Introduction to Distributed Databases
Homework #5: Will be released on Monday Nov 22\textsuperscript{nd}. It is due Dec 2\textsuperscript{nd} @ 11:59pm.

Project #4: Will be released today. It is due Dec 5\textsuperscript{th} @ 11:59pm.
UPCOMING DATABASE TALK

Fluree - Cloud-Native Ledger Graph Database

→ Mon Nov 15th @ 4:30pm ET
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
SYSTEM ARCHITECTURE

Shared Everything

Network

Shared Memory
SYSTEM ARCHITECTURE

- Shared Everything
- Shared Memory
- Shared Disk
SYSTEM ARCHITECTURE

Shared Everything

Shared Memory

Shared Disk

Shared Nothing

Network

Network

Network

Network
CPUs have access to common memory address space via a fast interconnect.

→ Each processor has a global view of all the in-memory data structures.
→ Each DBMS instance on a processor has to "know" about the other instances.
All CPUs can access a single logical disk directly via an interconnect, but each have their own private memories.

→ Can scale execution layer independently from the storage layer.
→ Must send messages between CPUs to learn about their current state.
All CPUs can access a single logical disk directly via an interconnect, but each have their own private memories.

→ Can scale execution layer independently from the storage layer.
→ Must send messages between CPUs to learn about their current state.
SHARED DISK EXAMPLE

Application Server

Node

Node

Storage
**SHARED DISK EXAMPLE**

Application Server -> Node

Get Id=101

Node -> Storage
SHARED DISK EXAMPLE

Application Server

Node 1

Node 2

Storage

Get Id=200

Page XYZ
SHARED DISK EXAMPLE

Application Server

Node

Storage

Node

Node
SHARED DISK EXAMPLE

Application Server

Storage

Node

Node

Node
SHARED DISK EXAMPLE

Application Server

Node

Get Id=101

Node

Page ABC

Node

Storage
SHARED DISK EXAMPLE

Application Server

Node

Node

Node

Storage
SHARED DISK EXAMPLE

Application Server

Update 101

Node

Node

Node

Storage
SHARED DISK EXAMPLE

Application Server

Node

Node

Node

Storage

Update 101

Page ABC
SHARED DISK EXAMPLE

Application Server

Node

Update 101

Node

Page ABC

Node

Storage

Page ABC
SHARED DISK EXAMPLE

Application Server

Update 101

Page ABC

Storage
Each DBMS instance has its own CPU, memory, and disk.

Nodes only communicate with each other via network.
→ Harder to scale capacity.
→ Harder to ensure consistency.
→ Better performance & efficiency.
Each DBMS instance has its own CPU, memory, and disk.

Nodes only communicate with each other via network.

→ Harder to scale capacity.
→ Harder to ensure consistency.
→ Better performance & efficiency.
SHARED NOTHING EXAMPLE

Application Server

Node
P1 → ID: 1-150

Node
P2 → ID: 151-300
**SHARED NOTHING EXAMPLE**

Application Server

```
Node
P1→ID:1-150
```

```
Node
P2→ID:151-300
```

Get Id=200
**SHAREDTOHING EXAMPLE**

Node P1→ID:1-150

Node P2→ID:151-300
Shared Nothing Example

Application Server

Node

P1→ID: 1-150

Get Id=10
Get Id=200

Node

P2→ID: 151-300
**SHARE NOTH EXAMPLE**

**Application Server**

1. Get Id=10
2. Get Id=200

**Node**

1. P1→ID:1-150
2. Get Id=200
3. P2→ID:151-300
**SHARED NOTHING EXAMPLE**

Node P1: ID: 1-150

Node P2: ID: 151-300
SHARED NOTHING EXAMPLE

Node

P1→ID: 1-150

Node

P2→ID: 151-300

Node

Application Server
**SHARED NOTHING EXAMPLE**

![Diagram of shared nothing example]

Node P1 → ID: 1-150

Node P2 → ID: 151-300

Application Server
SHARED NOTHING EXAMPLE

Node

P1→ID: 1-100

Node

P2→ID: 201-300

Node

P3→ID: 101-200

Application Server
EARLY DISTRIBUTED DATABASE SYSTEMS

MUFFIN – UC Berkeley (1979)
SDD-1 – CCA (1979)
Gamma – Univ. of Wisconsin (1986)
NonStop SQL – Tandem (1987)
DESIGN ISSUES

How does the application find data?

How to execute queries on distributed data?

→ Push query to data.
→ Pull data to query.

How does the DBMS ensure correctness?
HOMOGENOUS VS. HETEROGENOUS

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.
MONGODB HETEROGENEOUS ARCHITECTURE

Application Server

Router (mongos)
Router (mongos)

Config Server (mongod)

Shards (mongod)
P1
P2
P3
P4
MONGODB HETEROGENEOUS ARCHITECTURE

- **Application Server**
  - Get Id=101

- **Router (mongos)**
- **Router (mongos)**
- **Config Server (mongod)**

- **Shards (mongod)**
  - P1
  - P2
  - P3
  - P4
MONGODB HETEROGENOUS ARCHITECTURE

Application Server

Router (mongos)

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

Config Server (mongod)

Router (mongos)

⋮

Shards (mongod)

⋮

Get Id=101

Application Server

[Diagram showing the architecture with application server, router, config server, and shards]

CMU-DB 15-445/645 (Fall 2021)
MONGODB HETEROGENOUS ARCHITECTURE

Application Server

Config Server (mongod)

Router (mongos)

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

Router (mongos)

P1
P2
P3
P4

Config Server (mongod)

Shards (mongod)

Get Id=101
MONGODB HETEROGENEOUS ARCHITECTURE

Router (mongos)

Shards (mongod)

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

Get Id=101

Application Server

Config Server (mongod)
DATA TRANSPARENCY

Users should not be required to know where data is physically located, how tables are partitioned or replicated.

A query that works on a single-node DBMS should work the same on a distributed DBMS.
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.
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

Table1

Table2

Partitions

Ideal Query:

SELECT * FROM table
NAÏVE TABLE PARTITIONING

Table 1

Table 2

Partitions

Ideal Query:

```
SELECT * FROM table
```
NAÏVE TABLE PARTITIONING

Table1

Table2

Partitions

Ideal Query:

```
SELECT * FROM table
```
NAÏVE TABLE PARTITIONING

Table1

Table2

Partitions

Ideal Query:

```
SELECT * FROM table
```
HORIZONTAL PARTITIONING

Split a table's tuples into disjoint subsets.
→ Choose column(s) that divides the database equally in terms of size, load, or usage.
→ Hash Partitioning, Range Partitioning

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

Table1

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX</td>
<td>2019-11-29</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td>2019-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td>2019-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XYX</td>
<td>2019-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2019-11-29</td>
</tr>
</tbody>
</table>

Partitions

Ideal Query:

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

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

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

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

**Partitioning Key**

Table 1:

<table>
<thead>
<tr>
<th>ID</th>
<th>Partition Key</th>
<th>Value</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX</td>
<td>2019-11-29</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td>2019-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td>2019-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XXY</td>
<td>2019-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2019-11-29</td>
</tr>
</tbody>
</table>

**Partitions**

- **P1**
- **P2**
- **P3**
- **P4**

**Ideal Query:**

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

### Table 1

<table>
<thead>
<tr>
<th>ID</th>
<th>Partitioning Key</th>
<th>Value</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX</td>
<td>2019-11-29</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td>2019-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td>2019-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XYX</td>
<td>2019-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2019-11-29</td>
</tr>
</tbody>
</table>

### Partitions

- **P1**
- **P2**
- **P3**
- **P4**

### Ideal Query:

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

**Partitioning Key**

Table1

<table>
<thead>
<tr>
<th>ID</th>
<th>Value</th>
<th>String</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX</td>
<td>2019-11-29</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td>2019-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td>2019-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XYX</td>
<td>2019-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td>2019-11-29</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

### Table 1

<table>
<thead>
<tr>
<th>ID</th>
<th>Key</th>
<th>Value1</th>
<th>Value2</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>a</td>
<td>XXX</td>
<td></td>
<td>2019-11-29</td>
</tr>
<tr>
<td>102</td>
<td>b</td>
<td>XXY</td>
<td></td>
<td>2019-11-28</td>
</tr>
<tr>
<td>103</td>
<td>c</td>
<td>XYZ</td>
<td></td>
<td>2019-11-29</td>
</tr>
<tr>
<td>104</td>
<td>d</td>
<td>XXY</td>
<td></td>
<td>2019-11-27</td>
</tr>
<tr>
<td>105</td>
<td>e</td>
<td>XYY</td>
<td></td>
<td>2019-11-29</td>
</tr>
</tbody>
</table>

**Partitioning Key**
- hash(a) % 5 = P4
- hash(b) % 5 = P3
- hash(c) % 5 = P5
- hash(d) % 5 = P1
- hash(e) % 5 = P3

**Ideal Query:**

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

hash(key1)

P1, P2, P3
CONSISTENT HASHING

hash(key1)

P1

P2

P3
CONSISTENT HASHING

hash(key1)
CONSISTENT HASHING

hash(key1)

hash(key2)
CONSISTENT HASHING

hash(key1)

hash(key2)
CONSISTENT HASHING

hash(key1)

hash(key2)

P1

P2

P3

0.5
CONSISTENT HASHING

![Diagram of consistent hashing]

- $hash(key1)$
- $hash(key2)$
CONSISTENT HASHING
CONSISTENT HASHING

P1
P2
P3
P4
CONSISTENT HASHING

If hash(key) = P4
CONSISTENT HASHING
CONSISTENT HASHING
CONSISTENT HASHING

Replication Factor = 3
CONSISTENT HASHING

Replication Factor = 3
CONSISTENT HASHING

Replication Factor = 3
CONSISTENT HASHING

Replication Factor = 3
CONSISTENT HASHING

Replication Factor = 3

P1
P2
P3
P4
P5
P6
CONSISTENT HASHING

hash(key1)

Replication Factor = 3
CONSISTENT HASHING

Replication Factor = 3

hash(key1)
CONSISTENT HASHING

hash(key1)

Replication Factor = 3
LOGICAL PARTITIONING

Application Server

Node

Node

Storage

Node

Storage
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

Node

Storage

Get Id=1

Id=1
Id=2

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

Id=3
Id=4
LOGICAL PARTITIONING

Application Server

Node

Get Id=1

Id=1
Id=2

Id=3
Id=4

Node

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

Storage
LOGICAL PARTITIONING

Application Server

Node

Get Id=3

Node

Id=3

Id=4

Storage

Id=1

Id=2

Id=3

Id=4
PHYSICAL PARTITIONING

Application Server

Node

Node
PHYSICAL PARTITIONING

Node

Application Server

Node

Id=1
Id=2

Id=3
Id=4
PHYSICAL PARTITIONING

Application Server

Get Id=1

Node

Id=1
Id=2

Node

Id=3
Id=4
Application Server

Node
Id=1
Id=2

Get Id=3

Node
Id=3
Id=4
SINGLE-NODE VS. DISTRIBUTED

A **single-node** txn only accesses data that is contained on one partition.
→ The DBMS does not need coordinate 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.
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.

Many DBMSs now support the same functionality internally.
CENTRALIZED COORDINATOR

Coordinator

Partitions

P1

P2

P3

P4

Application Server
CENTRALIZED COORDINATOR

Coordinator

Partitions

P1

P2

P3

P4

Application Server
CENTRALIZED COORDINATOR

Coordinator

Lock Request

Application Server

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Application Server

Lock Request

Coordinator

Partitions

P1
P2
P3
P4
CENTRALIZED COORDINATOR

Application Server

Coordinator

Lock Request

Partitions

P1
P2
P3
P4
CENTRALIZED COORDINATOR

Coordinator

Lock Request

Acknowledgement

Application Server

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Coordinator

Partitions

Application Server

P1
P2
P3
P4

P1
P2
P3
P4
Centralized Coordinator

Coordinator

Partitions

Application Server

P1
P2
P3
P4

P1
P2
P3
P4
CENTRALIZED COORDINATOR

Commit Request

Application Server

Coordinator

Partitions

P1
P2
P3
P4
CENTRALIZED COORDINATOR

Commit Request

Application Server

Safe to commit?

Coordinator

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Coordinator

Commit Request

Application Server

Acknowledgement

Safe to commit?

Partitions

P1
P2
P3
P4
Centralized Coordinator

Commit Request

Coordinator

Acknowledgement

Safe to commit?

Partitions

Application Server
Centralized Coordinator

Middleware

Application Server

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Application Server

Middleware

Partitions

P1

P2

P3

P4

Query Requests
CENTRALIZED COORDINATOR

Middleware

Query Requests

Application Server

Partitions

P1

P2

P3

P4

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

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

Application Server → Middleware

Commit Request

Partitions:
- P1
- P2
- P3
- P4

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

Application Server

Middleware

Commit Request

Safe to commit?

Partitions

- P1
- P2
- P3
- P4

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

Application Server

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Begin Request

Master Node

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server → Query Request → Master Node → Partitions

- P1
- P2
- P3
- P4
DECENTRALIZED COORDINATOR

Application Server

Master Node

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Commit Request

Master Node

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Commit Request

Safe to commit?

Master Node

Partitions

P1

P2

P3

P4
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.
→ Clock Skew.
DISTRIBUTED 2PL

Application Server

A=1

Node 1

NETWORK

Node 2

B=8

Application Server
DISTRIBUTED 2PL

Application Server

Set A=2

A=1

Node 1

NETWORK

Node 2

Set B=7

B=8

Application Server
DISTRIBUTED 2PL

Application Server

Set A=2

A=1

Node 1

NETWORK

Set B=7

B=8

Node 2

Application Server
DISTRIBUTED 2PL

Application Server → Node 1: Set A=2

Node 1: A=2

Application Server ← Node 2: Set B=7

Node 2: B=7

NETWORK
DISTRIBUTED 2PL

Application Server  

Set A=2
Set B=9

Node 1

A=2

Network

Node 2

B=7

Set A=0
Set B=7

Application Server
DISTRIBUTED 2PL

Application Server

Set A=2
Set B=9

Node 1

A=2

Node 2

B=7

Set A=0
Set B=7

Application Server

NETWORK
DISTRIBUTED 2PL

**Waits-For Graph**

**Application Server**

**Set A = 2**

**Application Server**

**Set B = 7**

**Application Server**

**Set A = 0**

**Application Server**

**Set B = 9**

**Application Server**

**T_1**

**T_2**

**Waits**

**For**

**Graph**

**Node 1**

**A = 2**

**Node 2**

**B = 7**

**NETWORK**
CONCLUSION

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

It is **hard** to get this right.
Distributed OLTP Systems
Replication
CAP Theorem
Real-World Examples