Intro to Database Systems (15-445/645)

21

Introduction to Distributed Databases

Carnegie Mellon University

SPRING 2023

Charlie Garrod
ADMINISTRIVIA

Project 3 due last night

Homework 5 released today
→ Due Friday, April 21\textsuperscript{st} at 11:59 p.m.

Project 4 released today
→ Due Friday, April 28\textsuperscript{th} at 11:59 p.m.

Interested in TAing this course?
→ https://forms.gle/AvjfUtSaWtrNiJMXA

Final exam Monday, May 1\textsuperscript{st}, 8:30 – 11:30 a.m.
LAST TIME: RECOVERY

Fuzzy checkpoints
ARIES
RECALL: 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
SYSTEM ARCHITECTURE

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
- Distributed Shared Memory
- Shared Disk
- Shared Nothing
DISTRIBUTED SHARED MEMORY

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 must "know" about the other instances.
SHARED DISK

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

Get Id=101

Get Id=102

Node

Page ABC

Page XYZ

Node

Page ABC

Storage
**SHARED NOTHING**

Each DBMS instance has its own CPU, memory, and local disk. Nodes only communicate with each other via network.

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

Get Id=100
Get Id=200

Get Id=200

Id=200

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

Application Server

Node

Node

Node
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?
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.
MONGODB HETEROGENOUS ARCHITECTURE

Router (mongos)

Shards (mongod)

P1
P2
P3
P4

Config Server (mongod)

Application Server

Get Id=101

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

Ideal Query:
SELECT * FROM table
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 = ?
```
LOGICAL PARTITIONING

Application Server

Node

Node

Node

Node

Storage

Get Id=1

Get Id=2

Get Id=3

Get Id=4

Get Id=3

Get Id=2

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

Id=3
Id=4
# Horizontal Partitioning

## Table1

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

**Partitioning Key**

- $\text{hash}(a) \% 4 = P_2$
- $\text{hash}(b) \% 4 = P_3$
- $\text{hash}(c) \% 5 = P_5$
- $\text{hash}(d) \% 5 = P_1$
- $\text{hash}(e) \% 5 = P_3$

**Ideal Query:**

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

Replication Factor = 3

If hash(key) = P4

hash(key1)

hash(key2)
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.
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

Application Server

Coordinator

Partitions

Commit Request

Acknowledgement

Safe to commit?
CENTRALIZED COORDINATOR

Application Server

Middleware

Commit Request

Safe to commit?

Partitions

P1

P2

P3

P4

Commit Request

P1→ID: 1-100

P2→ID: 101-200

P3→ID: 201-300

P4→ID: 301-400

Safe to commit?
DECENTRALIZED COORDINATOR

Application Server

Commit Request

Leader Node

Partition P1

Partition P2

Partition P3

Partition P4

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

Application Server

Set A=2

Node 1

Set B=9

Waits

For

Graph

T1

T2

=7

=0

Application Server

Set B=7

Node 2

Set A=0

NETWORK
CONCLUSION

I have barely scratched the surface on distributed database systems…

It is hard to get this right.
PROJECT #4 – CONCURRENCY CONTROL

You will add support for concurrent transactions using two-phase locking in BusTub!
→ Deadlock Detection
→ Hierarchical Locking (Table, Tuple)
→ Multiple Isolation Levels
→ Aborts/Rollbacks

You do not need to worry about logging txns to disk.

https://15445.courses.cs.cmu.edu/spring2023/project4/
PROJECT #4 – TASKS

**Lock Manager**
→ Maintain internal lock table and queues.
→ Track the growing/shrinking phases of txns.
→ Notify waiting txns when their locks are available.

**Deadlock Detector:**
→ Build the waits-for graph and deterministically identify whattxn to kill to break deadlocks

**Execution Engine**
→ Modify Project #3 executors to support txn requests.
PROJECT #4 - LEADERBOARD

We have designed the Terrier benchmark to measure who has the fastest BusTub implementation.

Tasks:
→ UpdateExecutor
→ Predicate Pushdown
THINGS TO NOTE

Do not change any file other than the ones that you submit to Gradescope.

Make sure you pull in the latest changes from the BusTub main branch.

Post your questions on Piazza or come to TA office hours.

Compare against our solution in your browser.
NEXT CLASS

Distributed OLTP Systems
Replication
CAP Theorem
Real-World Examples