Distributed Databases
Distributed Databases

- Heterogeneous and Homogeneous Databases
- Distributed Data Storage
- Distributed Transactions
- Commit Protocols
- Concurrency Control in Distributed Databases
- Availability
- Distributed Query Processing
- Heterogeneous Distributed Databases
- Directory Systems
**Concepts**

**Distributed Database:** logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network.

**Distributed DBMS (DDBMS):** Software system, permits the management of the distributed database and makes the distribution transparent to users.

**Fundamental Principle:** make distribution transparent to user.

*The fact that fragments stored on different computers hidden to user (Abstraction)*

**DDBMS has following characteristics:**
- Collection of logically-related shared data.
- Data split into fragments.
- Fragments may be replicated.
- Fragments/replicas allocated to sites.
- Sites linked by a communication network.
- Data at each site is under control of a DBMS.
- DBMSs handle local applications autonomously.
- Each DBMS participates in at least one global application.
Important difference between DDBMS and distributed processing

DDBMS

Distributed processing of centralised DBMS

Site 1  Site 2  Site 3  Site 4

Computer network

DB

Site 1  Site 4

Computer network

DB

Site 3
Difference between DDBMS and Parallel database

**DDBMS**

- Site 1
- Site 2
- Site 3
- Site 4

**Parallel Database Architectures:**
- Shared: a) memory b) disk c) nothing
Advantages of DDBMSs

- Reflects organizational structure
- Improved shareability and local autonomy
- Improved availability
- Improved reliability
- Improved performance
- Cost
- Modular growth
Disadvantages of DDBMSs

- Complexity
- Cost
- Security
- Integrity control more difficult
- Lack of standards
- Lack of experience
- Database design more complex
Homogeneous Distributed Databases

- In a homogeneous distributed database
  - All sites have identical software
  - Are aware of each other and agree to cooperate in processing user requests.
  - Each site surrenders part of its autonomy in terms of right to change schemas or software
  - Appears to user as a single system

- In a heterogeneous distributed database
  - Different sites may use different schemas and software
    - Difference in schema is a major problem for query processing
    - Difference in software is a major problem for transaction processing
  - Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing
Distributed Data Storage

- Assume relational data model
- Replication
  - System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.
- Fragmentation
  - Relation is partitioned into several fragments stored in distinct sites
- Replication and fragmentation can be combined
  - Relation is partitioned into several fragments: system maintains several identical replicas of each such fragment.
Data Replication

- A relation or fragment of a relation is **replicated** if it is stored redundantly in two or more sites.

- **Full replication** of a relation is the case where the relation is stored at all sites.

- Fully redundant databases are those in which every site contains a copy of the entire database.
Data Replication (Cont.)

- Advantages of Replication
  - **Availability**: failure of site containing relation $r$ does not result in unavailability of $r$ as replicas exist.
  - **Parallelism**: queries on $r$ may be processed by several nodes in parallel.
  - **Reduced data transfer**: relation $r$ is available locally at each site containing a replica of $r$.

- Disadvantages of Replication
  - Increased cost of updates: each replica of relation $r$ must be updated.
  - Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.
    - One solution: choose one copy as **primary copy** and apply concurrency control operations on primary copy.
## Data Fragmentation

- Division of relation $r$ into fragments $r_1, r_2, \ldots, r_n$ which contain sufficient information to reconstruct relation $r$.

- **Horizontal fragmentation**: each tuple of $r$ is assigned to one or more fragments

- **Vertical fragmentation**: the schema for relation $r$ is split into several smaller schemas
  - All schemas must contain a common candidate key (or superkey) to ensure lossless join property.
  - A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

- Example: relation account with following schema

  $Account = (branch\_name, account\_number, balance)$
Horizontal Fragmentation of `account` Relation

<table>
<thead>
<tr>
<th>branch_name</th>
<th>account_number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hillside</td>
<td>A-305</td>
<td>500</td>
</tr>
<tr>
<td>Hillside</td>
<td>A-226</td>
<td>336</td>
</tr>
<tr>
<td>Hillside</td>
<td>A-155</td>
<td>62</td>
</tr>
</tbody>
</table>

account\(_1\) = \(\sigma_{\text{branch\_name}=\text{"Hillside"}}(\text{account})\)

<table>
<thead>
<tr>
<th>branch_name</th>
<th>account_number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valleyview</td>
<td>A-177</td>
<td>205</td>
</tr>
<tr>
<td>Valleyview</td>
<td>A-402</td>
<td>10000</td>
</tr>
<tr>
<td>Valleyview</td>
<td>A-408</td>
<td>1123</td>
</tr>
<tr>
<td>Valleyview</td>
<td>A-639</td>
<td>750</td>
</tr>
</tbody>
</table>

account\(_2\) = \(\sigma_{\text{branch\_name}=\text{"Valleyview"}}(\text{account})\)
### Vertical Fragmentation of `employee_info` Relation

<table>
<thead>
<tr>
<th>branch_name</th>
<th>customer_name</th>
<th>tuple_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hillside</td>
<td>Lowman</td>
<td>1</td>
</tr>
<tr>
<td>Hillside</td>
<td>Camp</td>
<td>2</td>
</tr>
<tr>
<td>Valleyview</td>
<td>Camp</td>
<td>3</td>
</tr>
<tr>
<td>Valleyview</td>
<td>Kahn</td>
<td>4</td>
</tr>
<tr>
<td>Hillside</td>
<td>Kahn</td>
<td>5</td>
</tr>
<tr>
<td>Valleyview</td>
<td>Kahn</td>
<td>6</td>
</tr>
<tr>
<td>Valleyview</td>
<td>Green</td>
<td>7</td>
</tr>
</tbody>
</table>

\[
deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id} (employee\_info)
\]

<table>
<thead>
<tr>
<th>account_number</th>
<th>balance</th>
<th>tuple_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-305</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>A-226</td>
<td>336</td>
<td>2</td>
</tr>
<tr>
<td>A-177</td>
<td>205</td>
<td>3</td>
</tr>
<tr>
<td>A-402</td>
<td>10000</td>
<td>4</td>
</tr>
<tr>
<td>A-155</td>
<td>62</td>
<td>5</td>
</tr>
<tr>
<td>A-408</td>
<td>1123</td>
<td>6</td>
</tr>
<tr>
<td>A-639</td>
<td>750</td>
<td>7</td>
</tr>
</tbody>
</table>

\[
deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id} (employee\_info)
\]
Advantages of Fragmentation

- **Horizontal:**
  - allows parallel processing on fragments of a relation
  - allows a relation to be split so that tuples are located where they are most frequently accessed

- **Vertical:**
  - allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id attribute allows efficient joining of vertical fragments
  - allows parallel processing on a relation

- **Vertical and horizontal fragmentation can be mixed.**
  - Fragments may be successively fragmented to an arbitrary depth.
Open Database access and interoperability

Open Group formed Specification Working Group (SWG)
to provide specifications that create database infrastructure environment where there is:

- **Common SQL API**: allows client applications to be written that do not need to know vendor of DBMS they are accessing.

- **Common database protocol**: enables DBMS from one vendor to communicate directly with DBMS from another vendor without need for a gateway.

- **Common network protocol**: allows communications between different DBMSs.
Multidatabase system (MDBS)

**MDBS:** distributed DBMS where each site maintains complete autonomy

Resides transparently on top of existing database and file systems - presents a single database to its users.

Allows users to *access* and *share* data without requiring physical database integration.

2 types:
- *Unfederated MDBS* (no local users)
- *Federated MDBS*. 
### Overview of Networking

**Network:** interconnected collection of autonomous computers, capable of exchanging information.

- **Local Area Network (LAN)** intended for connecting computers at the same site.
- **Wide Area Network (WAN)** used when computers or LANs need to be connected over long distances.

<table>
<thead>
<tr>
<th>WAN</th>
<th>LAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distances up to thousands of kilometers</td>
<td>Distances up to a few kilometers</td>
</tr>
<tr>
<td>Link autonomous computers</td>
<td>Link computers that cooperate in distributed applications</td>
</tr>
<tr>
<td>Network managed by independent organization (using telephone or satellite links)</td>
<td>Network managed by users (using privately owned cables)</td>
</tr>
<tr>
<td>Data rate up to 33.6 kbit/s (dial-up via modem), 45 Mbit/s (T3 circuit)</td>
<td>Data rate up to 2500 Mbit/s (ATM)</td>
</tr>
<tr>
<td>Complex protocol</td>
<td>Simpler protocol</td>
</tr>
<tr>
<td>Use point-to-point routing</td>
<td>Use broadcast routing</td>
</tr>
<tr>
<td>Use irregular topology</td>
<td>Use bus or ring topology</td>
</tr>
<tr>
<td>Error rate about $1:10^5$</td>
<td>Error rate about $1:10^9$</td>
</tr>
</tbody>
</table>
A reference architecture consists of:
- Set of global external schemas.
- Global conceptual schema (GCS).
- Fragmentation schema and allocation schema.
- Set of schemas for each local DBMS conforming to 3-level ANSI/SPARC.

For federated MDBS:
In DDBMS: GCS is union of all local conceptual schemas.
In FMDBS: GCS is subset of local conceptual schemas (LCS), consisting of data that each local system agrees to share.

GCS of *tightly coupled system* involves integration of either parts of LCSs or local external schemas.
FMDBS with no GCS is called *loosely coupled.*
Reference Architecture for DDBMS
Reference Architecture for Tightly-Coupled FMDBS
### Data Allocation

#### Four alternative strategies regarding placement of data:

- **Centralized**: single database and DBMS stored at one site with users distributed across the network.
- **Partitioned**: Database partitioned into disjoint fragments, each fragment assigned to one site.
- **Complete Replication**: Consists of maintaining complete copy of database at each site.
- **Selective Replication**: Combination of partitioning, replication, and centralization.

#### Table 22.3 Comparison of strategies for data allocation.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Locality of reference</th>
<th>Reliability and availability</th>
<th>Performance</th>
<th>Storage costs</th>
<th>Communication costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized Fragmented</td>
<td>Lowest</td>
<td>Lowest</td>
<td>Unsatisfactory</td>
<td>Lowest</td>
<td>Highest</td>
</tr>
<tr>
<td>Complete replication</td>
<td>High&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Low for item; high for system</td>
<td>Satisfactory&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Lowest</td>
<td>Low&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>Selective replication</td>
<td>High&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Low for item; high for system</td>
<td>Satisfactory&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Average</td>
<td>Low&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

<sup>a</sup> Indicates subject to good design.
Transparencies in a DDBMS

Transparency hides implementation details from users.

Overall objective: equivalence to user of DDBMs to centralised DBMS

- FULL transparency not universally accepted objective

Four main types:

1. Distribution transparency
2. Transaction transparency
3. Performance transparency
4. **DBMS transparency** (only applicable to heterogeneous)
1. Distribution Transparency

**Distribution transparency:** allows user to perceive database as a single, logical entity.

If DDBMS exhibits distribution transparency, user does not need to know:

- **fragmentation transparency:** data is fragmented
- **Location transparency:** location of data items
- otherwise call this local mapping transparency
- **replication transparency:** user unaware of replication of fragments

**Naming transparency:** each item in a DDB must have a unique name.

- One solution: create central name server - loss of some local autonomy.
- central site may become a bottleneck. - low availability: if the central site fails.

Alternative solution: prefix object with identifier of creator site, each fragment and its copies. Then each site uses alias.
2. Transaction Transparency

**Transaction transparency:** Ensures all distributed Ts maintain distributed database’s integrity and consistency.

- Distributed T accesses data stored at more than one location.
- Each T is divided into no. of subTs, one for each site that has to be accessed.
- DDBMS must ensure the indivisibility of both the global T and each of the subTs.
2. Transaction Transparency

Concurrency transparency: All Ts must execute independently and be logically consistent with results obtained if Ts executed in some arbitrary serial order.

- Replication makes concurrency more complex

Failure transparency: must ensure atomicity and durability of global T.

- Means ensuring that subTs of global T either all commit or all abort.

Classification transparency: In IBM’s Distributed Relational Database Architecture (DRDA), four types of Ts:
- Remote request (Application Server, ASP protocol)
- Remote unit of work
- Distributed unit of work (Database Server, DSP Protocol)
- Distributed request.
3. Performance Transparency

**DDBMS:** - no performance degradation due to distributed architecture.
  - determine most cost-effective strategy to execute a request.

**Distributed Query Processor (DQP)** maps data request into ordered sequence of operations on local databases.
  - Must consider fragmentation, replication, and allocation schemas.

DQP has to decide:
1. which fragment to access
2. which copy of a fragment to use
3. which location to use.

- produces execution strategy optimized with respect to some cost function.

Typically, **costs** associated with a distributed request include: I/O cost; CPU cost, communication cost.
Dates 12 Rules for DDBMS

**Fundamental Principle:** To the user, distributed system should look exactly like a nondistributed system.

1. Local Autonomy
2. No Reliance on a Central Site
3. Continuous Operation
4. Location Independence
5. Fragmentation Independence
6. Replication Independence
7. Distributed Query Processing
8. Distributed Transaction Processing

**Ideals:**
9. Hardware Independence
10. Operating System Independence
11. Network Independence
12. Database Independence
Naming of Data Items - Criteria

1. Every data item must have a system-wide unique name.
2. It should be possible to find the location of data items efficiently.
3. It should be possible to change the location of data items transparently.
4. Each site should be able to create new data items autonomously.
Centralized Scheme - Name Server

- **Structure:**
  - name server assigns all names
  - each site maintains a record of local data items
  - sites ask name server to locate non-local data items

- **Advantages:**
  - satisfies naming criteria 1-3

- **Disadvantages:**
  - does not satisfy naming criterion 4
  - name server is a potential performance bottleneck
  - name server is a single point of failure
Use of Aliases

- Alternative to centralized scheme: each site prefixes its own site identifier to any name that it generates i.e., site 17.account.
  - Fulfills having a unique identifier, and avoids problems associated with central control.
  - However, fails to achieve network transparency.

- Solution: Create a set of aliases for data items; Store the mapping of aliases to the real names at each site.

- The user can be unaware of the physical location of a data item, and is unaffected if the data item is moved from one site to another.
Distributed Transactions

- Transaction may access data at several sites.
- Each site has a local transaction manager responsible for:
  - Maintaining a log for recovery purposes
  - Participating in coordinating the concurrent execution of the transactions executing at that site.
- Each site has a transaction coordinator, which 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, which may result in the transaction being committed at all sites or aborted at all sites.
Transaction System Architecture

Transaction coordinating

Transaction manager

Computer 1

Computer n
Distributed Transaction Management

**Objectives** of distributed T processing same as centralized.
– more complex: ensure atomicity of global T and each sub T

Previously:
Four CENTRALIZED database modules:
1. **Transaction manager**
2. **Scheduler:** (or **Lock manager**)
3. **Recovery manager**
4. **Buffer manager**

- DDBMS has these in local DBMSs
- In addition: **Global Transaction Manager** (or transaction coordinator **TC**) at each site.
  - TC coordinates local and global Ts initiated at that site.
  - Inter-site communication still through **Data communications**
Two Phase Commit Protocol (2PC)

- Assumes **fail-stop** model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.
- 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
- 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 $\langle\text{prepare } T\rangle$ to the log and forces log to stable storage
  - sends $\text{prepare } T$ 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 $\langle\text{no } T\rangle$ to the log and send abort $T$ message to $C_i$
  - if the transaction can be committed, then:
    - add the record $\langle\text{ready } T\rangle$ to the log
    - force all records for $T$ to stable storage
    - send ready $T$ message to $C_i$
Phase 2: Recording the Decision

- If $T$ has been committed, $C_i$ received a ready $T$ message from all the participating sites: otherwise $T$ must be aborted.

- Coordinator adds a decision record, `<commit T>` or `<abort T>`, to the log and forces record onto stable storage. Once the record is 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 phases: a voting phase and a decision phase.

- Coordinator asks all participants if they are prepared to commit T.
  - If one participant votes abort, or fails to respond within a timeout period, coordinator instructs all participants to abort transaction.
  - If all vote commit, coordinator instructs all participants to commit.

All participants must adopt global decision.

- Assumes each site has own local log and can rollback or commit T reliably.
- If participant fails to vote, abort is assumed.
- If participant gets no vote instruction from coordinator, can abort.

State transitions 2PL.
(a) Coordinator (b) Participant
Termination Protocols: invoked whenever a coordinator or participant fails to receive an expected message and times out.

Coordinator
- Timeout in WAITING state - Globally abort the transaction.
- Timeout in DECIDED state - Send global decision again to sites that have not acknowledged.

Participant
Simplest termination protocol: leave participant blocked until comm with coordinator is re-established. Alternatively:
- Timeout in INITIAL state - Unilaterally abort the transaction.
- Timeout in the PREPARED state - Without more information, participant blocked. Could get decision from another participant.
2-Phase commit

**Recovery Protocols:** Action to be taken by operational site in event of failure. Depends on what stage coordinator or participant had reached.

**Coordinator Failure**
- Failure in INITIAL state procedure - Recovery starts commit procedure.
- Failure in WAITING state procedure - Recovery restarts commit procedure.
- Failure in DECIDED state - On restart, if coordinator received all, complete successfully. Otherwise, has to initiate termination protocol.

**Participant Failure**
Objective to ensure that participant on restart performs same action as all other participants and that this restart can be performed independently.
- Failure in INITIAL state - Unilaterally abort the transaction.
- Failure in PREPARED state - Recovery via termination protocol.
- Failure in ABORTED/COMMITTED states - On restart, no further action.
2PC is not a non-blocking protocol.

For example, a process that times out after voting commit, but before receiving global instruction, is blocked if it can communicate only with sites that do not know global decision.

Existing systems use 2PC.

3PC introduces 3rd phase: **pre-commit**
- On receiving all votes from participants, coordinator sends global pre-commit message.
- Participant who receives global pre-commit, knows all other participants have voted commit and that, in time, participant itself will definitely commit.
X/Open DTP Model

Open Group is vendor-neutral consortium. Mission: cause creation of viable, global information infrastructure.

- Formed by merge of X/Open and Open Software Foundation.
- X/Open established DTP Working Group with objective of specifying and fostering appropriate APIs for TP.
- Group concentrated on TP elements that provided **ACID** properties.

X/Open DTP standard that emerged specified three interacting components:

- an application
- a transaction manager (TM)
- a resource manager (RM)
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

- System maintains a *single* lock manager that resides in a *single* chosen site, say $S_i$
- When a transaction needs to lock a data item, it sends a lock request to $S_i$ and lock manager determines whether the lock can be granted immediately
  - If yes, lock manager sends a message to the site which initiated the request
  - If no, request is delayed until it can be granted, at which time a message is sent to the initiating site
Single-Lock-Manager Approach

- 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 this approach, functionality of locking is implemented by lock managers at each site
  - Lock managers control access to local data items
    - But special protocols may be used for replicas
- Advantage: work is distributed and can be made robust to failures
- Disadvantage: deadlock detection is more complicated
  - Lock managers cooperate for deadlock detection

- Several variants of this approach
  - Primary copy
  - Majority protocol
  - Biased protocol
  - Quorum consensus
Choose one replica of data item to be the primary copy.
- Site containing the 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 Protocol

- 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.
Majority Protocol (Cont.)

- 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.

- Benefit
  - Can be used even when some sites are unavailable

- 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 - e.g., each of 3 transactions may have locks on 1/3rd of the replicas of a data.
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 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 Consensus Protocol

- A generalization 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 so that the sum of the site weights is $\geq Q_r$
- Each write must lock enough replicas so that the sum of the site weights is $\geq Q_w$
Timestamping

- Timestamp based concurrency-control protocols can be used in distributed systems
- Each transaction must be given a unique timestamp
- Main problem: how to generate a timestamp in a distributed fashion
  - Each site generates a unique local timestamp using either a logical counter or the local clock.
  - Global unique timestamp is obtained by concatenating the unique local timestamp with the unique identifier.
A site with a slow clock will assign smaller timestamps
- Still logically correct: serializability not affected
- Disadvantage for transaction processing.

To fix this problem
- Define within each site $S_i$ a logical clock ($LC_i$), which generates the unique local timestamp
- Require that $S_i$ advance its logical clock whenever a request is received from a transaction $T_i$ with timestamp $< x, y>$ and $x$ is greater than the current value of $LC_i$.
- In this case, site $S_i$ advances its logical clock to the value $x + 1$. 
**Timestamp Protocols**

**Objective:** order Ts globally so older Ts (smaller timestamps) get priority in event of conflict.

**Distributed:**
- Need to generate unique timestamps both locally and globally.
- System clock/event counter at each site unsuitable.
- Concatenate local timestamp with a unique site identifier:
  
  \[ <\text{local timestamp}, \text{site identifier}> \]

  Site identifier placed in least significant position - ensures events ordered according to occurrence not location.

- To prevent busy site generating larger timestamps than slower sites:
  - Each site includes their timestamps in messages.
  - Site compares timestamps with message and, if its timestamp smaller, sets it to some value greater than message timestamp.
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.
Handling of Failures - Site Failure

When site \( S_i \) 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 \( T \) replies 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 \( (T) \)
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 $T$. Can therefore 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.
Alternative Models of Transaction Processing

- Notion of a single transaction spanning multiple sites is inappropriate for many applications
  - E.g. transaction crossing an organizational boundary
  - No organization would like to permit an externally initiated transaction to block local transactions for an indeterminate period
- Alternative models carry out transactions by sending messages
  - Code to handle messages must be carefully designed to ensure atomicity and durability properties for updates
    - Isolation cannot be guaranteed, in that intermediate stages are visible, but code must ensure no inconsistent states result due to concurrency
  - **Persistent messaging systems** are systems that provide transactional properties to messages
    - Messages are guaranteed to be delivered exactly once
Alternative Models

Motivating example: funds transfer between two banks

- Two phase commit would have the potential to block updates on the accounts involved in funds transfer
- Alternative solution:
  - Debit money from source account and send a message to other site
  - Site receives message and credits destination account
- Messaging has long been used for distributed transactions (even before computers were invented!)

Atomicity issue

- Once transaction sending a message is committed, message must guaranteed to be delivered
  - Guarantee as long as destination site is up and reachable, code to handle undeliverable messages must also be available
    - e.g. credit money back to source account.
- If sending transaction aborts, message must not be sent
Error Conditions with Persistent Messaging

- Code to handle messages has to take care of variety of failure situations (even assuming guaranteed message delivery)
  - E.g. if destination account does not exist, failure message must be sent back to source site
  - When failure message is received from destination site, or destination site itself does not exist, money must be deposited back in source account
    - Problem if source account has been closed
      - get humans to take care of problem
- User code executing transaction processing using 2PC does not have to deal with such failures
- There are many situations where extra effort of error handling is worth the benefit of absence of blocking
  - E.g. pretty much all transactions across organizations
Persistent Messaging and Workflows

- **Workflows** provide a general model of transactional processing involving multiple sites and possibly human processing of certain steps
  - E.g. when a bank receives a loan application, it may need to
    - Contact external credit-checking agencies
    - Get approvals of one or more managers
  and then respond to the loan application
- Persistent messaging forms the underlying infrastructure for workflows in a distributed environment
Replication with Weak Consistency

- Many commercial databases support replication of data with weak degrees of consistency (i.e., without a guarantee of serializability)
- E.g.: master-slave replication: updates are performed at a single “master” site, and propagated to “slave” sites.
  - Propagation is not part of the update transaction: it is decoupled
    - May be immediately after transaction commits
    - May be periodic
  - Data may only be read at slave sites, not updated
    - No need to obtain locks at any remote site
  - Particularly useful for distributing information
    - E.g. from central office to branch-office
  - Also useful for running read-only queries offline from the main database
Replication with Weak Consistency (Cont.)

- Replicas should see a **transaction-consistent snapshot** of the database
  - That is, a state of the database reflecting all effects of all transactions up to some point in the serialization order, and no effects of any later transactions.

- E.g. Oracle provides a **create snapshot** statement to create a snapshot of a relation or a set of relations at a remote site
  - snapshot refresh either by recomputation or by incremental update
  - Automatic refresh (continuous or periodic) or manual refresh
Multimaster and Lazy Replication

- With multimaster replication (also called update-anywhere replication) updates are permitted at any replica, and are automatically propagated to all replicas
  - Basic model in distributed databases, where transactions are unaware of the details of replication, and database system propagates updates as part of the same transaction
    - Coupled with 2 phase commit
- Many systems support **lazy propagation** where updates are transmitted after transaction commits
  - Allows updates to occur even if some sites are disconnected from the network, but at the cost of consistency
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:

$$
T_1: \quad \text{write (X)} \\
\phantom{T_1:} \quad \text{write (Y)}
$$

$$
T_2: \quad \text{write (Y)} \\
\phantom{T_2:} \quad \text{write (X)}
$$

<table>
<thead>
<tr>
<th>X-lock on X</th>
<th>X-lock on Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>write (X)</td>
<td>write (Y)</td>
</tr>
<tr>
<td>wait for X-lock on X</td>
<td>wait for X-lock on X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wait for X-lock on Y</th>
</tr>
</thead>
</table>

Result: deadlock which cannot be detected locally at either site.
Centralized 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

Local

- Site $S_1$
  - $T_1 \rightarrow T_2 \rightarrow T_3$, $T_5 \rightarrow T_2$

- Site $S_2$
  - $T_2 \rightarrow T_4

Global

- $T_1 \rightarrow T_2 \rightarrow T_4$
- $T_5 \rightarrow T_2 \rightarrow T_3$
Example Wait-For Graph for False Cycles

Initial state:

- **S₁**:
  - $T_1$ → $T_2$

- **S₂**:
  - $T_1$ → $T_3$

- Coordinator:
  - $T_1$ → $T_2$ → $T_3$
False Cycles (Cont.)

- Suppose that starting from the state shown in figure,
  1. $T_2$ releases resources at $S_1$
     - resulting in a message remove $T_1 \rightarrow T_2$ message from the Transaction Manager at site $S_1$ to the coordinator
  2. And then $T_2$ requests a resource held by $T_3$ at site $S_2$
     - resulting in a message insert $T_2 \rightarrow T_3$ from $S_2$ to the coordinator
- Suppose further that the insert message reaches before the delete message
  - this can happen due to network delays
- The coordinator would then find a false cycle
  \[
  T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1
  \]
- The false cycle above never existed in reality.
- False cycles cannot occur if two-phase locking is used.
Unnecessary Rollbacks

- Unnecessary rollbacks may result when deadlock has indeed occurred and a victim has been picked, and meanwhile one of the transactions was aborted for reasons unrelated to the deadlock.

- Unnecessary rollbacks can result from false cycles in the global wait-for graph; however, likelihood of false cycles is low.
Availability

- High availability: time for which system is not fully usable should be extremely low (e.g. 99.99% availability)
- Robustness: ability of system to function spite of failures of components
- Failures are more likely in large distributed systems
- To be robust, a distributed system must
  - Detect failures
  - Reconfigure the system so computation may continue
  - Recovery/reintegration when a site or link is repaired
- Failure detection: distinguishing link failure from site failure is hard
  - (partial) solution: have multiple links, multiple link failure is likely a site failure
Reconfiguration:

- Abort all transactions that were active at a failed site
  - Making them wait could interfere with other transactions since they may hold locks on other sites
  - However, in case only some replicas of a data item failed, it may be possible to continue transactions that had accessed data at a failed site.
- If replicated data items were at failed site, update system catalog to remove them from the list of replicas.
  - This should be reversed when failed site recovers, but additional care needs to be taken to bring values up to date
- If a failed site was a central server for some subsystem, an election must be held to determine the new server
  - E.g. name server, concurrency coordinator, global deadlock detector
Since network partition may not be distinguishable from site failure, the following situations must be avoided:

- Two or more central servers elected in distinct partitions
- More than one partition updates a replicated data item

Updates must be able to continue even if some sites are down

Solution: majority based approach

- Alternative of “read one write all available” is tantalizing but causes problems
Majority-Based Approach

- The majority protocol for distributed concurrency control can be modified to work even if some sites are unavailable
  - Each replica of each item has a **version number** which is updated when the replica is updated, as outlined below
  - A lock request is sent to at least ½ the sites at which item replicas are stored and operation continues only when a lock is obtained on a majority of the sites
  - Read operations look at all replicas locked, and read the value from the replica with largest version number
    - May write this value and version number back to replicas with lower version numbers (no need to obtain locks on all replicas for this task)
Majority-Based Approach

- Majority protocol (Cont.)
  - Write operations
    - find highest version number like reads, and set new version number to old highest version + 1
    - Writes are then performed on all locked replicas and version number on these replicas is set to new version number
  - Failures (network and site) cause no problems as long as
    - Sites at commit contain a majority of replicas of any updated data items
    - During reads a majority of replicas are available to find version numbers
    - Subject to above, 2 phase commit can be used to update replicas
  - Note: reads are guaranteed to see latest version of data item
  - Reintegration is trivial: nothing needs to be done
  - Quorum consensus algorithm can be similarly extended
Biased protocol is a special case of quorum consensus
- Allows reads to read any one replica but updates require all replicas to be available at commit time (called **read one write all**)

Read one write all available (ignoring failed sites) is attractive, but incorrect
- If failed link may come back up, without a disconnected site ever being aware that it was disconnected
- The site then has old values, and a read from that site would return an incorrect value
- If site was aware of failure reintegration could have been performed, but no way to guarantee this
- With network partitioning, sites in each partition may update same item concurrently
  - believing sites in other partitions have all failed
Site Reintegration

- When failed site recovers, it must catch up with all updates that it missed while it was down
  - Problem: updates may be happening to items whose replica is stored at the site while the site is recovering
  - Solution 1: halt all updates on system while reintegrating a site
    - Unacceptable disruption
  - Solution 2: lock all replicas of all data items at the site, update to latest version, then release locks
    - Other solutions with better concurrency also available
Comparison with Remote Backup

- Remote backup (hot spare) systems are also designed to provide high availability.

- Remote backup systems are simpler and have lower overhead:
  - All actions performed at a single site, and only log records shipped
  - No need for distributed concurrency control, or 2 phase commit

- Using distributed databases with replicas of data items can provide higher availability by having multiple (> 2) replicas and using the majority protocol:
  - Also avoid failure detection and switchover time associated with remote backup systems
Coordinator Selection

■ Backup coordinators
  ● site which maintains enough information locally to assume the role of coordinator if the actual coordinator fails
  ● executes the same algorithms and maintains the same internal state information as the actual coordinator fails executes state information as the actual coordinator
  ● allows fast recovery from coordinator failure but involves overhead during normal processing.

■ Election algorithms
  ● used to elect a new coordinator in case of failures
  ● Example: Bully Algorithm - applicable to systems where every site can send a message to every other site.
Bully Algorithm

- If site $S_i$ sends a request that is not answered by the coordinator within a time interval $T$, assume that the coordinator has failed $S_i$ tries to elect itself as the new coordinator.

- $S_i$ sends an election message to every site with a higher identification number, $S_i$ then waits for any of these processes to answer within $T$.

- If no response within $T$, assume that all sites with number greater than $i$ have failed, $S_i$ elects itself the new coordinator.

- If answer is received $S_i$ begins time interval $T$, waiting to receive a message that a site with a higher identification number has been elected.
Bully Algorithm (Cont.)

- If no message is sent within $T'$, assume the site with a higher number has failed; $S_i$ restarts the algorithm.
- After a failed site recovers, it immediately begins execution of the same algorithm.
- If there are no active sites with higher numbers, the recovered site forces all processes with lower numbers to let it become the coordinator site, even if there is a currently active coordinator with a lower number.
Distributed Query Processing

- For centralized systems, the primary criterion for measuring the cost of a particular strategy is the number of disk accesses.
- In a distributed system, other issues must be taken into account:
  - The cost of a data transmission over the network.
  - The potential gain in performance from having several sites process parts of the query in parallel.
Query Transformation

- Translating algebraic queries on fragments.
  - It must be possible to construct relation \( r \) from its fragments
  - Replace relation \( r \) by the expression to construct relation \( r \) from its fragments

- Consider the horizontal fragmentation of the \textit{account} relation into

\[
account_1 = \sigma_{branch\_name = \text{“Hillside”}} (account)
\]

\[
account_2 = \sigma_{branch\_name = \text{“Valleyview”}} (account)
\]

- The query \( \sigma_{branch\_name = \text{“Hillside”}} (account) \) becomes

\[
\sigma_{branch\_name = \text{“Hillside”}} (account_1 \cup account_2)
\]

which is optimized into

\[
\sigma_{branch\_name = \text{“Hillside”}} (account_1) \cup \sigma_{branch\_name = \text{“Hillside”}} (account_2)
\]
Example Query (Cont.)

- Since \( account_1 \) has only tuples pertaining to the Hillside branch, we can eliminate the selection operation.
- Apply the definition of \( account_2 \) to obtain

\[
\sigma \ branch\_name = "Hillside" \left( \sigma \ branch\_name = "Valleyview" \ \( account \ \right)
\]

- This expression is the empty set regardless of the contents of the \( account \) relation.
- Final strategy is for the Hillside site to return \( account_1 \) as the result of the query.
Simple Join Processing

- Consider the following relational algebra expression in which the three relations are neither replicated nor fragmented:
  \[
  \text{account} \bowtie \text{depositor} \bowtie \text{branch}
  \]
- \text{account} is stored at site \( S_1 \)
- \text{depositor} at \( S_2 \)
- \text{branch} at \( S_3 \)
- For a query issued at site \( S_1 \), the system needs to produce the result at site \( S_1 \)
Possible Query Processing Strategies

- Ship copies of all three relations to site $S_1$ and choose a strategy for processing the entire locally at site $S_1$.

- Ship a copy of the account relation to site $S_2$ and compute $temp_1 = account \Join depositor$ at $S_2$. Ship $temp_1$ from $S_2$ to $S_3$, and compute $temp_2 = temp_1$ branch at $S_3$. Ship the result $temp_2$ to $S_1$.

- Devise similar strategies, exchanging the roles $S_1$, $S_2$, $S_3$

- Must consider following factors:
  - amount of data being shipped
  - cost of transmitting a data block between sites
  - relative processing speed at each site
Semijoin Strategy

- Let $r_1$ be a relation with schema $R_1$ stores at site $S_1$
  - Let $r_2$ be a relation with schema $R_2$ stores at site $S_2$
- Evaluate the expression $r_1 \bowtie r_2$ and obtain the result at $S_1$.
  1. Compute $temp_1 \leftarrow \prod_{R_1 \cap R_2} (r_1)$ at $S_1$.
  2. Ship $temp_1$ from $S_1$ to $S_2$.
  3. Compute $temp_2 \leftarrow r_2 \bowtie temp_1$ at $S_2$.
  4. Ship $temp_2$ from $S_2$ to $S_1$.
  5. Compute $r_1 \bowtie temp_2$ at $S_1$. This is the same as $r_1 \bowtie r_2$. 
Join Strategies that Exploit Parallelism

- Consider $r_1 \bowtie r_2 \bowtie r_3$ \bowtie $r_4$ where relation $r_i$ is stored at site $S_i$. The result must be presented at site $S_1$.

- $r_1$ is shipped to $S_2$ and $r_1 \bowtie r_2$ is computed at $S_2$: simultaneously $r_3$ is shipped to $S_4$ and $r_3 \bowtie r_4$ is computed at $S_4$.

- $S_2$ ships tuples of $(r_1 \bowtie r_2)$ to $S_1$ as they produced; $S_4$ ships tuples of $(r_3 \bowtie r_4)$ to $S_1$.

- Once tuples of $(r_1 \bowtie r_2)$ and $(r_3 \bowtie r_4)$ arrive at $S_1$ $(r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_4)$ is computed in parallel with the computation of $(r_1 \bowtie r_2)$ at $S_2$ and the computation of $(r_3 \bowtie r_4)$ at $S_4$. 
Many database applications require data from a variety of preexisting databases located in a heterogeneous collection of hardware and software platforms.

Data models may differ (hierarchical, relational, etc.).

Transaction commit protocols may be incompatible.

Concurrency control may be based on different techniques (locking, timestamping, etc.).

System-level details almost certainly are totally incompatible.

A multidatabase system is a software layer on top of existing database systems, which is designed to manipulate information in heterogeneous databases.

- Creates an illusion of logical database integration without any physical database integration.
Advantages

- Preservation of investment in existing
  - hardware
  - system software
  - Applications
- Local autonomy and administrative control
- Allows use of special-purpose DBMSs
- Step towards a unified homogeneous DBMS
  - Full integration into a homogeneous DBMS faces
    - Technical difficulties and cost of conversion
    - Organizational/political difficulties
      - Organizations do not want to give up control on their data
      - Local databases wish to retain a great deal of autonomy
Unified View of Data

- Agreement on a common data model
  - Typically the relational model
- Agreement on a common conceptual schema
  - Different names for same relation/attribute
  - Same relation/attribute name means different things
- Agreement on a single representation of shared data
  - E.g. data types, precision,
  - Character sets
    - ASCII vs EBCDIC
    - Sort order variations
- Agreement on units of measure
- Variations in names
  - E.g. Köln vs Cologne, Mumbai vs Bombay
Query Processing

- Several issues in query processing in a heterogeneous database
- Schema translation
  - Write a **wrapper** for each data source to translate data to a global schema
  - Wrappers must also translate updates on global schema to updates on local schema
- Limited query capabilities
  - Some data sources allow only restricted forms of selections
    - E.g. web forms, flat file data sources
  - Queries have to be broken up and processed partly at the source and partly at a different site
- Removal of duplicate information when sites have overlapping information
  - Decide which sites to execute query
- Global query optimization
Mediator Systems

- **Mediator** systems are systems that integrate multiple heterogeneous data sources by providing an integrated global view, and providing query facilities on global view
  - Unlike full fledged multidatabase systems, mediators generally do not bother about transaction processing
  - But the terms mediator and multidatabase are sometimes used interchangeably
  - The term **virtual database** is also used to refer to mediator/multidatabase systems
Directory Systems

- Typical kinds of directory information
  - Employee information such as name, id, email, phone, office addr, ..
  - Even personal information to be accessed from multiple places
    - e.g. Web browser bookmarks

- White pages
  - Entries organized by name or identifier
    - Meant for forward lookup to find more about an entry

- Yellow pages
  - Entries organized by properties
  - For reverse lookup to find entries matching specific requirements

- When directories are to be accessed across an organization
  - Alternative 1: Web interface. Not great for programs
  - Alternative 2: Specialized directory access protocols
    - Coupled with specialized user interfaces
Directory Access Protocols

- Most commonly used directory access protocol:
  - LDAP (Lightweight Directory Access Protocol)
  - Simplified from earlier X.500 protocol

- Question: Why not use database protocols like ODBC/JDBC?

- Answer:
  - Simplified protocols for a limited type of data access, evolved parallel to ODBC/JDBC
  - Provide a nice hierarchical naming mechanism similar to file system directories
    - Data can be partitioned amongst multiple servers for different parts of the hierarchy, yet give a single view to user
      - E.g. different servers for Bell Labs Murray Hill and Bell Labs Bangalore
  - Directories may use databases as storage mechanism
LDAP: Lightweight Directory Access Protocol

- LDAP Data Model
- Data Manipulation
- Distributed Directory Trees
LDAP Data Model

- LDAP directories store **entries**
  - Entries are similar to objects
- Each entry must have unique **distinguished name (DN)**
- DN made up of a sequence of **relative distinguished names (RDNs)**
- E.g. of a DN
  - cn=Silberschatz, ou-Bell Labs, o=Lucent, c=USA
  - Standard RDNs (can be specified as part of schema)
    - cn: common name  ou: organizational unit
    - o: organization     c: country
  - Similar to paths in a file system but written in reverse direction
LDAP Data Model (Cont.)

- Entries can have attributes
  - Attributes are multi-valued by default
  - LDAP has several built-in types
    - Binary, string, time types
    - Tel: telephone number    PostalAddress: postal address

- LDAP allows definition of **object classes**
  - Object classes specify attribute names and types
  - Can use inheritance to define object classes
  - Entry can be specified to be of one or more object classes
    - No need to have single most-specific type
LDAP Data Model (cont.)

- Entries organized into a **directory information tree** according to their DNs
  - Leaf level usually represent specific objects
  - Internal node entries represent objects such as organizational units, organizations or countries
  - Children of a node inherit the DN of the parent, and add on RDNs
    - E.g. internal node with DN c=USA
      - Children nodes have DN starting with c=USA and further RDNs such as o or ou
    - DN of an entry can be generated by traversing path from root
  - Leaf level can be an alias pointing to another entry
    - Entries can thus have more than one DN
      - E.g. person in more than one organizational unit
LDAP Data Manipulation

- Unlike SQL, LDAP does not define DDL or DML
- Instead, it defines a network protocol for DDL and DML
  - Users use an API or vendor specific front ends
  - LDAP also defines a file format
    - LDAP Data Interchange Format (LDIF)
- Querying mechanism is very simple: only selection & projection
LDAP Queries

- LDAP query must specify
  - Base: a node in the DIT from where search is to start
  - A search condition
    - Boolean combination of conditions on attributes of entries
      - Equality, wild-cards and approximate equality supported
  - A scope
    - Just the base, the base and its children, or the entire subtree from the base
  - Attributes to be returned
  - Limits on number of results and on resource consumption
  - May also specify whether to automatically dereference aliases

- LDAP URLs are one way of specifying query
- LDAP API is another alternative
LDAP URLs

- First part of URL specifies server and DN of base
  - ldap://aura.research.bell-labs.com/o=Lucent,c=USA

- Optional further parts separated by ? symbol
  - ldap://aura.research.bell-labs.com/o=Lucent,c=USA??sub?cn=Verma
  - Optional parts specify
    1. attributes to return (empty means all)
    2. Scope (sub indicates entire subtree)
    3. Search condition (cn=Korth)
```c
#include <stdio.h>
#include <ldap.h>
main( ) {
    LDAP *ld;
    LDAPMessage *res, *entry;
    char *dn, *attr, *attrList[] = {"telephoneNumber", NULL};
    BerElement *ptr;
    int vals, i;
    // Open a connection to server
    ld = ldap_open("aura.research.bell-labs.com", LDAP_PORT);
    ldap_simple_bind(ld, "avi", "avi-passwd");
    ... actual query (next slide) ...
    ldap_unbind(ld);
}
```
C Code using LDAP API (Cont.)

```c
ldap_search_s(ld, "o=Lucent, c=USA", LDAP_SCOPE_SUBTREE,
            "cn=Korth", attrList, /* attrsonly*/ 0, &res);
/*attrsonly = 1 => return only schema not actual results*/
printf("found%d entries", ldap_count_entries(ld, res));
for (entry=ldap_first_entry(ld, res); entry != NULL;
entry=ldap_next_entry(id, entry)) {  
    dn = ldap_get_dn(ld, entry);
    printf("dn: %s", dn);   /* dn: DN of matching entry */
    ldap_memfree(dn);
    for(attr = ldap_first_attribute(ld, entry, &ptr); attr != NULL;
        attr = ldap_next_attribute(ld, entry, ptr))
    {                                       // for each attribute
        printf("%s:", attr);               // print name of attribute
        vals = ldap_get_values(ld, entry, attr);
        for (i = 0; vals[i] != NULL; i ++)
            printf("%s", vals[i]);  // since attrs can be multivalued
        ldap_value_free(vals);
    }
}
ldap_msgfree(res);
```
LDAP API (Cont.)

- LDAP API also has functions to create, update and delete entries
- Each function call behaves as a separate transaction
  - LDAP does not support atomicity of updates
**Distributed Directory Trees**

- Organizational information may be split into multiple directory information trees
  - **Suffix** of a DIT gives RDN to be tagged onto to all entries to get an overall DN
    - E.g. two DITs, one with suffix `o=Lucent, c=USA` and another with suffix `o=Lucent, c=India`
  - Organizations often split up DITs based on geographical location or by organizational structure
  - Many LDAP implementations support replication (master-slave or multi-master replication) of DITs (not part of LDAP 3 standard)

- A node in a DIT may be a **referral** to a node in another DIT
  - E.g. `Ou= Bell Labs` may have a separate DIT, and DIT for `o=Lucent` may have a leaf with `ou=Bell Labs` containing a referral to the Bell Labs DIT
  - Referalls are the key to integrating a distributed collection of directories
  - When a server gets a query reaching a referral node, it may either
    - Forward query to referred DIT and return answer to client, or
    - Give referral back to client, which transparently sends query to referred DIT (without user intervention)
Three Phase Commit (3PC)

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
Figure

- Site $S_1$: $T_1$ and $T_5$ connected, $T_2$ and $T_3$ connected.
- Site $S_2$: $T_2$, $T_3$, and $T_4$ connected.
Figure

\[ S_1 \]

\[ S_2 \]

coordinator
### Figure

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$r$  

$s$