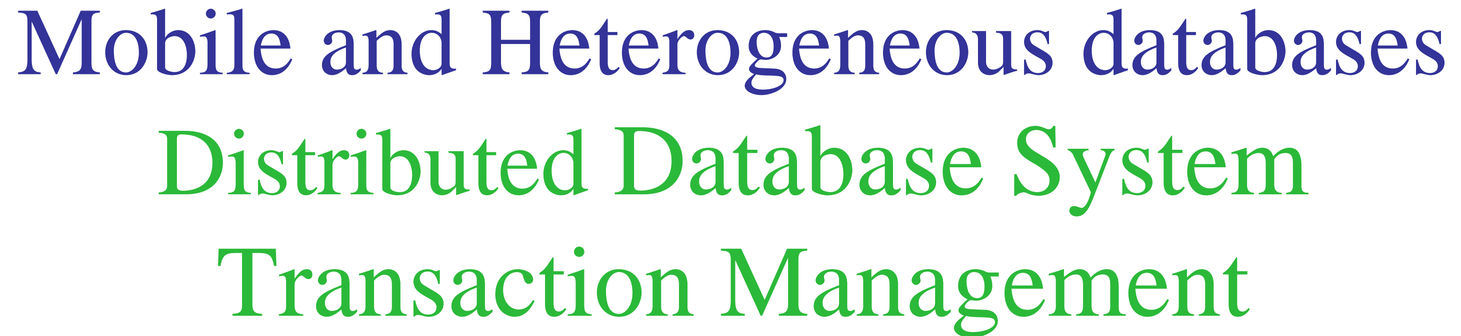


Mobile and Heterogeneous databases
Distributed Database System
Transaction Management

A.R. Hurson

Computer Science

Missouri Science & Technology

Distributed Database System

Note, this unit will be covered in four lectures. In case you finish it earlier, then you have the following options:

1) Take the early test and start CS6302.module4
2) Study the supplement module (supplement CS6302.module3)
3) Act as a helper to help other students in studying CS6302.module3

Note, options 2 and 3 have extra credits as noted in course outline.

Distributed Database System

Enforcement of background

Glossary of prerequisite topics

Familiar with the topics?

No

Review CS6302 module3background

Yes

Take Test

Pass?

No

Remedial action

Yes

Glossary of topics

At the end: take exam, record the score, impose remedial action if not successful

Current Module

Familiar with the topics?

No

Take the Module

Yes

Take Test

Pass?

No

Yes

Options

Study next module?

Lead a group of students in this module (extra credits)?

Study more advanced related topics (extra credits)?

Extra Curricular activities

Distributed Database System

- You are expected to be familiar with:
  - Transaction processing in centralized database configuration
- If not, you need to study CS6302.module3.background

Distributed Databases

- Module 2 concentrated on query processing, query optimization, and more specifically, query processing in distributed databases. This module will concentrate on transaction processing in general and transaction processing in distributed system. Note that we will distinguish transaction from query. A query does not change the data in base sources (e.g., relations), however, a transaction my do so. As a result, queries, initiated by several users, do not have conflicts with each other and can be executed in any order (including simultaneously). However, this is not true for transactions, since they may be in conflict with each other and hence needs to be executed in a proper sequence.

Distributed Databases

- In this module, we will talk about:
  - Transaction processing and management
  - Formal definition of transactions
  - ACID property
  - Serializability
  - Concurrency control
  - Concurrency control protocols
  - Transaction processing
  - Transaction processing in distributed system

Distributed Databases

- Distributed transaction management
  - Several issues hinder transaction consistency:
    - Concurrent execution of transactions,
    - Replicated data, and
    - Failure.
  - A replicated database is in a mutually consistent state if copies of every data item in it have identical values — one copy equivalence.

Distributed Databases

- Distributed transaction management
  - In general, in a database system, one needs to ensure **A**tomicity, **C**onsistency, **I**solation, and **D**urability properties of transactions:

Distributed Databases

- Distributed transaction management
  - Atomicity (all or nothing): either all operations of the transaction are reflected in database, or none are.
  - Consistency (no violation of integrity rules): Execution of transaction in isolation preserves the consistency of the database.
  - Isolation (Concurrent changes invisible and serializable): Even though multiple transactions may execute concurrently, each transaction assumes it is executed in isolation (it is unaware of other transactions executing concurrently in the system).
  - Durability (Committed updates persist): After a transaction completes successfully, its results are becoming persistence.

Distributed Databases

- Distributed transaction management
  - It is much easier if internally consistent transactions are run serially — each transaction is executed alone, one after the another. However, there are two good motivations to allow concurrent execution of transactions:
    - Improved throughput and resource utilization
    - Improved average response time.
  - Concurrent execution of transactions means that they should be scheduled in order to ensure consistency.

Distributed Databases

- Distributed transaction management
  - The concurrency control mechanism attempts to find a suitable trade-off between maintaining the consistency of the database and maintaining a high level of concurrency.
  - Note concurrency control deals with the isolation and consistency properties of transactions.

Distributed Databases

- Distributed transaction management
  - A schedule (a history) over a set of transactions $T = \{T_1, T_2, \ldots, T_n\}$ is an interleaved order of execution of these transactions.
  - A schedule is a complete schedule, if it defines the execution order of all operations in its domain.

Distributed Databases

- ## Distributed transaction management

Formally a complete schedule S_T^C over a set of transactions $T = \{T_1, T_2, \ldots, T_n\}$ is a partial order $S_T^C = \{\Sigma_T, \prec_T\}$ where:

$$\Sigma_T = \bigcup_{i=1}^{n} \Sigma_i$$

$$\prec_T \supseteq \bigcup_{i=1}^{n} \prec_i$$

For any two conflicting operations $O_{ij}, O_{kl} \in \Sigma_T$ either $O_{ij} \prec_T O_{kl}$ or $O_{kl} \prec_T O_{ij}$.

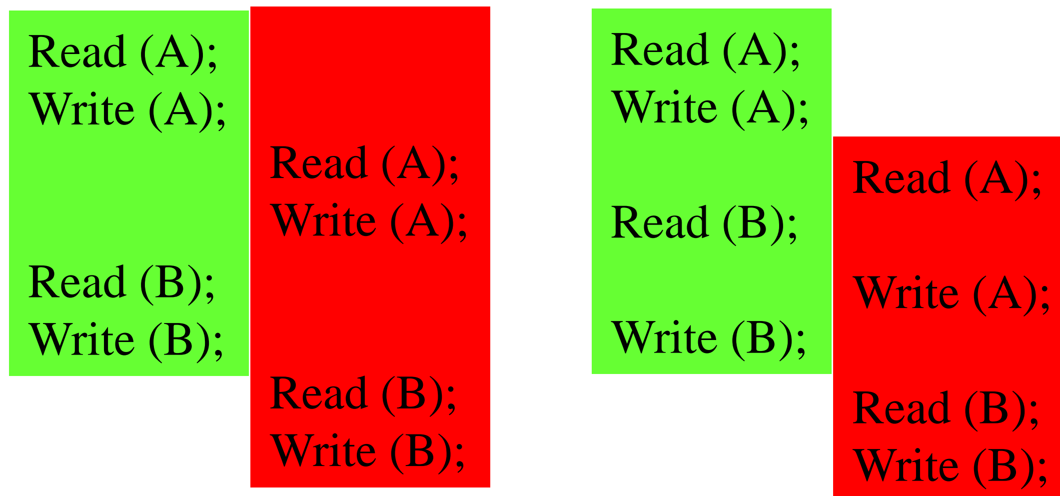
1st rule shows that the schedule must contain all operations in participating transactions.
2nd rule shows that the ordering relation on *T* is a superset of ordering relations of individual transactions.
3rd rule shows the execution order among conflicting operations.

Distributed Databases

- Distributed transaction management
  - When interleaving instructions from different transactions, one can come up with a number of execution sequence (schedule).
  - In this case, we can ensure consistency if the concurrent schedule has the same effect as a serial schedule of transactions – Concurrent schedule is equivalent to a serial schedule.

Distributed Databases

- Distributed transaction management – Conflict Serializability
  - Two schedules *S* and *S'* are conflict equivalent if *S'* is generated by a series of swaps of non conflicting instructions in *S*.



- Distributed transaction management – Conflict Serializability
  - Formally, two schedules *S* and *S'* over a set of transactions are conflict equivalent if for each pair of conflicting operations O_{ij}, O_{kl} ($i \neq k$), whenever $O_{ij} \prec_S O_{kl}$, then $O_{ij} \prec_{S'} O_{kl}$.

Distributed Databases

- Distributed transaction management – Conflict Serializability
  - Concept of conflict equivalent leads to the concept of conflict serailizability.
  - A schedule *S* is conflict serializable if it is conflict equivalent to a serial schedule.

Distributed Databases

- Distributed transaction management
  - The primary function of concurrency controller is to generate a serializable schedule for execution of a sequence of transactions – to devise algorithms that guarantee the generation of serializable schedules.

Distributed Databases

- Distributed transaction management
  - In a distributed databases, with no data replication if local schedules are serializable then their union (global schedule) is also serializable as long as local serialization order is identical.

Distributed Databases

- Distributed transaction management
  - In distributed databases, in case of replication, local schedules could be serializable, but the global schedule might not — mutual consistency of database is compromised.
  - Consider the following transactions and the two schedules:

Distributed Databases

- Distributed transaction management

T_1:
Read (x)
$x \leftarrow x + 5$
Write (x)
Commit

T_2:
Read (x)
$x \leftarrow x * 10$
Write (x)
Commit

$S_1 = \{R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2\}$

$S_2 = \{R_2(x), W_2(x), C_2, R_1(x), W_1(x), C_1\}$

Distributed Databases

- Distributed transaction management
  - In case of replicated databases, it is the task of replica control protocol to make sure that the operations (read and write) on logical data are mapping correctly onto physical data.
  - Assume we have a data item (x) with copies x_1, x_2, …, x_n.

Distributed Databases

- Distributed transaction management
  - read(x) needs to be mapped onto one of the replica, so say it will be modified as read(X_5).
  - write(x) will be extended to write(x_1), write(x_2), write(x_3), …, write(x_n).

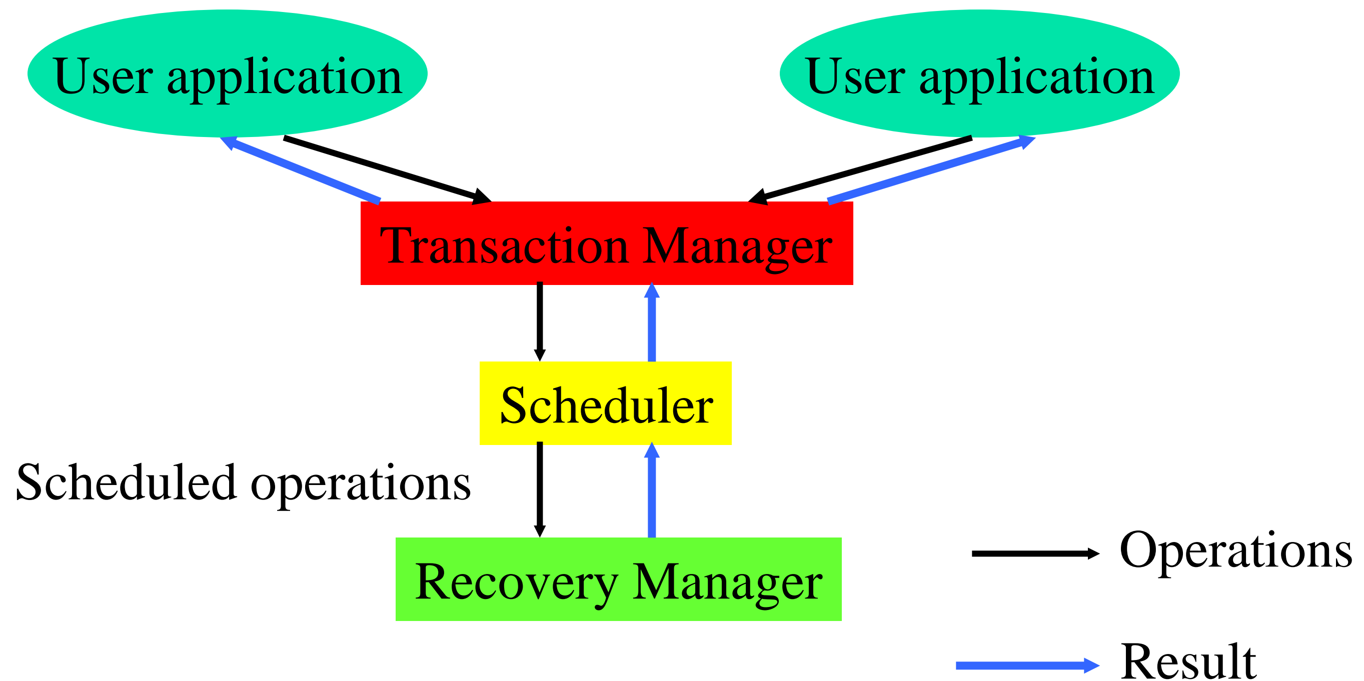
- Distributed transaction management
  - Within the scope of the distributed databases, we distinguish two entities:
    - Local entities, and
    - Global entity.
  - As a result, we can talk about local transactions *vs*. global transactions.

Distributed Databases

- Distributed transaction management
  - In this environment, each site has its own local transaction manager, whose function is to ensure ACID properties for those local transactions executed at that site.
  - Different transaction managers cooperate with each other to execute global transactions.

Distributed Databases

- Centralized Transaction Execution

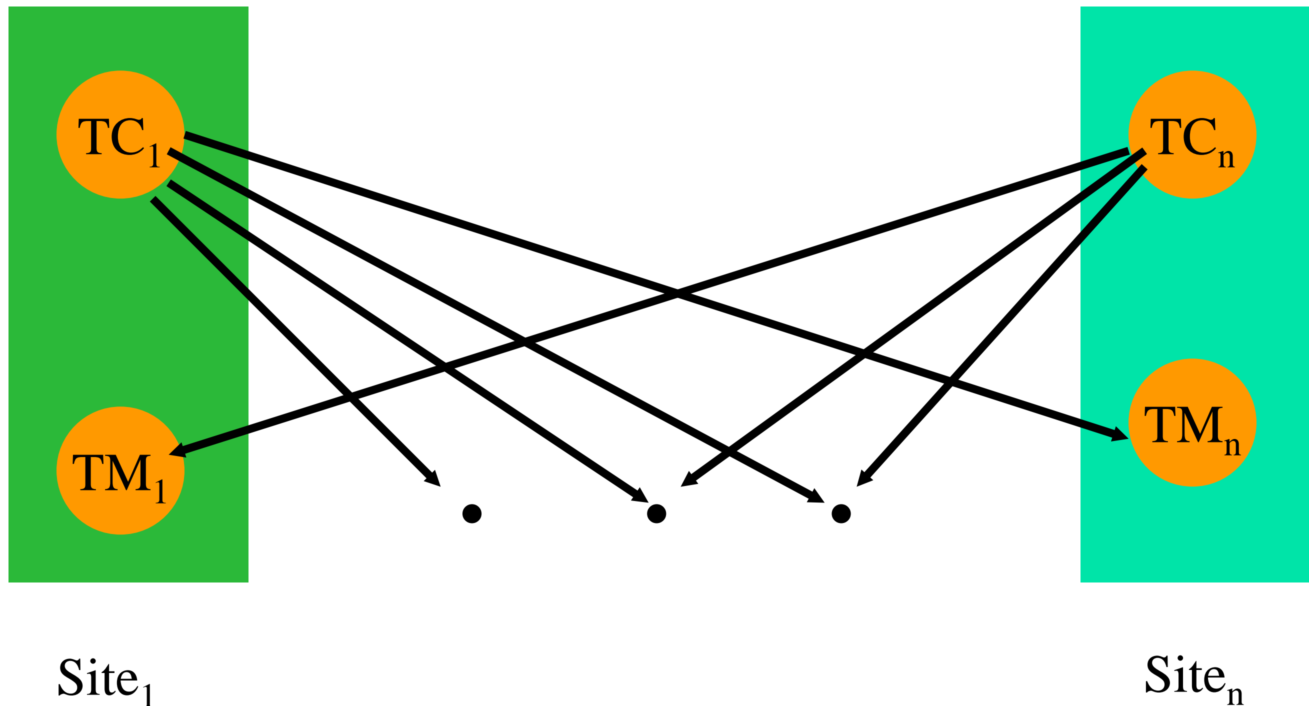


Distributed Databases

- Centralized Transaction Execution
  - Transaction Manager is responsible for coordinating the execution of the database operations on behalf of an application.
  - Scheduler is responsible for the implementation of a specific concurrency control algorithm.
  - Recovery manager is responsible to implement procedures that transform database into a consistent state after a failure.

Distributed Databases

- Distributed transaction management – General configuration



Distributed Databases

- Distributed transaction management – General configuration
  - Transaction manager manages the execution of transactions (local transaction or global sub-transaction) that access local data.
  - Transaction coordinator coordinates the execution of transactions (local or global) that are initiated at that site.

Distributed Databases

- Distributed transaction management – General configuration
  - The structure of transaction manager is similar to the one in a centralized database. It is responsible to:
    - Maintain a log for recovery,
    - Participate in a concurrency control protocol to coordinate concurrent execution of transaction at that site.

Distributed Databases

- Distributed transaction management – General configuration
  - The transaction coordinator is a new entity and responsible for:
    - Starting the execution of the transaction,
    - Converting a global transaction into sub-transactions and distribute sub-transactions to the designated sites, and
    - Coordinate the termination of the transactions – note a global transaction must be committed or aborted at all sites.

Distributed Databases

- Distributed transaction management – General configuration
  - Besides the type of failures common in a centralized database (software/hardware errors, disk crashes), a distributed system suffers from:
    - Site failure,
    - Loss/corruption of messages,
    - Link failure,
    - Network partitioning

Distributed Databases

- Distributed transaction management – General configuration
  - Within a distributed system, sites are communicating with each other via messages. If two sites are not physically linked together, then messages must be routed through a sequence of communication links.
  - In case of link failure, messages might be able to find different route through the network.

Distributed Databases

- Distributed transaction management – General configuration
  - Network partitioning is the result of link failure, where a group of sites cannot communicate with each other – system is partitioned into subsystems.

Distributed Databases

- Distributed transaction management – General Comments
  - Lock Based approach: Transactions are synchronized by physical or logical locks on some portion (granule) of the database – Locking granularity.
  - In a distributed environment, we can distinguish three classes of lock based protocols:
    - Centralized Locking
    - Primary Copy Locking
    - Decentralized Locking

- Distributed transaction management – Lock Based approach
  - Centralized Locking: One of the sites in the network is designated as the primary site, where the lock tables for the entire database are stored, in charge with the task of granting locks to transactions.

Distributed Databases

- Distributed transaction management – Lock Based approach
  - Primary Copy Locking: One of the copies of each data unit is designated as the primary copy and it is the primary copy that has to be locked for the purpose of accessing that data unit – all transactions desiring to access data item obtain their lock at the site where the primary copy resides at.
  - If data item is not replicated, the primary copy protocol distributes the lock management task among a number of sites.

Distributed Databases

- Distributed transaction management – Lock Based approach
  - Decentralized Locking: The lock manager duty is shared by all the sites in the network – Execution of a transaction involves the participation and coordination of schedulers at several sites.
  - In case of replication, transaction accessing a data item must obtain locks at all sites.

Distributed Databases

- Distributed transaction management – General Comments
  - Timestamp Ordering: Assigns a unique identifier to each transaction and data items in order to organize their execution sequence.
  - In this group we can talk about:
    - Basic Timestamp Ordering
    - Multi-version Timestamp Ordering
    - Conservative Timestamp Ordering

Distributed Databases

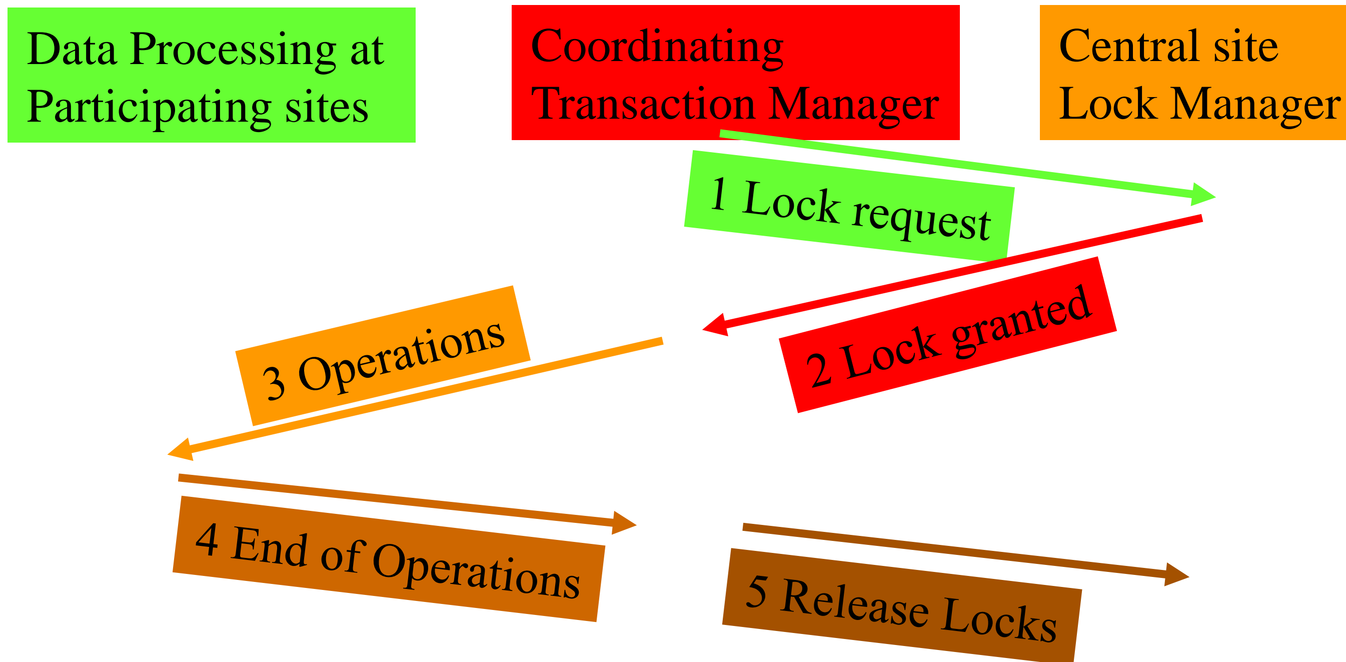
- Distributed transaction management – General Comments
  - Note there is a class of Hybrid Concurrency control algorithms which are mixed of locking-based and timestamp-based schemes. This class is intended to improve efficiency and the level of concurrency.

Distributed Databases

- Distributed transaction management – Centralized Two-phase Locking protocol
  - Lock manager responsibility is delegated to a single site.
  - Transaction coordinators at the other sites communicate with the centralized lock manager rather than with their own lock managers.
  - This scheme also is referred to as: primary site two-phase locking.

Distributed Databases

- Distributed transaction management – Centralized Two-phase Locking protocol



- Distributed transaction management – Centralized Two-phase Locking protocol
  - Centralized lock manager does not communicate with the data processing sites directly.
  - The distributed transaction manager must implement replica control protocol, if database is replicated.
  - Bottleneck at central lock manager and reliability are the major drawbacks of this approach.

Distributed Databases

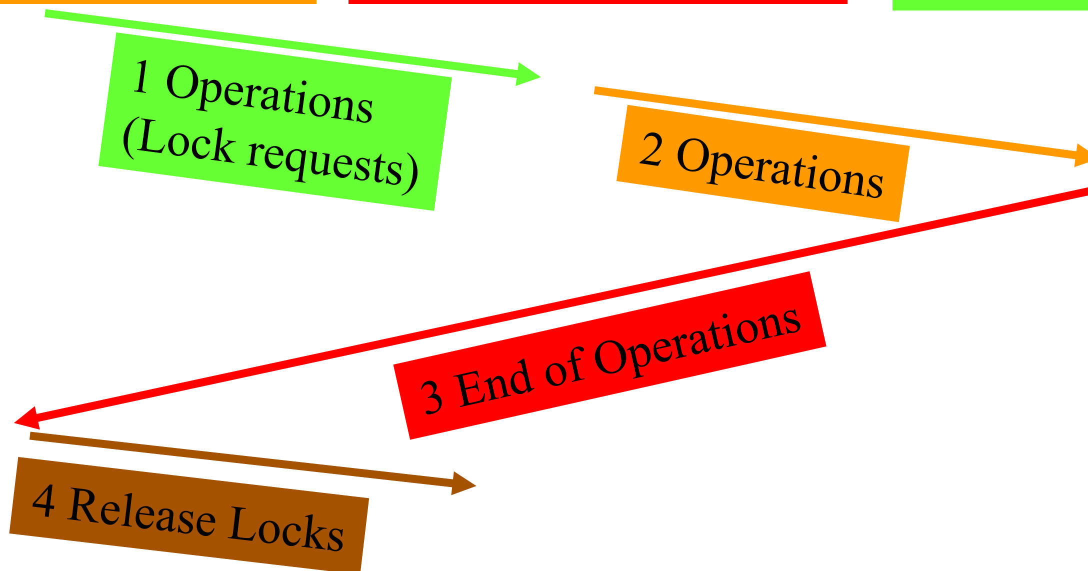
- Distributed transaction management — Primary copy Two-phase Locking protocol
  - This scheme is the direct extension of centralized two-phase locking protocol in an attempt to remove its performance bottleneck.
  - It simply distributes the task of lock manager among several lock managers for a given set of lock units.
  - The transaction coordinators send their lock and unlock requests to the lock managers that are responsible for specific lock unit — The location of the primary copy of each data item needs to be determined before sending a lock or unlock request.

Distributed Databases

- Distributed transaction management – Distributed Two-phase Locking protocol
  - Here a lock manager exists at each site.
    - In case of no replication, this scheme degenerates into the Primary copy Two-phase Locking protocol.
    - In case of replica, the algorithm implements the read once/write all (ROWA) replica control protocol.

Distributed Databases

- Distributed transaction management – Distributed Two-phase Locking protocol



Distributed Databases

- Distributed transaction management – Distributed Two-phase Locking protocol
  - In comparison to centralized approach, lock and unlock messages are sent to the lock managers at all participating sites. In addition, operations are sent, by participating lock managers, to the data processors instead of coordinating transaction manager.

Distributed Databases

- Distributed transaction management – Two-phase commit protocol
  - Transaction T is initiated at site S_i with the transaction coordinator TC_i.
  - T is decomposed by TC_i and transmitted to different sites for execution.
  - When all sites at which T is executed inform TC_i that T is completed, TC_i starts the 2-phase commit protocol.

Distributed Databases

- Distributed transaction management – Two-phase commit protocol (Phase1)
  - TC_i adds the record "prepare T" to the log and forces the log to permanent storage.
  - "Prepare T" message is communicated with all the sites involved.
    - At each designated site, the transaction manager follows the following actions:
      - If not willing to "commit", then adds "no T" to its log and sends an "abort T" message to TC_i.
      - If willing to commit, then adds "ready T" to its log and sends a "ready T" message to TC_i.

Distributed Databases

- Distributed transaction management – Two-phase commit protocol (Phase2)
  - On receiving replies from the sites, TC_i determines whether or not the transaction must be committed or aborted.
  - Based on the local decisions, either a "commit T" or an "abort T" is logged on permanent storage.
  - TC_i sends either an "abort T" or "commit T" to the participating sites.
  - The message from the TC_i is logged at the participating sites.

Distributed Databases

- Distributed transaction management – Two-phase commit protocol
  - A "ready T" message from a participating site to TC_i is a promise that the site will follow the coordinator commit or abort command.
  - Unanimity is required by the coordinator to commit a transaction.
  - In some implementation, at the end of the 2nd phase, participating sites send "acknowledge T" message to the TC_i. TC_i upon receiving all "acknowledge T" messages, adds the "complete T" record into its log.

Distributed Databases

- Distributed transaction management – Two-phase commit protocol
  - Failure of a participating site: Transaction is aborted if TC_i detects a failure before receiving "ready T" message, otherwise the coordinator continue normal sequence of operations.

Distributed Databases

- Distributed transaction management – Two-phase commit protocol
  - After recovery from failure, the site must examine its log to determine the fate of those transactions that were in the midst of execution when the failure occurred.
    - If log contain "commit T", then the site executes "redo (T)",
    - If log contains "abort T", then the site performs "undo (T)",
    - If log contains "ready T", then it should consult with the coordinator,
    - If log contains no "commit, abort, ready" messages about *T*, then it performs "undo (T)".

Distributed Databases

- Distributed transaction management — Two-phase commit protocol
  - Failure of the coordinator: If the coordinator fails in the midst of operation of a transaction *T*:
    - If an active site contains "commit T" in its log, then *T* must be committed,
    - If an active site contains "abort T" in its log, then *T* must be aborted,
    - If some active site do not contain "ready T" in their logs, then *T* must be aborted,
    - If none of these cases holds, then all active sites must have "ready T" record in their logs. In this case they must wait for the coordinator site to recover — this will cause blocking problem since data items involved in *T* might be unavailable to other transactions.

Distributed Databases

- Distributed transaction management – Two-phase commit protocol
  - Network Partition: In this case we have two possibilities:
    - Coordinator and all sites are in the same partition. In this case the failure has no effect on the fate of transaction.
    - Coordinator and all sites involved belong to several partitions. In this case, coordinator and sites in the same partition, follow the protocol that the other sites in other partitions have failed. Sites in the partitions that does not contain the coordinator follow the protocol that the coordinator has failed.

Distributed Databases

- Distributed transaction management – Three-phase commit protocol
  - This is an extension to two-phase commit that avoids blocking under certain assumptions. It assumes no network partitioning and can tolerate up to *K* sites failure.
  - Under aforementioned assumptions, it introduces a third phase where multiple sites are involved in the decision to commit.

Distributed Databases

- Distributed transaction management – Three-phase commit protocol
  - Instead of noting the commit decision in its log (permanent storage) and then informing other sites involved, the coordinator make sure that at least *k* sites are aware of its intension to commit.
  - If the coordinator fails, the remaining sites first select a coordinator, the new coordinator, checks the status of the transaction from other sites. If there was a decision to commit the transaction, the new coordinator respects that and initiate the third phase, otherwise, it will abort the transaction.

Distributed Databases

- Distributed transaction management – Locking protocols
  - First we will look at the schemes that require update to be done on all replicated data. Then, we will look at schemes that allow us to relax this restriction.

Distributed Databases

- Distributed transaction management – Locking protocol
  - Locking protocol as we studied before can be used for distributed environment by extending the scope of lock-manager in order to handle replicated data.
  - Two cases will be considered:
    - Single lock-manager approach
    - Distributed lock-manager approach

Distributed Databases

- Distributed transaction management – Single lock-manager approach
  - This is the same as locking scheme in centralized database environment.
  - A single lock-manager is maintained in one site – say S_i. As a result, every lock and unlock requests are made at site S_i.

Distributed Databases

- Distributed transaction management – Single lock-manager approach
  - Upon a request, lock-manager determines whether or not the lock can be granted immediately:
    - If so, a message to that effect is sent to the site requesting the lock.
    - If no, request is delayed until it can be granted and a message to this fact is sent to the site requesting the lock.
  - A transaction can read data from any replica, but all replicas participate in a write operation.

Distributed Databases

- Distributed transaction management – Single lock-manager approach
  - Simple implementation
  - Simple deadlock detection
    - Are major advantages of this approach.
  - Bottleneck at the lock-manager
  - Vulnerability and lack of fault tolerance
    - Are the disadvantages of this approach.

Distributed Databases

- Distributed transaction management – Distributed lock-manager approach
  - Each site maintains a local lock manager whose function is to administer the lock and unlock requests for data items stored at that site.

Distributed Databases

- Distributed transaction management – Distributed lock-manager approach (request for unreplicated data)
  - When a transaction wishes to lock data item from a site (S_i) and if the data item is not replicated, a message is sent to the lock manager at site S_i.
    - If the data item is locked in an incompatible mode, then the request is delayed until it can be granted.
    - Once it is determined that the lock can be granted, the lock manager sends a message back to the initiator that the lock request is granted.

Distributed Databases

- Distributed transaction management — Distributed lock-manager approach (request for unreplicated data)
  - This approach has low implementation overhead, it is easy to implement, and without bottleneck at a local site.
  - However, it is harder to implement deadlock detection — there is a potential for inter-site deadlock even when there is no deadlock within a single site.

Distributed Databases

- Distributed transaction management – Distributed lock-manager approach (request for replicated data)
  - Primary copy protocol
    - In this case, one replica is chosen as the primary copy and hence, its corresponding site is called primary site.
    - Any lock request for a data item must be sent to the primary site. As a result, this approach allows implementation of concurrency control for replicated data as the replica does not exist.

Distributed Databases

- Distributed transaction management — Distributed lock-manager approach (request for replicated data)
  - Majority protocol
    - In case data item is replicated in n sites, a lock request must be sent to more than one-half of the sites.
    - A transaction cannot operate on the requested data item until it has obtained a lock on the majority of the replicas.

Distributed Databases

- Distributed transaction management – Distributed lock-manager approach (request for replicated data)
  - Majority protocol
    - This approach is more complicated to implement and requires more messages to lock and unlock a data item.
    - There is also a potential for global deadlocks.

Distributed Databases

- Distributed transaction management — Distributed lock-manager approach (request for replicated data)
  - Biased protocol
    - This is a version of the majority protocol where the request for shared locks are given more favorable treatment than the request for exclusive locks.
    - Consequently, it imposes less overhead on read operations than does the majority protocol — it is more appropriate for application domains which require much more read operations than write operations.
    - As before, the transaction does not operate on a data item until it has successfully obtained a lock on a majority of the replica.

Distributed Databases

- Distributed transaction management – Distributed lock-manager approach (request for replicated data)
  - Biased protocol
    - Shared lock request on a data item is sent to just one site holding a replica.
    - Exclusive lock request on a data item is sent to all sites holding the replicas.

Distributed Databases

- Distributed transaction management – Distributed lock-manager approach (request for replicated data)
  - Quorum Consensus protocol
    - It is a generalization of both majority and biased protocols.
    - Each site is an assigned weight, w_i.
    - Read and write operations on an item *x* is enhanced by two integers, read Quorum Q_r and write Quorum Q_w, that must satisfy the following relations: $Q_r^x + Q_w^x > \Sigma w_i^x$ $2 * Q_w^x > \Sigma w_i^x$

Distributed Databases

- Distributed transaction management – Distributed lock-manager approach (request for replicated data)
  - Quorum Consensus protocol
    - To execute a read/write on *x*, enough replicas must be read/written that their total weight satisfy the following: $\sum w_i^x \geq Q_r^x / Q_w^x$

Distributed Databases

- Distributed transaction management — Distributed lock-manager approach (request for replicated data)
  - Quorum Consensus protocol
    - This approach is more dynamic which allows one, based on the application domain, to favor read or write operations.
    - In addition, site weight can be assigned such that the sites that are more reliable weighted higher.

Distributed Databases

- Distributed transaction management – Conservative timestamp-ordering
  - What is the major problem with the timestamp-ordering in a distributed system?
  - Timestamp-ordering is a deadlock free protocol, since operations never wait, but it forces transaction restart.
  - This is a major problem in distributed systems since transactions generated by inactive sites executed at active sites will keep being rejected continuously.

- Distributed transaction management – timestamp-ordering
  - To avoid continuous restart of rejected transactions, counters at different sites must be synchronized.
  - Synchronization cost of counters is expensive – large number of required messages.
  - Simple solution can be adapted to avoid high cost of synchronization (if we use system clock and if clocks are of comparable speed).

Distributed Databases

- Distributed transaction management — timestamp-ordering
  - As another solution, we let the transaction coordinators to communicate with each other.
  - A transaction coordinator sends its remote operations to other transaction coordinators at other sites — instead of transaction managers.
  - At receiving sites, each transaction coordinator whose counter is less than the incoming timestamp, adjust its counter to one more than the incoming one — this policy ensures that none of the counters gets away or lags behind the others significantly.

Distributed Databases

- Distributed transaction management – Timestamp ordering
  - This approach assigns a unique identifier to each transaction in order to decide the serialization order. So in a distributed environment the challenge lies in the generation of the unique identifier.
  - Two cases can be recognized:
    - Centralized timestamping,
    - Distributed timestamping.

Distributed Databases

- Distributed transaction management – Timestamp ordering
  - Centralized timestamping: In this case, a single site distributes the timestamping.
  - Distributed timestamping: In this case, a global timestamp is composed of two entities:
    - A unique local timestamp, as in centralized environment,
    - A unique site identifier.

Local site timestamp	Site identifier