Lecture #22

Intro to Distributed Databases
ADMINISTRIVIA

HW #5 is released. Due Sat April 20, 2024 @ 11:59pm

Project #4 is released. Due Sun April 28, 2024 @ 11:59pm
→ Checkpoint to keep you on track and not wait till the last minute

Final Exam
→ Thu May 2, 2024 @ 05:30pm-08:30pm

Lectures #23 and #24
→ Recorded lectures and will be posted next week

Lecture #26: Guest Speaker from Snowflake
→ Devin Petersohn on “Beyond SQL: Dataframes in the Database”
COURSE STATUS

Databases are hard.
Distributed databases are harder.
COURSE STATUS

Databases are hard.
Distributed databases are harder.
Databases are hard.
Distributed databases are harder.
PARALLEL VS. DISTRIBUTED

Parallel DBMSs:
→ Nodes are physically close to each other.
→ Nodes connected with high-speed LAN.
→ Communication cost is assumed to be small.

Distributed DBMSs:
→ Nodes can be far from each other.
→ Nodes connected using public network.
→ Communication cost and problems cannot be ignored.
DISTRIBUTED DBMSs

Use the building blocks that we covered in single-node DBMSs to now support transaction processing and query execution in distributed environments.

→ Optimization & Planning
→ Concurrency Control
→ Logging & Recovery
TODAY'S AGENDA

System Architectures
Design Issues
Partitioning Schemes
Distributed Concurrency Control
A distributed DBMS's system architecture specifies what shared resources are directly accessible to CPUs.

This affects how CPUs coordinate with each other and where they retrieve/store objects in the database.
SYSTEM ARCHITECTURE

Shared Everything

Shared Nothing

Shared Disk

Shared Memory
Each DBMS node has its own CPU, memory, and local disk. Nodes only communicate with each other via network. → Better performance & efficiency. → Harder to scale capacity. → Harder to ensure consistency.
SHARED NOTHING EXAMPLE

Node

P1 → ID: 1–150

Node

P2 → ID: 151–300

Application Server
SHARED NOTHING EXAMPLE

Catalog Meta-Data

Application Server

Node

P1→ID: 1–150

P2→ID: 151–300
SHARED NOTHING EXAMPLE

Catalog Meta-Data

Application Server

Get Id=200

Node

P1→ID: 1-150

Node

P2→ID: 151-300
SHARED NOTHING EXAMPLE

Get Id=100
Get Id=200

Application Server

Node

P1→ID: 1-150

Node

P2→ID: 151-300
SHARED NOTHING EXAMPLE

Get Id=100
Get Id=200

Get Id=200

Application Server

Catalog Data

Node

P1→ID: 1–150

P2→ID: 151–300
SHARED NOTHING EXAMPLE

Application Server

Catalog Data

Node

P1\(\rightarrow\)ID: 1-150

Get Id=100
Get Id=200

Id=200

P2\(\rightarrow\)ID: 151-300

Get Id=200

Get Id=200

Node
SHARED NOTHING EXAMPLE

Catalog Meta-Data

Application Server

Node

P1 → ID: 1-150

P2 → ID: 151-300
SHARED NOTHING EXAMPLE

Catalog Meta-Data

Application Server

Node

P1→ID:1–150

Node

P2→ID:151–300

Node
SHARED NOTHING EXAMPLE

Catalog Meta-Data

Application Server

Node

P1\(\rightarrow\)ID: 1–150

Node

P2\(\rightarrow\)ID: 151–300

Node
SHARED NOTHING EXAMPLE

Catalog Meta-Data

Application Server

Node

P1→ID: 1-100

Node

P3→ID: 101-200

Node

P2→ID: 201-300
Nodes access a single logical disk via an interconnect, but each have their own private memories.

→ Scale execution layer independently from the storage layer.
→ Nodes can still use direct attached storage as a slower/larger cache.
→ This architecture facilitates **data lakes** and **serverless** systems.
SHARED DISK EXAMPLE

Catalog Meta-Data

Application Server

Node

Node

Storage
SHARED DISK EXAMPLE

Application Server

Get Id=101

Node

Catalog Meta-Data

Node

Storage
**SHARED DISK EXAMPLE**

Application Server

Catalog Meta-Data

Node

Get Id=101

Storage
SHARED DISK EXAMPLE

Application Server

Catalog Meta-Data

Get Id=101

Node

Page ABC

Storage
SHARED DISK EXAMPLE

Catalog Meta-Data

Application Server

Node

Node

Storage
SHARED DISK EXAMPLE

Catalog Meta-Data

Application Server

Node

Node

Node

Storage
SHARED DISK EXAMPLE

Application Server

Catalog Meta-Data

Node

Node

Node

Storage

Get Id=101

Page ABC
SHARED DISK EXAMPLE

Catalog Meta-Data

Application Server

Node

Node

Node

Storage
**SHARED DISK EXAMPLE**

- **Application Server**
- **Catalog Meta-Data**
  - Update 101
- **Node**
SHARED DISK EXAMPLE

Application Server

Catalog Meta-Data

Update 101

Node

Node

Node

Page ABC

Storage
SHARED DISK EXAMPLE

Catalog Meta-Data

Update 101

Node

Node

Node

Storage

Application Server

Page ABC
**SHARED DISK EXAMPLE**

- **Catalog Meta-Data**
- **Update 101**
- **Page ABC**

![Diagram showing shared disk example](image)
Nodes access a common memory address space via a fast interconnect.

→ Each node has a global view of all the in-memory data structures.
→ Can still use local memory / disk for intermediate results.

This looks a lot like shared-everything. Nobody does this.
EARLY DISTRIBUTED DATABASE SYSTEMS

MUFFIN – UC Berkeley (1979)
SDD-1 – CCA (1979)
**System R** – IBM Research (1984)
**Gamma** – Univ. of Wisconsin (1986)
**NonStop SQL** – Tandem (1987)
DESIGN ISSUES

How does the application find data?
Where does the application send queries?
How to execute queries on distributed data?
→ Push query to data.
→ Pull data to query.
How does the DBMS ensure correctness?
How do we divide the database across resources?

Next Class
HOMOGENOUS VS. HETEROGENEOUS

Approach #1: Homogenous Nodes
→ Every node in the cluster can perform the same set of tasks (albeit on potentially different partitions of data).
→ Makes provisioning and failover "easier".

Approach #2: Heterogenous Nodes
→ Nodes are assigned specific tasks.
→ Can allow a single physical node to host multiple "virtual" node types for dedicated tasks.
DATA TRANSPARENCY

Applications should not be required to know where data is physically located in a distributed DBMS.

→ Any query that run on a single-node DBMS should produce the same result on a distributed DBMS.

In practice, developers need to be aware of the communication costs of queries to avoid excessively "expensive" data movement.
DATABASE PARTITIONING

Split database across multiple resources:
→ Disks, nodes, processors.
→ Often called "sharding" in NoSQL systems.

The DBMS executes query fragments on each partition and then combines the results to produce a single answer.

The DBMS can partition a database **physically** (shared nothing) or **logically** (shared disk).
NAÏVE TABLE PARTITIONING

Assign an entire table to a single node.
Assumes that each node has enough storage space for an entire table.

Ideal if queries never join data across tables stored on different nodes and access patterns are uniform.
NAÏVE TABLE PARTITIONING

Table 1

Table 2

Partitions

Ideal Query:

```
SELECT * FROM table1
```
NAÏVE TABLE PARTITIONING

Ideal Query:

```sql
SELECT * FROM table1
```
NAÏVE TABLE PARTITIONING

Ideal Query:

SELECT * FROM table1
NAÏVE TABLE PARTITIONING

**Table 1**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Partitions**

- **Table 1**

- **Table 2**

**Ideal Query:**

```
SELECT * FROM table1
```
VERTICAL PARTITIONING

Split a table's attributes into separate partitions.

Must store tuple information to reconstruct the original record.

```sql
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```
VERTICAL PARTITIONING

Split a table's attributes into separate partitions.
Must store tuple information to reconstruct the original record.

CREATE TABLE foo (
    attr1 INT,
    attr2 INT,
    attr3 INT,
    attr4 TEXT
);
VERTICAL PARTITIONING

Split a table's attributes into separate partitions.

Must store tuple information to reconstruct the original record.

CREATE TABLE foo (  
  attr1 INT,  
  attr2 INT,  
  attr3 INT,  
  attr4 TEXT  
);
HORIZONTAL PARTITIONING

Split a table's tuples into disjoint subsets based on some partitioning key and scheme.
→ Choose column(s) that divides the database equally in terms of size, load, or usage.

Partitioning Schemes:
→ Hashing
→ Ranges
→ Predicates
HORIZONTAL PARTITIONING

### Ideal Query:
```
SELECT * FROM table
WHERE partitionKey = ?
```
### Horizontal Partitioning

#### Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Partitioning Key</th>
<th>2022-11-29</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XYX</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
</tr>
</tbody>
</table>

#### Partitions

```
Ideal Query:
```
```
SELECT * FROM table
WHERE partitionKey = ?
```
**HORIZONTAL PARTITIONING**

**Partitioning Key**

Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Key</th>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX</td>
<td>2022-11-29</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td>2022-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td>2022-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XYX</td>
<td>2022-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2022-11-29</td>
</tr>
</tbody>
</table>

Partitions

- hash(a)%4 = P2
- hash(b)%4 = P4
- hash(c)%4 = P3
- hash(d)%4 = P2
- hash(e)%4 = P1

**Ideal Query:**

```
SELECT * FROM table
WHERE partitionKey = ?
```
**HORIZONTAL PARTITIONING**

**Partitioning Key**

<table>
<thead>
<tr>
<th>Table</th>
<th>101</th>
<th>a</th>
<th>XXX</th>
<th>2022-11-29</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>102</td>
<td>b</td>
<td>XXX</td>
<td>2022-11-28</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>c</td>
<td>XXX</td>
<td>2022-11-29</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>d</td>
<td>XXX</td>
<td>2022-11-27</td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2022-11-29</td>
</tr>
</tbody>
</table>

**Partitions**

- hash(a)%4 = P2
- hash(b)%4 = P4
- hash(c)%4 = P3
- hash(d)%4 = P2
- hash(e)%4 = P1

**Ideal Query:**

```sql
SELECT * FROM table
WHERE partitionKey = ?
```
HORIZONTAL PARTITIONING

Partitioning Key

Table

<table>
<thead>
<tr>
<th>101</th>
<th>a</th>
<th>XXX</th>
<th>2022-11-29</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td>2022-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td>2022-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XYX</td>
<td>2022-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2022-11-29</td>
</tr>
</tbody>
</table>

Partitions

- P1
- P2
- P3
- P4

Ideal Query:

```
SELECT * FROM table
WHERE partitionKey = ?
```
**Horizontal Partitioning**

**Partitioning Key**

<table>
<thead>
<tr>
<th>Table</th>
<th>101</th>
<th>a</th>
<th>XXX</th>
<th>2022-11-29</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td>2022-11-28</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td>2022-11-29</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>d</td>
<td>XXY</td>
<td>2022-11-27</td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2022-11-29</td>
</tr>
</tbody>
</table>

Ideal Query:

```
SELECT * FROM table
WHERE partitionKey = ?
```
LOGICAL PARTITIONING
LOGICAL PARTITIONING
LOGICAL PARTITIONING
LOGICAL PARTITIONING

Application Server

Get Id=1

Node

Id=1
Id=2

Node

Id=3
Id=4

Storage

Id=1
Id=2
Id=3
Id=4
LOGICAL PARTITIONING

Application Server

Node

Get Id=1

Id=1
Id=2

Id=3
Id=4

Storage

Id=1
Id=2
Id=3
Id=4

Node
LOGICAL PARTITIONING
LOGICAL PARTITIONING

Application Server

Node
Id=1
Id=2

Node
Id=3
Id=4

Storage
Id=1
Id=2
Id=3
Id=4
LOGICAL PARTITIONING

Application Server

Node

Id=1
Id=2

Get Id=3
Get Id=2

Node

Id=3
Id=4

Storage

Id=1
Id=2
Id=3
Id=4
LOGICAL PARTITIONING

Application Server

Storage

Node

Get Id=3
Get Id=2

Id=1
Id=2

Id=1
Id=2
Id=3
Id=4

Id=3
Id=4
PHYSICAL PARTITIONING
PHYSICAL PARTITIONING

Application
Server

Node
Id=1
Id=2

Node
Id=3
Id=4
PHYSICAL PARTITIONING

Application Server

Node

Get Id=1

Id=1
Id=2

Id=3
Id=4
PHYSICAL PARTITIONING

Application Server

Get Id=3

Node

Id=1
Id=2

Node

Id=3
Id=4
HORIZONTAL PARTITIONING

**Partitioning Key**

<table>
<thead>
<tr>
<th>Table</th>
<th>Partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>101 a XXX 2022-11-29</td>
<td>hash(a)%4 = P2</td>
</tr>
<tr>
<td>102 b XXY 2022-11-28</td>
<td>hash(b)%4 = P4</td>
</tr>
<tr>
<td>103 c XYZ 2022-11-29</td>
<td>hash(c)%4 = P3</td>
</tr>
<tr>
<td>104 d XYX 2022-11-27</td>
<td>hash(d)%4 = P2</td>
</tr>
<tr>
<td>105 e XYY 2022-11-29</td>
<td>hash(e)%4 = P1</td>
</tr>
</tbody>
</table>

**Ideal Query:**

```sql
SELECT * FROM table
WHERE partitionKey = ?
```
HORIZONTAL PARTITIONING

Partitioning Key

Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Key</th>
<th>Value</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX</td>
<td>2022-11-29</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td>2022-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td>2022-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XYX</td>
<td>2022-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2022-11-29</td>
</tr>
</tbody>
</table>

Partitions

- P1
- P2
- P3
- P4

Ideal Query:

```sql
SELECT * FROM table
WHERE partitionKey = ?
```
HORIZONTAL PARTITIONING

**Partitioning Key**

<table>
<thead>
<tr>
<th>Table</th>
<th>Hash Key</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX 2022-11-29</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY 2022-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ 2022-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XYX 2022-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY 2022-11-29</td>
</tr>
</tbody>
</table>

**Ideal Query:**

```
SELECT * FROM table
WHERE partitionKey = ?
```
CONSISTENT HASHING
CONSISTENT HASHING

- P1
- P2
- P3
CONSISTENT HASHING

hash(key1)

P1

P2

P3
CONSISTENT HASHING

hash(key1)
CONSISTENT HASHING

hash(key1)

hash(key2)

P1

P2

P3
CONSISTENT HASHING

hash(key1)

hash(key2)
CONSISTENT HASHING

hash(key1)

hash(key2)
CONSISTENT HASHING

hash(key1)

hash(key2)
CONSISTENT HASHING

New Partition

P1
P2
P3
P4
If hash(key) = P4

New Partition
CONSISTENT HASHING

New Partition
CONSISTENT HASHING

New Partition
CONSISTENT HASHING

Replication Factor = 3
CONSISTENT HASHING

Replication Factor = 3
CONSISTENT HASHING

Replication Factor = 3

hash(key1)
CONSISTENT HASHING

Replication Factor = 3

hash(key1)

CMU-DB
15-445/645 (Spring 2024)
CONSISTENT HASHING

Replication Factor = 3

hash(key1)

P1
P2
P3
P4
P5
P6

0.5

1.0
SINGLE-NODE VS. DISTRIBUTED

A **single-node** txn only accesses data that is contained on one partition.
→ The DBMS may not need check the behavior concurrent txns running on other nodes.

A **distributed** txn accesses data at one or more partitions.
→ Requires expensive coordination.
TRANSACTION COORDINATION

If our DBMS supports multi-operation and distributed txns, we need a way to coordinate their execution in the system.

Two different approaches:
→ **Centralized**: Global "traffic cop".
→ **Decentralized**: Nodes organize themselves.

Most distributed DBMSs use a hybrid approach where they periodically elect some node to be a temporary coordinator.
TP MONITORS

A **TP Monitor** is an example of a centralized coordinator for distributed DBMSs. Originally developed in the 1970-80s to provide txns between terminals and mainframe databases.

→ Examples: ATMs, Airline Reservations.

Standardized protocol from 1990s: **X/Open XA**
CENTRALIZED COORDINATOR

Coordinator

Partitions

P1

P2

P3

P4

Application Server
CENTRALIZED COORDINATOR

Coordinator

Partitions

Application Server

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Application Server

Lock Request

Coordinator

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Coordinator

Lock Request

Application Server

Partitions

P1
P2
P3
P4
CENTRALIZED COORDINATOR

Coordinator

Lock Request

Application Server

Partitions

P1
P2
P3
P4
CENTRALIZED COORDINATOR

Application Server

Lock Request

Coordinator

Acknowledgement

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Application Server

Coordinator

Partitions

P1
P2
P3
P4

P1
P2
P3
P4
CENTRALIZED COORDINATOR

Application Server

Coordinator

Partitions

P1
P2
P3
P4
CENTRALIZED COORDINATOR

Coordinator

Partitions

Application Server
CENTRALIZED COORDINATOR

Commit Request

Coordinator

Partitions

P1
P2
P3
P4

Application Server
CENTRALIZED COORDINATOR

Commit Request

Coordinator

Partitions

Application Server

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Commit Request

Coordinator

Partitions

Application Server
Centralized Coordinator

Application Server

Commit Request

Coordinator

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Application Server

Coordinator

Partitions

Commit Request

Safe to commit?
CENTRALIZED COORDINATOR

Coordinator

Commit Request

Application Server

Partitions

Acknowledgement

Safe to commit?
CENTRALIZED COORDINATOR

Commit Request

Coordinator

Acknowledgement

Safe to commit?

Application Server

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Application Server

Middleware

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Middleware

Application Server

Query Requests

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Application Server

Middleware

Query Requests

Partitions

P1→ID: 1-100
P2→ID: 101-200
P3→ID: 201-300
P4→ID: 301-400
CENTRALIZED COORDINATOR

Middleware

Query Requests

Application Server

Partitions

P1
ID: 1-100

P2
ID: 101-200

P3
ID: 201-300

P4
ID: 301-400
CENTRALIZED COORDINATOR

Middleware

Safe to commit?

Application Server

Partitions

P1

P2

P3

P4

P1\(\rightarrow\)ID:1\(\rightarrow\)100

P2\(\rightarrow\)ID:101\(\rightarrow\)200

P3\(\rightarrow\)ID:201\(\rightarrow\)300

P4\(\rightarrow\)ID:301\(\rightarrow\)400
DECENTRALIZED COORDINATOR

Application Server

Partitions

P1  P2

P3  P4
DECENTRALIZED COORDINATOR

Application Server

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Begin Request

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Begin Request

Leader Node

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Query Request

Leader Node

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Leader Node

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Commit Request

Leader Node

Partitions
P1
P2
P3
P4
DECENTRALIZED COORDINATOR

Application Server

Commit Request

Safe to commit?

Leader Node

Partitions

P1

P2

P3

P4
OBSERVATION

We have assumed that the nodes in our distributed systems are running the same DBMS software. But organizations often run many different DBMSs in their applications.

It would be nice if we could have a single interface for all our data.
FEDERATED DATABASES

Distributed architecture that connects disparate DBMSs into a single logical system.
→ Expose a single query interface that can access data at any location.

This is hard and nobody does it well
→ Different data models, query languages, limitations.
→ No easy way to optimize queries
→ Lots of data copying (bad).
FEDERATED DATABASE EXAMPLE

- Application Server
- Middleware
- Back-end DBMSs: MySQL, MongoDB, SUBWAY, redis
FEDERATED DATABASE EXAMPLE

Application Server

Middleware

Query Requests

Back-end DBMSs

MySQL

MongoDB

SUBWAY

redis
FEDERATED DATABASE EXAMPLE

Application Server

Middleware

Query Requests

Back-end DBMSs

MySQL

MongoDB

SUBWAY

redis
FEDERATED DATABASE EXAMPLE

Application Server

Middleware

Connectors

Back-end DBMSs

- MongoDB
- MySQL
- redis

Query Requests
DISTRIBUTED CONCURRENCY CONTROL

Need to allow multiple txns to execute simultaneously across multiple nodes.
→ Many of the same protocols from single-node DBMSs can be adapted.

This is harder because of:
→ Replication.
→ Network Communication Overhead.
→ Node Failures (Permanent + Ephemeral).
→ Clock Skew.
DISTRIBUTED 2PL

Application Server

Node 1

Node 2

Set A=2

Set B=7

NETWORK

Application Server

A=1

B=8

A=1

B=8
DISTRIBUTED 2PL

Application Server → Node 1

Set A=2

Node 1

A=1

Node 2

Set B=7

Application Server

B=8

NETWORK
DISTRIBUTED 2PL

Node 1

Set A=2
Set B=9

Node 2

Set B=7
Set A=0

Application Server

Application Server

Network

A=2

B=7
DISTRIBUTED 2PL

Application Server

Node 1

Node 2

Application Server

Set A=2
Set B=9

Set B=7
Set A=0

A=2

B=7

NETWORK
DISTRIBUTED 2PL

Waits-For Graph

Application Server

Set A=2

Set B=7

T1

T2

=7

=0

Application Server

Set A=0

Set B=9

Node 1

Node 2

NETWORK

A=2

B=7

Waits For Graph
CONCLUSION

We have barely scratched the surface on distributed database systems...

It is **hard** to get this right.
NEXT CLASS

Distributed OLTP Systems
Replication
CAP Theorem
Real-World Examples