Chapters 21-23 : Distributed Databases

Sistemas de Bases de Dados 2019/20

Capítulo refere-se a: Database System Concepts, 7th Ed

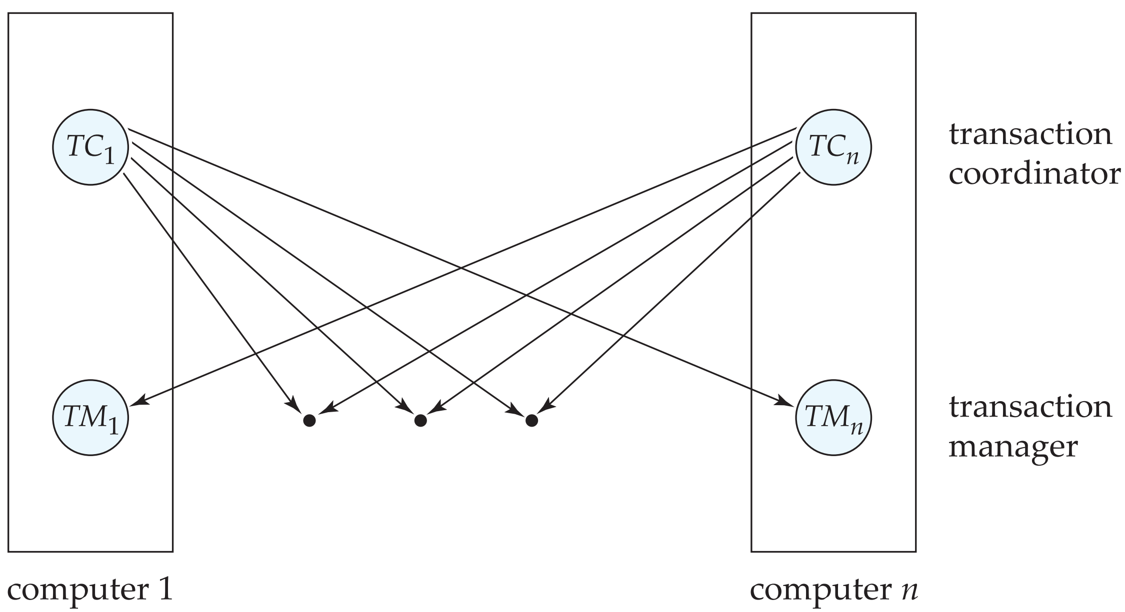
Distributed Transactions

Distributed Transactions

- **Local transactions**
  - Access/update data at only one database
- **Global transactions**
  - Access/update data at more than one database
- Key issue: how to ensure ACID properties for transactions in a system with global transactions spanning multiple database

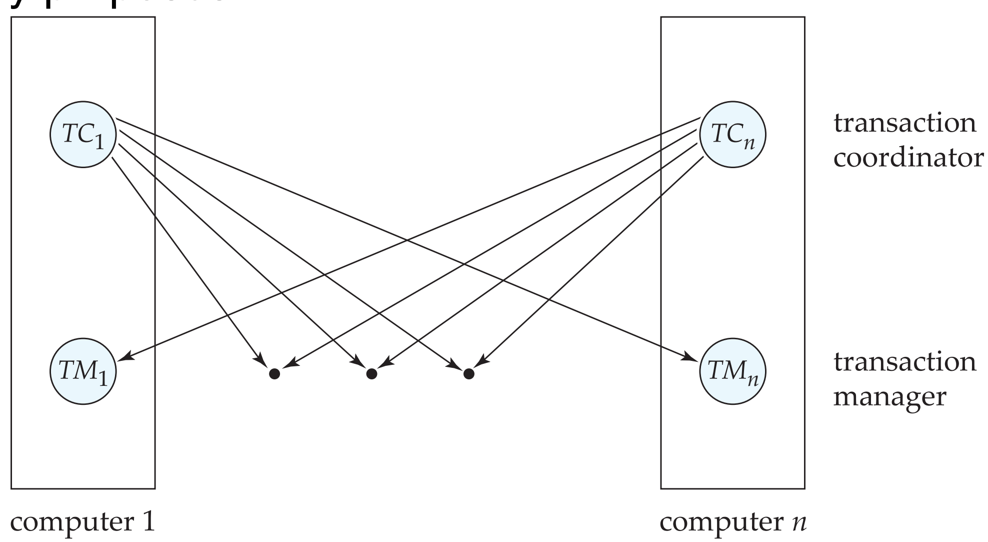
Distributed Transactions

- Transaction may access data at several sites.
  - Each site has a local **transaction manager**
  - Each site has a **transaction coordinator**
    - Global transactions submitted to any transaction coordinator



Distributed Transactions

- Each transaction coordinator is responsible for:
  - Starting the execution of transactions that originate at the site.
  - Distributing subtransactions at appropriate sites for execution.
  - Coordinating the termination of each transaction that originates at the site
    - **transaction must be committed at all sites or aborted at all sites.**
- Each local transaction manager is responsible for:
  - Maintaining a log for recovery purposes
  - Coordinating the execution and commit/abort of the transactions executing at that site.



System Failure Modes

- Failures unique to distributed systems:
  - Failure of a site.
  - Loss of messages
    - Handled by network transmission control protocols such as TCP-IP
  - Failure of a communication link
    - Handled by network protocols, by routing messages via alternative links
  - **Network partition**
    - A network is said to be **partitioned** when it has been split into two or more subsystems that lack any connection between them
      - Note: a subsystem may consist of a single node
- Network partitioning and site failures are generally indistinguishable.

Commit Protocols

- Commit protocols are used to **ensure atomicity** across sites
  - a transaction which executes at multiple sites must either be committed at all the sites or aborted at all the sites.
    - cannot have transaction committed at one site and aborted at another
- The *two-phase commit* (2PC) protocol is widely used
- *Three-phase commit* (3PC) protocol avoids some drawbacks of 2PC, but is more complex
- *Consensus protocols* solve a more general problem, but can be used for atomic commit
  - More on these later
- These protocols assume **fail-stop** model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.

Two Phase Commit Protocol (2PC)

- Execution of the protocol is initiated by the coordinator after the last step of the transaction has been reached.
- The protocol involves all the local sites at which the transaction executed
- Protocol has two phases
- Let T be a transaction initiated at site S_i, and let the transaction coordinator at S_i be C_i

Phase 1: Obtaining a Decision

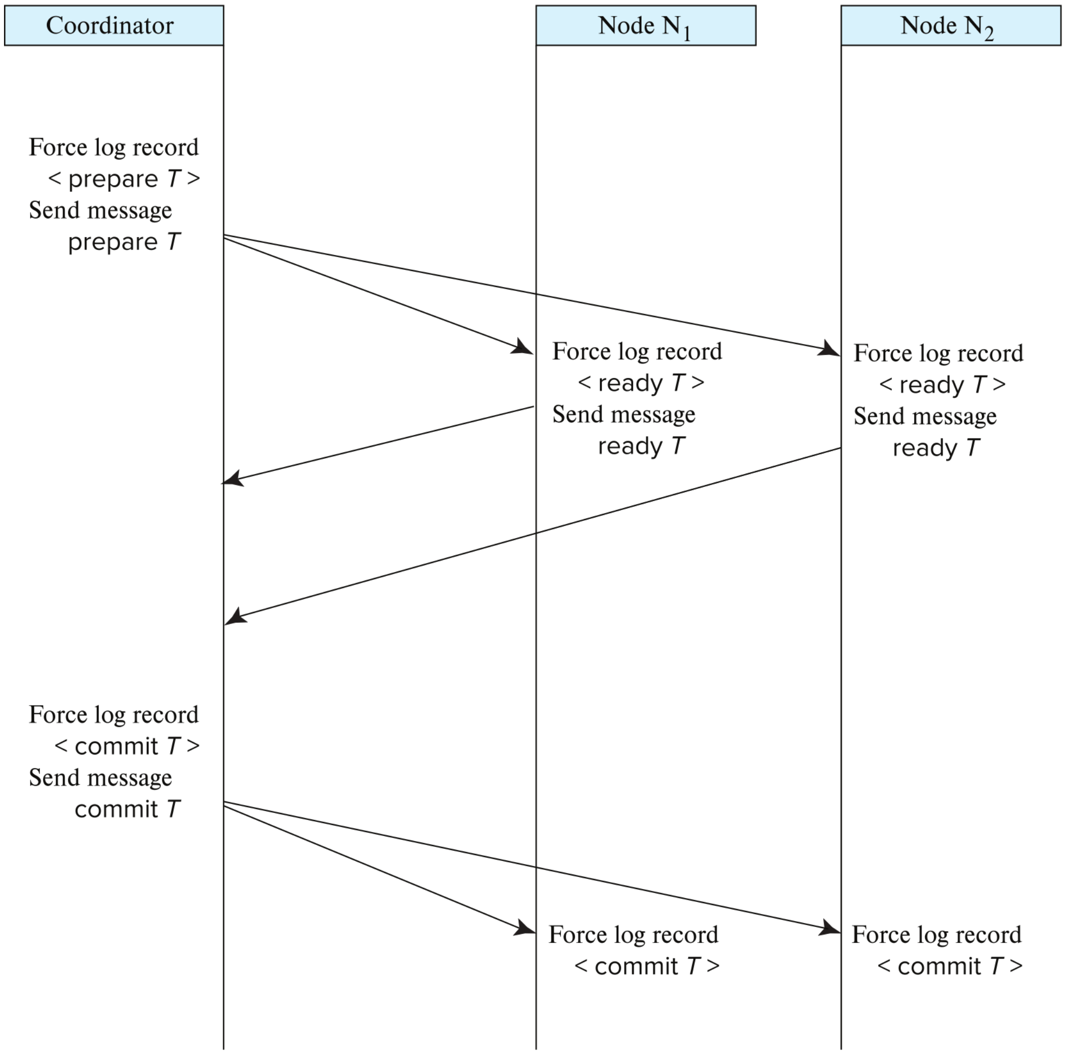
- Coordinator asks all participants to *prepare* to commit transaction T_i.
  - C_i adds the records <**prepare** *T*> to the log and forces log to stable storage
  - sends **prepare** *T* messages to all sites at which *T* executed
- Upon receiving message, transaction manager at site determines if it can commit the transaction
  - if not, add a record <**no** *T*> to the log and send **abort** *T* message to C_i
  - if the transaction can be committed, then:
    - add the record <**ready** *T*> to the log
    - force *all records* for *T* to stable storage
    - send **ready** *T* message to C_i

  Transaction is now in ready state at the site

Phase 2: Recording the Decision

- T can be committed if C_i received a **ready** T message from all the participating sites: otherwise T must be aborted.
- Coordinator adds a decision record, <**commit** T> or <a**bort** T>, to the log and forces record onto stable storage. Once the record stable storage it is irrevocable (even if failures occur)
- Coordinator sends a message to each participant informing it of the decision (commit or abort)
- Participants take appropriate action locally.

Two-Phase Commit Protocol



Handling of Failures - Site Failure

When site S_k recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- Log contain <**commit** *T*> record: site executes **redo** (*T*)
- Log contains <**abort** *T*> record: site executes **undo** (*T*)
- Log contains <**ready** *T*> record: site must consult C_i to determine the fate of *T*.
  - If *T* committed, **redo** (*T*)
  - If *T* aborted, **undo** (*T*)
- The log contains no control records concerning *T* implies that S_k failed before responding to the **prepare** *T* message from C_i
  - since the failure of S_k precludes the sending of such a response C_i must abort *T*
  - S_k must execute **undo** (*T*)

Handling of Failures- Coordinator Failure

- If coordinator fails while the commit protocol for *T* is executing, then participating sites must decide on *T*' s fate:
  1. If an active site contains a <**commit** *T*> record in its log, then *T* must be committed.
  2. If an active site contains an <**abort** *T*> record in its log, then *T* must be aborted.
  3. If some active participating site does not contain a <**ready** *T*> record in its log, then the failed coordinator C_i cannot have decided to commit *T*. So, it can abort *T*.
  4. If none of the above cases holds, then all active sites must have a <**ready** *T*> record in their logs, but no additional control records (such as <**abort** *T*> of <**commit** *T*>). In this case active sites must wait for C_i to recover, to find decision.
- **Blocking problem**: active sites may have to wait for failed coordinator to recover.

Handling of Failures - Network Partition

- If the coordinator and all its participants remain in one partition, the failure has no effect on the commit protocol.
- If the coordinator and its participants belong to several partitions:
  - Sites that are not in the partition containing the coordinator think the coordinator has failed and execute the protocol to deal with failure of the coordinator.
    - No harm results, but sites may still have to wait for decision from coordinator.
- The coordinator and the sites are in the same partition as the coordinator think that the sites in the other partition have failed and follow the usual commit protocol.
    - Again, no harm results

Recovery and Concurrency Control

- **In-doubt transactions** have a <**ready** *T*>, but neither a <**commit** *T*>, nor an <**abort** *T*> log record.
- The recovering site must determine the commit-abort status of such transactions by contacting other sites; this can slow and potentially block recovery.
- Recovery algorithms can note lock information in the log.
  - Instead of <**ready** *T*>, write out <**ready** *T*, *L*> *L* = list of locks held by *T* when the log is written (read locks can be omitted).
  - For every in-doubt transaction *T*, all the locks noted in the <**ready** *T*, *L*> log record are reacquired.
- After lock reacquisition, transaction processing can resume; the commit or rollback of in-doubt transactions is performed concurrently with the execution of new transactions.

Avoiding Blocking During Consensus

- Blocking problem of 2PC is a serious concern
- Idea: involve multiple nodes in decision process, so failure of a few nodes does not cause blocking as long as majority don't fail
- More general form: **distributed consensus problem**
  - A set of *n* nodes need to agree on a decision
  - Inputs to make the decision are provided to all the nodes, and then each node votes on the decision
  - The decision should be made in such a way that all nodes will "learn" the same value for the even if some nodes fail during the execution of the protocol, or there are network partitions.
  - Further, the distributed consensus protocol should not block, as long as a majority of the nodes participating remain alive and can communicate with each other

Three-Phase Commit

- Assumptions:
  - No network partitioning
  - At any point, at least one site must be up.
  - At most K sites (participants as well as coordinator) can fail
- Phase 1: Obtaining Preliminary Decision: Identical to 2PC Phase 1.
  - Every site is ready to commit if instructed to do so
- Phase 2 of 2PC is split into 2 phases, Phase 2 and Phase 3 of 3PC
  - In phase 2 coordinator makes a decision as in 2PC (called the **pre-commit** decision) and records it in multiple (at least K) sites
  - In phase 3, coordinator sends commit/abort message to all participating sites,
- Under 3PC, knowledge of pre-commit decision can be used to commit despite coordinator failure
  - Avoids blocking problem as long as < K sites fail
- Drawbacks:
  - higher overheads
  - assumptions may not be satisfied in practice

Concurrency Control

- Modify concurrency control schemes for use in distributed environment.
- We assume that each site participates in the execution of a commit protocol to ensure global transaction atomicity.
- We assume all replicas of any item are updated

Single-Lock-Manager Approach

- In the **single lock-manager** approach, lock manager runs on a *single* chosen site, say S_i
  - All lock requests sent to central lock manager
- The transaction can read the data item from *any* one of the sites at which a replica of the data item resides.
- Writes must be performed on all replicas of a data item
- Advantages of scheme:
  - Simple implementation
  - Simple deadlock handling
- Disadvantages of scheme are:
  - Bottleneck: lock manager site becomes a bottleneck
  - Vulnerability: system is vulnerable to lock manager site failure.

Distributed Lock Manager

- In the **distributed lock-manager** approach, functionality of locking is implemented by lock managers at each site
  - Lock managers control access to local data items
  - Locking is performed separately on each site accessed by transaction
    - Every replica must be locked and updated
    - But special protocols may be used for replicas (more on this later)
- Advantage: work is distributed and can be made robust to failures
- Disadvantage:
  - Possibility of a global deadlock without local deadlock at any single site
  - Lock managers must cooperate for deadlock detection

Primary Copy

- Choose one replica of data item to be the **primary copy**.
  - The site containing the chosen replica is called the **primary site** for that data item
  - Different data items can have different primary sites
- When a transaction needs to lock a data item *Q*, it requests a lock at the primary site of *Q*.
  - Implicitly gets lock on all replicas of the data item
- Benefit
  - Concurrency control for replicated data handled similarly to unreplicated data - simple implementation.
- Drawback
  - If the primary site of *Q* fails, *Q* is inaccessible even though other sites containing a replica may be accessible.

Majority Protocols

- Local lock manager at each site administers lock and unlock requests for data items stored at that site.
- When a transaction wishes to lock an unreplicated data item *Q* residing at site S_i, a message is sent to S_i 's lock manager.
  - If *Q* is locked in an incompatible mode, then the request is delayed until it can be granted.
  - When the lock request can be granted, the lock manager sends a message back to the initiator indicating that the lock request has been granted.
- In case of replicated data
  - If *Q* is replicated at *n* sites, then a lock request message must be sent to more than half of the *n* sites in which *Q* is stored.
  - The transaction does not operate on *Q* until it has obtained a lock on a majority of the replicas of *Q*.
  - When writing the data item, transaction performs writes on *all* replicas.

Majority Protocols

- Benefit
  - Can be used even when some sites are unavailable
    - See the book, for details about how to handle writes in the presence of site failure
- Drawback
  - Requires $2(n/2 + 1)$ messages for handling lock requests, and $(n/2 + 1)$ messages for handling unlock requests.
  - Potential for deadlock even with single item (when the total number of replicas is even):
    - Consider Q replicated in sites S1 to S4, and transactions T1 and T2 requiring Q
    - Further consider that T1 acquired the lock in S1 and S2, and T2 at S3 and S4
    - None gets the majority - deadlock

Biased Protocol

- Local lock manager at each site as in majority protocol. However, requests for shared locks are handled differently than requests for exclusive locks.
  - **Shared locks**. When a transaction needs to lock data item *Q*, it simply requests a lock on *Q* from the lock manager at one site containing a replica of *Q*.
  - **Exclusive locks**. When a transaction needs to lock data item *Q*, it requests a lock on *Q* from the lock manager at all sites containing a replica of *Q*.
- Advantage - imposes less overhead on **read** operations.
- Disadvantage - additional overhead on writes

Quorum Consencus Protocol

- generalisation of both majority and biased protocols
- Each site is assigned a weight
  - Let S be the total of all site weights
- Choose two values read quorum Q_r and write quorum Q_w
  - Such that $Q_r + Q_w > S$ and $2 * Q_w > S$
  - Quorums can be chosen (and S computed) separately for each item
- Each read must lock enough replicas that the sum of the site weights is >= Q_r
- Each write must lock enough replicas that the sum of the site weights is >= Q_w
- Here we assume all replicas are written

Deadlock Handling

Consider the following two transactions and history, with item X and transaction T_1 at site 1, and item Y and transaction T_2 at site 2:

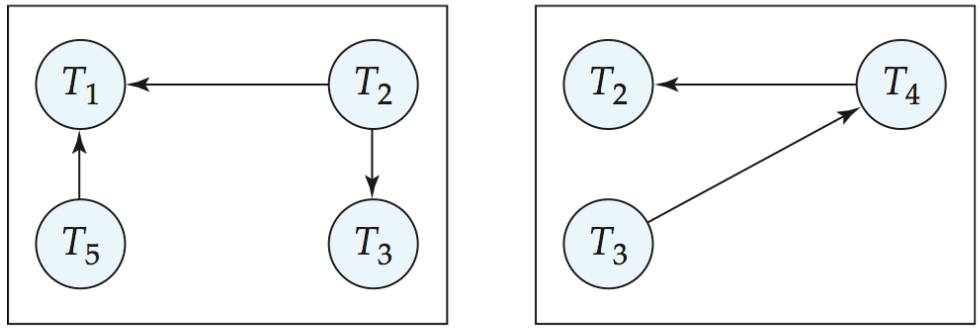
T_1: write (X) write (Y)	T_2: write (X) write (Y)
X-lock on X write (X)	
	X-lock on Y write (Y) wait for X-lock on X
Wait for X-lock on Y	

Result: deadlock which cannot be detected locally at either site

Deadlock Detection

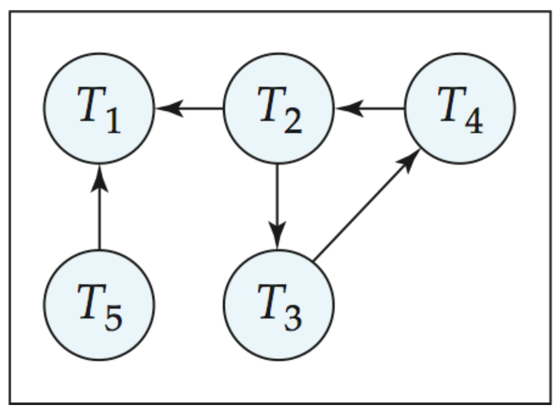
- In the **centralized deadlock-detection** approach, a global wait-for graph is constructed and maintained in a *single* site; the deadlock-detection coordinator
  - *Real graph*: Real, but unknown, state of the system.
  - *Constructed graph*: Approximation generated by the controller during the execution of its algorithm .
- the global wait-for graph can be constructed when:
  - a new edge is inserted in or removed from one of the local wait-for graphs.
  - a number of changes have occurred in a local wait-for graph.
  - the coordinator needs to invoke cycle-detection.
- If the coordinator finds a cycle, it selects a victim and notifies all sites. The sites roll back the victim transaction.

Local and Global Wait-For Graphs



site S_1 site S_2

Local



Global

Distributed Database in Oracle

- Some fragmentation can be achieved *expanding* a local database with another:
- **create database link *linkname***
  - The relations from *linkname* are known by relation@linkname
  - It is also possible to create aliases (cf. above in these slides) with
- **create synonym *alias* for *relation@linkname***
- This can be coupled with **materialized view**s which further enable vertical fragmentation
  - It is possible to establish how and when a materialised view is updated
    - **Fast refresh** uses materialised view logs to update only the rows that have changed since the last refresh.
    - **Complete refresh** always updates the entire materialised view.
    - **Force refresh** performs a fast refresh when possible. When a fast refresh is not possible, force refresh performs a complete refresh
- Queries can be distributed over the various sites

Replication in Oracle

- Oracle has support for homogeneous distributed replicated databases
  - It supports a multimaster replication with two-phase commit protocol
  - It supports master-slave replication by creating snapshots
- To create a replica
- **create snapshot *name* as *select query* with *type***
  - Replicas can be **read only** or **updatable** (*type* above)
- Groups of replicas, and their refreshing mechanisms can be define via special API procedures (Advanced Replication Management API)
  - In the labs you'll test DBMS_REFRESH
    - To create a refresh group and establish the refresh policy
- DBMS_REFRESH.MAKE(…)
    - To force a refresh
- DBMS_REFRESH. REFRESH(…)

Wrap-up

Sistemas de Bases de Dados 2019/20

Capítulo refere-se a: Database System Concepts, 7th Ed

Syllabus revisited

- Storing and file structure
  - Basis on how data is stored and (low-level) accessed in database systems
  - Understand how that is related to the OS storing, and how it affects performance of databases
- Indexing and hashing
  - Data structures for fast access, and how their performance depends on the specific data
- Query processing and optimisation
  - Get to know how a database system processes queries
  - Algorithms for query processing (including in parallel databases)
  - Understanding how the performance of algorithms depends on the specific data
  - Learn about optimisation methods for (automatically) tailoring the queries to the specific data

Syllabus revisited

- Transactions, concurrency and recovery
  - Understand the concept of ACID transaction
  - Protocols for isolation, and for recovery
  - Understanding the need for various (weaker) isolation levels in transactional database systems, and tailor their use
- Distributed Databases
  - Basis for distributed databases and their practical use
  - Adaptation of transaction protocols and of query processing

Goals revisited

- *Pretende-se dotar os alunos das bases necessárias à compreensão dos problemas envolvidos na construção e funcionamento de sistemas de gestão de bases de dados, dando ênfase à utilização eficiente de sistemas de bases de dados.*

- The most important components, and underlying concepts, of a database system have been exposed in the lectures
- They have been tested, and used in practice, in the labs
  - Not in big examples, but enough for understanding the differences between the various concepts/approaches
- They have been witnessed in Oracle, and in the database systems used for the project assignment
- This provides a systemic view of information systems, and a basis for more advanced courses in this area