the wait-for graph.



site S_1 site S_2

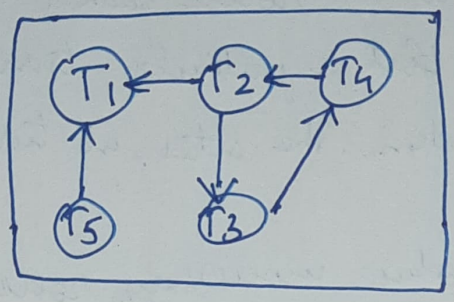
Common techniques for dealing with the issue require that each site keeping a local wait-for graph. The nodes of the graph correspond to all the transactions (local as well as non-local) that are currently either holding or requesting any of the items local to that site. For example

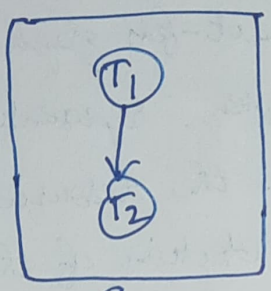
the above diagram depicts a system consisting of two sites, each maintaining its local wait-for graph. Note that transactions T_2 and T_3 appear in both graphs, indicating that the transactions have requested items at both sites.

These local wait-for graphs are constructed in the usual manner for local transactions and data items. 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_i, the system inserts an edge $T_i \rightarrow T_j$ in the local wait-for graph of site S_2. Clearly, if any local wait-for graphs has a cycle, dealock has occured. On the other hand, the fact that there are no cycles in any of the local wait-for graphs does not mean that there are no cycles in any of the local wait-for graphs does not mean there are no dealocks. Each wait-for graph is acyclic; nevertheless, a deadlock exists in the system because the union of the local wait-for graphs contains a cycle.

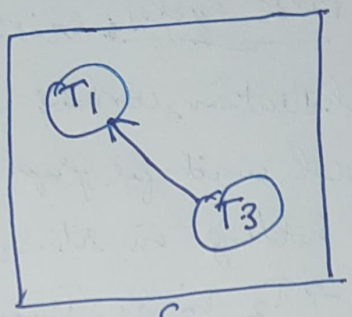
In the centralized deadlock approach, the system constructs and maintain a global wait-for graph in a single site: the deadlock-detection coordinator. Since there is communication delay in the system, we must distinguish between the two types of wait-for graphs. The real graph describes the real but unknown state of the system at any instance in time, as would be seen by an omniscient observer. The constructed graph is an

approximation generated by the controller during the execution of the controller's algorithm.

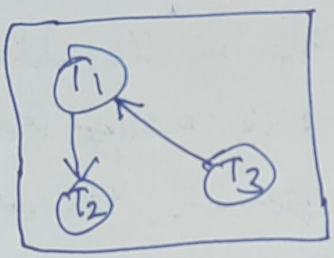




S1



S2



Coordinator

False cycles in the global wait-for graph

The global wait-for graph can be reconstructed or updated under these conditions :-

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

When the coordinator invokes the deadlock-detection algorithm, it searches its global graph. If it finds a cycle, it selects a victim to be rolled back. The coordinator must notify all the sites that a particular transaction has been selected as victim. The sites, in turn, roll back the victim transaction.

This scheme may produce unnecessary rollbacks if:

* False cycles exist in the global wait-for graph. As an illustration, consider a snapshot of the system represented by the local wait-for graphs. Suppose T_2 releases the resource that it is holding in site S_1, resulting in the deletion of the 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 the addition of the 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 coordinator may discover the false cycle $T_1 \rightarrow T_2 \rightarrow T_3$ after the insert. Deadlock recovery may be initiated.

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