ADMINISTRIVIA

Homework #5: Monday Dec 3\textsuperscript{rd} @ 11:59pm

Project #4: Monday Dec 10\textsuperscript{th} @ 11:59pm

Extra Credit: Wednesday Dec 10\textsuperscript{th} @ 11:59pm

Final Exam: Monday Dec 9\textsuperscript{th} @ 5:30pm
ADMINISTRIVIA

Monday Dec 2\textsuperscript{th} – Oracle Lecture
\rightarrow Shasank Chavan (VP In-Memory Databases)

Wednesday Dec 4\textsuperscript{th} – Potpourri + Review
\rightarrow Vote for what system you want me to talk about.
\rightarrow https://cmudb.io/f19-systems

Sunday Nov 24\textsuperscript{th} – Extra Credit Check
\rightarrow Submit your extra credit assignment early to get feedback from me.
UPCOMING DATABASE EVENTS

Oracle Research Talk
→ Tuesday December 4th @ 12:00pm
→ CIC 4th Floor
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 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 Memory
- Shared Disk
- Shared Nothing

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.
SHARED DISK EXAMPLE

Application Server

Node

Get Id=101

Node

Get Id=101

Get Id=200

Node

Page ABC

Page ABC

Page XYZ

Storage

Get Id=101
Each DBMS instance has its own CPU, memory, and disk. Nodes only communicate with each other via network. → Hard to increase capacity. → Hard to ensure consistency. → Better performance & efficiency.
SHARED NOTHING EXAMPLE

Node P1→ID: 1-100
Node P2→ID: 151-300
Node P3→ID: 101-200
Node P2→ID: 201-300

Application Server

Get Id=10
Get Id=200

Get Id=200

Get Id=200
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. 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.
**MongoDB Heterogeneous Architecture**

- **Router (mongos)**
  - P1 → ID: 1-100
  - P2 → ID: 101-200
  - P3 → ID: 201-300
  - P4 → ID: 301-400

- **Config Server (mongod)**

- **Shards (mongod)**
  - P1
  - P2
  - P3
  - P4

Application Server

Get Id=101
DATA TRANSPARENCY

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

A SQL 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.
→ Sometimes called "sharding"

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

Each node stores one and only table. Assumes that each node has enough storage space for a 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 physical (shared nothing) or logically (shared disk).
SELECT * FROM table
WHERE partitionKey = ?
CONSISTENT HASHING

Replication Factor = 3

If \( \text{hash(key)} = D \)

\( \text{hash(key1)} \)

\( \text{hash(key2)} \)
LOGICAL PARTITIONING

Application Server

Node

Get Id=1

Get Id=3

Node

Storage

Id=1
Id=2

Id=3
Id=4

Id=1
Id=2
Id=3
Id=4
PHYSICAL PARTITIONING

Application Server

Get Id=1

Get Id=3

Node

Id=1
Id=2

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.
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.
Example of a centralized coordinator.
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

Commit Request

Application Server

Acknowledgement

Safe to commit?

Partitions

P1

P2

P3

P4
Centralized Coordinator

Middleware

Commit Request

Safe to commit?

Application Server

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

Commit Request

Query Request

commit?
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

Waits-For Graph

Node 1
A=2
NETWORK

Node 2
B=7

Application Server
Set A=2

Application Server
Set B=7

Application Server
Set A=0

Application Server
Set B=8

Waits For
Graph

T1

T2

=0

T1

T2

=7
I have barely scratched the surface on distributed database systems…
It is **hard** to get right.

More info (and humiliation):
→ [Kyle Kingsbury's Jepsen Project](#)
NEXT CLASS

Distributed OLTP Systems
Replication
CAP Theorem
Real-World Examples