AN HONORS UNIVERSITY IN MARYLAND

CMSC 461, Database Management Systems
Spring 2018

Lecture 24 – Big Data and Distributed Databases

These slides are based on "Database System Concepts" 6th edition book (whereas some quotes and figures are used from the book) and are a modified version of the slides which accompany the book (http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html), in addition to the 2009/2012 CMSC 461 slides by Dr. Kalpakis

Dr. Jennifer Sleeman https://www.csee.umbc.edu/~jsleem1/courses/461/spr18

Logistics

- Homework 6 due 5/2/2018
- Final Project Plan 5/14/2018

Reminder: Presentation Slots
Reminder: Presentation share for slides - LASTNAME.ext

Distributed Databases

Distributed Database System

- A distributed database system consists of loosely coupled sites that share no physical component
- Database systems that run on each site are independent of each other
- Transactions may access data at one or more sites

Homogeneous/Heterogeneous 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

Homogeneous/Heterogeneous Distributed Databases

- 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

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

- Advantages of Replication
  - **Availability**: failure of site containing relation *r* does not result in unavailability of *r* is 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*.

Data Replication

- 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, …, r_n which contain sufficient information to reconstruct relation r.
- **Horizontal fragmentation**: each tuple of *r* is assigned to one or more fragments

Horizontal Fragmentation of *account* Relation

branch_name	*account_number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

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

branch_name	*account_number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

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

Based on and image from “Database System Concepts” book and slides, 6th edition

Data Fragmentation

- **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.

Vertical Fragmentation of *employee_info* Relation

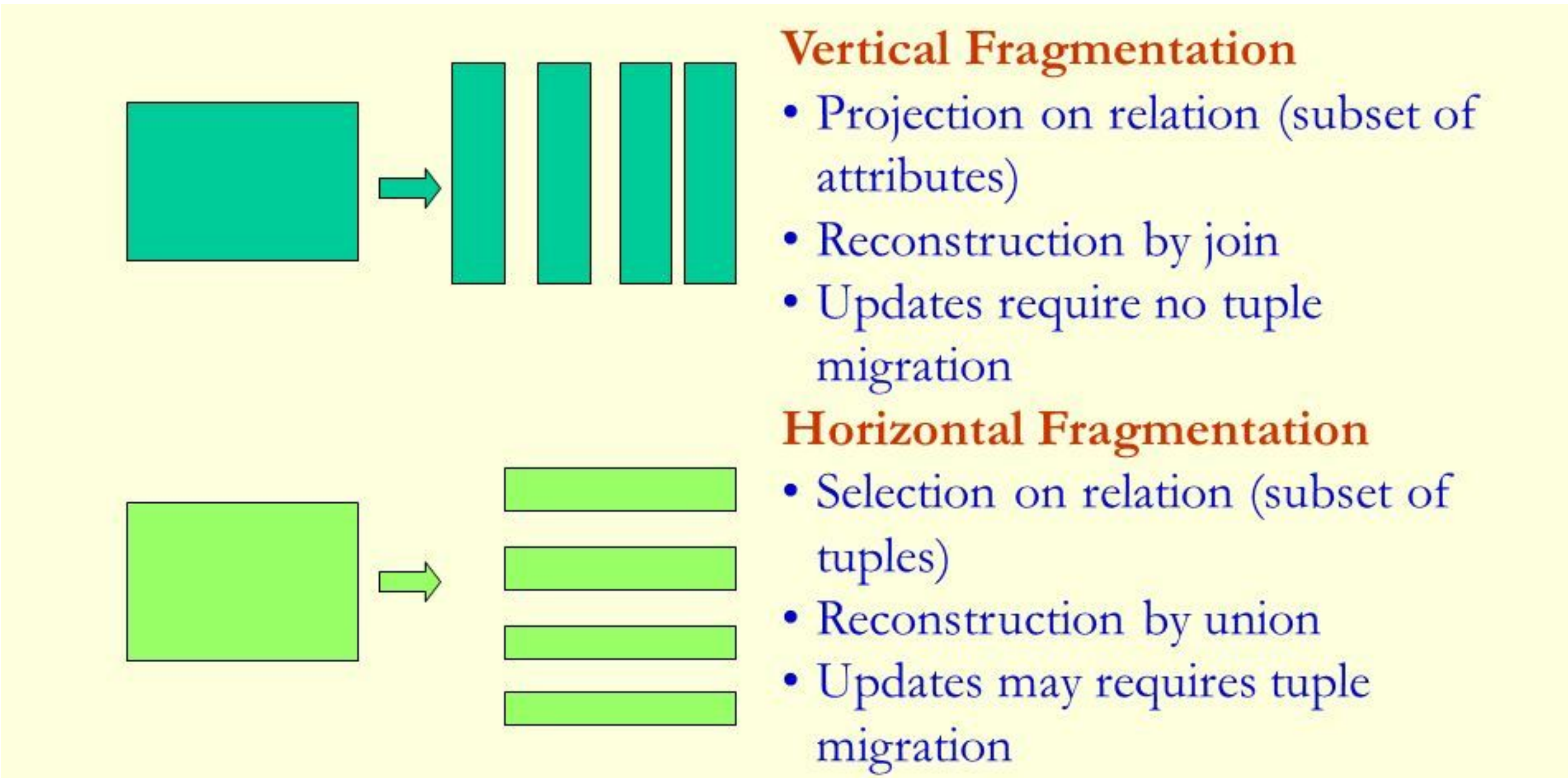
branch_name	*customer_name*	*tuple_id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

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

account_number	*balance*	*tuple_id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$$deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id}(employee\_info)$$

Fragmentation



Picture from “Session-9 Data Management for Decision Support” and based on “Database System Concepts” book and slides, 6th edition

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

Advantages of Fragmentation

- 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.

Data Transparency

- **Data transparency**: Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system
- Consider transparency issues in relation to:
  - Fragmentation transparency
  - Replication transparency
  - Location transparency

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.

Use of Aliases

- 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

Based on and image from “Database System Concepts” book and slides, 6th edition

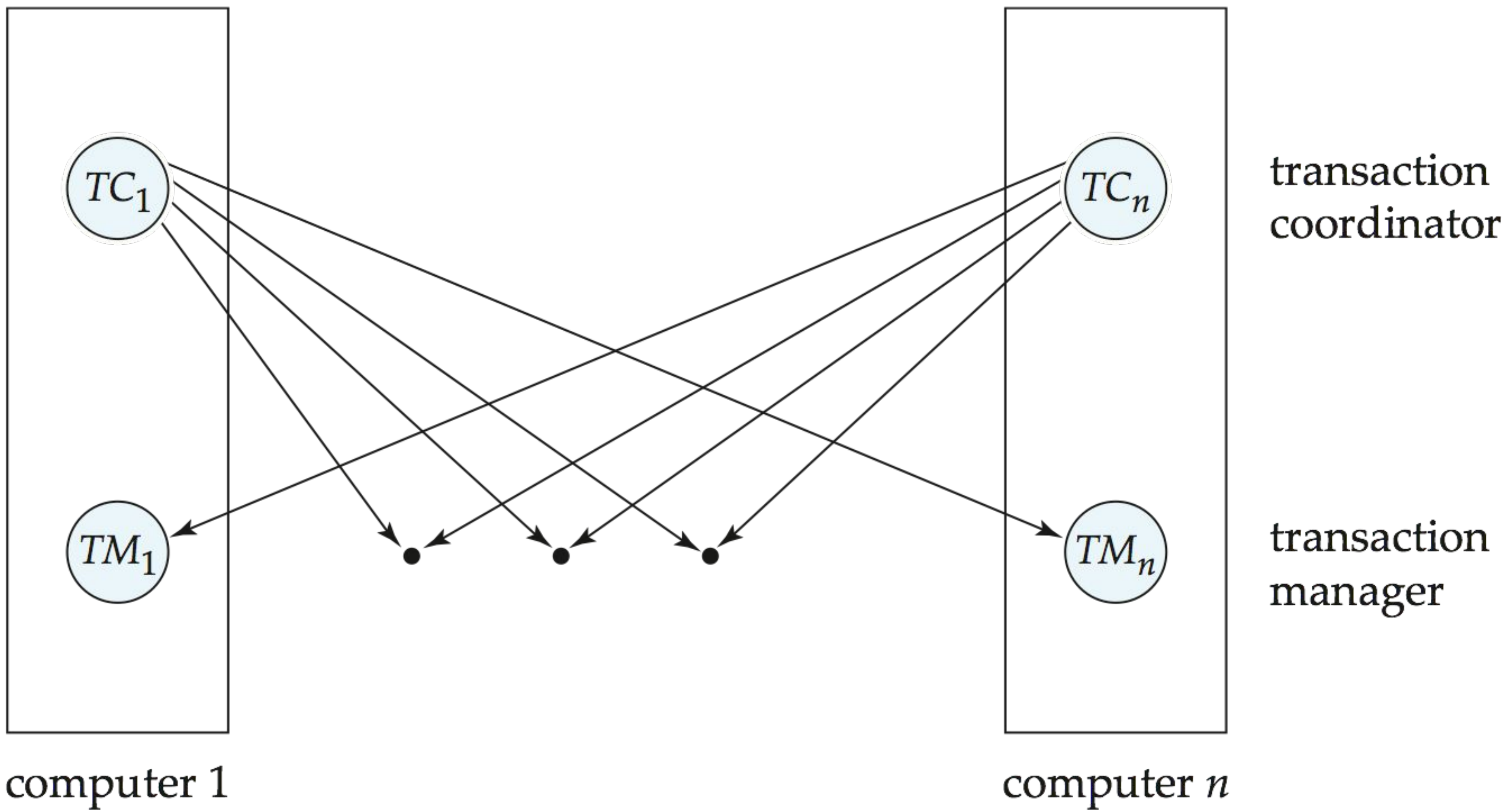
Distributed Transactions

- 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.

Distributed Transactions

- 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



Based on and image from "Database System Concepts" book and slides, 6th edition

System Failure Modes

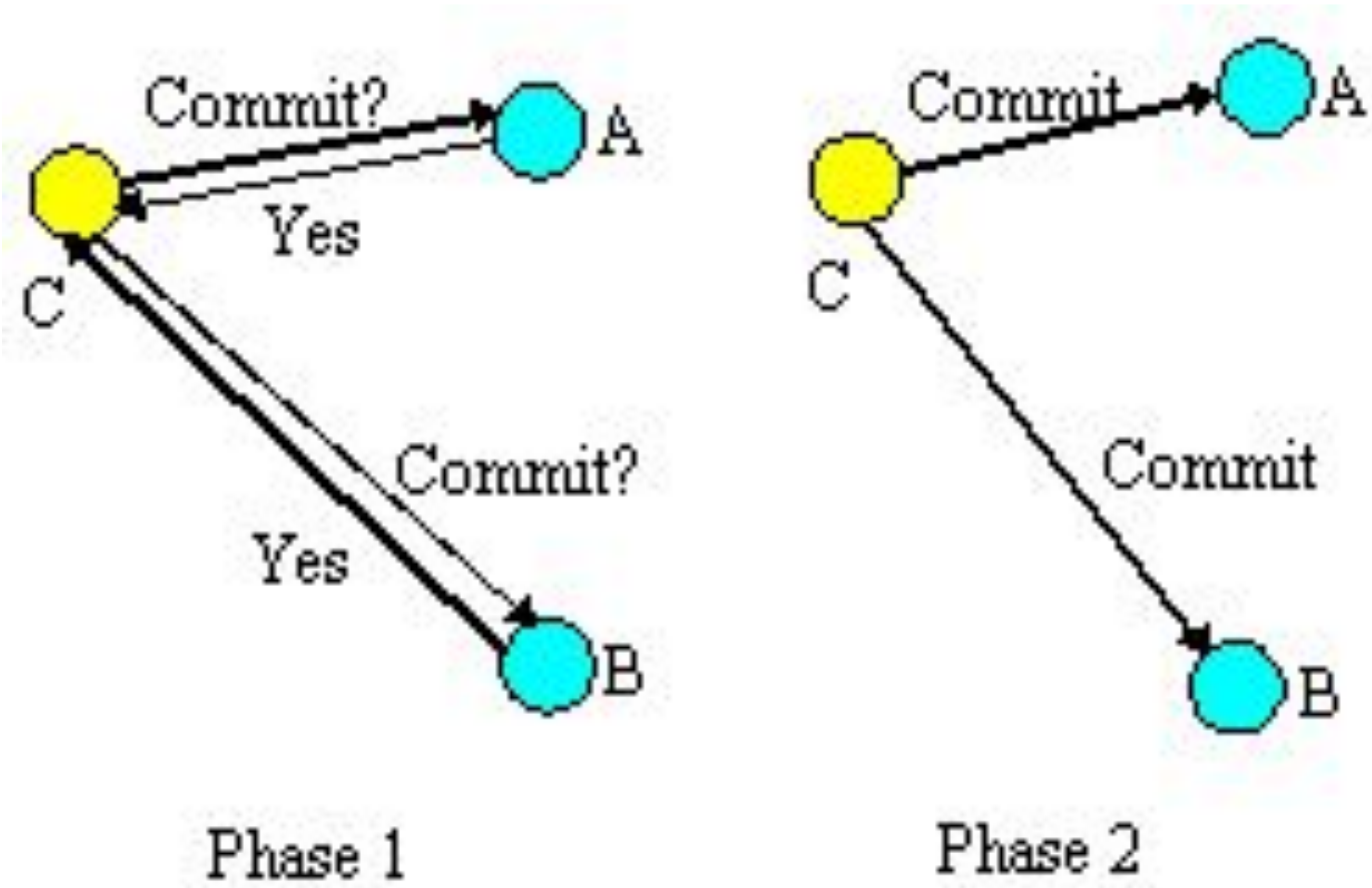
- Failures unique to distributed systems:
  - Failure of a site.
  - Loss of massages
    - Handled by network transmission control protocols such as TCP-IP
  - Failure of a communication link
    - Handled by network protocols, by routing messages via alternative links
  - **Network partition**
   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

Commit Protocols

Commit Protocols

- Commit protocols are used to ensure atomicity across sites
  - a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - not acceptable to have a transaction committed at one site and aborted at another
- The *two-phase commit* (2PC) protocol is widely used
- The *three-phase commit* (3PC) protocol is more complicated and more expensive, but avoids some drawbacks of two-phase commit protocol. This protocol is not used in practice. We will not cover

Two Phase Commit Protocol (2PC)



Picture from "http://www.ra.ethz.ch/CDStore/www6/Technical/Paper012/Paper12.html" Based on "Database System Concepts" book and slides, 6th edition

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

Based on and image from "Database System Concepts" book and slides, 6th edition

Phase 1: Obtaining a Decision

- Coordinator asks all participants to *prepare* to commit transaction T_i.
  - C_i adds the records <**prepare** *T*> to the log and forces log to stable storage
  - sends **prepare** *T* messages to all sites at which *T* executed

Phase 1: Obtaining a Decision

- Upon receiving message, transaction manager at site determines if it can commit the transaction
  - if not, add a record <**no** *T*> to the log and send **abort** *T* message to C_i
  - if the transaction can be committed, then:
  - add the record <**ready** *T*> to the log
  - force *all records* for *T* to stable storage
  - send **ready** *T* message to C_i

Based on and image from "Database System Concepts" book and slides, 6th edition

Phase 2: Recording the Decision

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

Handling of Failures- Site Failure

- When site Si recovers, it examines its log to determine the fate of transactions active at the time of the failure.
  - Log contain <commit T> record: txn had completed, nothing to be done
  - Log contains <abort T> record: txn had completed, nothing to be done
  - Log contains <ready T> record: site must consult Ci to determine the fate of T.
    - If T committed, redo (T); write <commit T> record
    - If T aborted, undo (T)
  - The log contains no log records concerning T:
    - Implies that Sk failed before responding to the prepare T message from Ci
    - since the failure of Sk precludes the sending of such a response, coordinator C1 must abort T
    - Sk must execute undo (T)

Handling of Failures- Coordinator Failure

- If coordinator fails while the commit protocol for *T* is executing then participating sites must decide on *T*'s fate:
    - If an active site contains a <**commit** *T*> record in its log, then *T* must be committed.
    - If an active site contains an <**abort** *T*> record in its log, then *T* must be aborted.
    - 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;* however, such a site must reject any subsequent <**prepare** *T*> message from C_i
    - 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.

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.

Concurrency Control

- Concurrency Control schemes can be modified to work with distributed databases
- Assuming a commit protocol and participation of sites in execution
- Goal is to ensure global transaction atomicity

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 Approach

- 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

Distributed Lock Manager Approach

- 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

Based on and image from “Database System Concepts” book and slides, 6th edition

Replication with Weak Consistency

- Many commercial databases support replication of data with weak degrees of consistency (I.e., without a guarantee of serializabiliy)

Replication with Weak Consistency

- 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: its 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

- 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.

Multimaster and Lazy Replication

- 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

Heterogeneous Distributed Databases

Heterogeneous Distributed Databases

- 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

Heterogeneous Distributed Databases

- 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

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

Query Processing

- 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

Cloud-Based Databases

Data Storage on the Cloud

- Need to store and retrieve massive amounts of data
- Traditional parallel databases not designed to scale to 1000’s of nodes (and expensive)
- Initial needs did not include full database functionality
  - Store and retrieve data items by key value is minimum functionality
    - **Key-value stores**

Data Storage on the Cloud

- Several implementations
  - Bigtable from Google,
  - HBase, an open source clone of Bigtable
  - Dynamo, which is a key-value storage system from Amazon
  - Cassandra, from FaceBook
  - Sherpa/PNUTS from Yahoo!

Key Value Stores

- Key-value stores support
  - put(key, value): used to store values with an associated key,
  - get(key): which retrieves the stored value associated with the specified key.
- Some systems such as Bigtable additionally provide range queries on key values
- Multiple versions of data may be stored, by adding a timestamp to the key

Data Representation

- Records in many big data applications need to have a flexible schema
  - Not all records have same structure
  - Some attributes may have complex substructure
- XML and JSON data representation formats widely used
- An example of a JSON object is:

```
{
    "ID": "22222",
    "name": {
          "firstname: "Albert",
          "lastname: "Einstein"
    },
    "deptname": "Physics",
    "children": [
          { "firstname": "Hans", "lastname": "Einstein" },
          { "firstname": "Eduard", "lastname": "Einstein" }
    ]
}
```

-

Partitioning and Retrieving Data

- Key-value stores partition data into relatively small units (hundreds of megabytes).
- These partitions are often called tablets (a tablet is a fragment of a table)
- Partitioning of data into tablets is dynamic:
  - as data are inserted, if a tablet grows too big, it is broken into smaller parts
  - if the load (get/put operations) on a tablet is excessive, the tablet may be broken into smaller tablets, which can be distributed across two or more sites to share the load.
  - the number of tablets is much larger than the number of sites
    - similar to virtual partitioning in parallel databases
- Each get/put request must be routed to the correct site

Partitioning and Retrieving Data

- Partitioning dynamic
  - If tablet grows too big broken up into smaller parts
  - If tablet load too large broken up into smaller parts
- **Tablet controller** tracks the partitioning function and tablet-to-site mapping
  - map a get() request to one or more tablets,
  - Tablet mapping function to track which site responsible for which tablet
- Requests must be routed to correct site
  - Mapping information can be replicated on a set of router sites

Transaction and Replication

- Transactions
  - Data storage systems do not typically support full ACID transactions
  - Cannot support transactionally consistent secondary index
  - Support transactions on data within a single tablet
- Replication
  - Tablets are replicated to multiple machines in a cluster
  - Data likely to be available even if machine in cluster goes down
    - Cluster – a collection of machines in a data center
  - Replication also used across geographically distributed clusters
  - When a site fails tablet is reassigned to a different site that has copy of tablet
    - Becomes the new master site for the tablet
  - Entire data center can become unavailable
    - Replication at a remote site essential for high availability

Traditional Databases on the Cloud

- Extensive use of virtual machines
- Virtual machines very good for applications that are easily parallelized
- Each VM can run database locally
  - Behaves similar to homogeneous distributed database system

Challenges

- Require frequent communication and coordination among sites
  - To access data on another physical machine
  - To obtain locks on remote data
  - To ensure atomic transaction commit using 2 phase commit
- Physical location of data under control of vendor
- Query optimization based on physical location
  - Without knowledge optimizater relies on estimates
- Replication further complicates cloud based data management