Distributed OLTP Systems
ADMINISTRIVIA

Homework #6: TODAY @ 11:59pm

Project #4: Wednesday December 6th @ 11:59am
ADMINISTRIVIA

Monday Dec 4th – NuoDB
→ Barry Morris (Co-Founder, Exec. Chairman)

Wednesday Dec 6th – Potpourri + Final Review
→ Vote for what system you want me to talk about.
→ http://cmudb.io/f17-systems
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 & query execution in distributed environments.
→ Optimization & Planning
→ Concurrency Control
→ Logging & Recovery
OLTP VS. OLAP

On-line Transaction Processing (OLTP):
→ Short-lived txns.
→ Small footprint.
→ Repetitive operations.

On-line Analytical Processing (OLAP):
→ Long running queries.
→ Complex joins.
→ Exploratory queries.
WORKLOAD CHARACTERIZATION

Operation Complexity

Complex

Simple

OLAP

HTAP

OLTP

Workload Focus

Writes

Reads

Source: Michael Stonebraker
TODAY'S AGENDA

System Architectures
Design Issues
Distributed Concurrency Control
Replication
CAP Theorem
SYSTEM ARCHITECTURES

- Shared Everything
- Shared Memory
- Shared Disk
- Shared Nothing
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.
→ Have to send messages between CPUs to learn about their current state.
SHARED DISK EXAMPLE
SHARED DISK EXAMPLE

Application Server

Node

Node

Storage
SHARED DISK EXAMPLE

Application Server

Node

Node

Node

Storage
SHARED DISK EXAMPLE
SHARED DISK EXAMPLE
Each DBMS instance has its own CPU, memory, and disk.
Nodes only communicate with each other via network.
→ Easy to increase capacity.
→ Hard to ensure consistency.
SHARED NOTHING EXAMPLE

Application Server

Get Id=200

Node

P1→ID: 1-150

Node

P2→ID: 151-300
Get Id=10 & Id=200

Get Id=200

P1→ID: 1-150

P2→ID: 151-300
SHARED NOTHING EXAMPLE
SHARE NOTHING EXAMPLE

Application Server

Node

P1→ID: 1-100

Node

P3→ID: 101-200

Node

P2→ID: 201-300
EARLY DISTRIBUTED DATABASE SYSTEMS

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

How do we store data across nodes?
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?
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.
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.
→ Each tuple contains all of its columns.

Three main approaches:
→ Round-robin Partitioning.
→ Hash Partitioning.
→ Range Partitioning.
CREATE TABLE WAREHOUSE (
  w_id INT PRIMARY KEY,
  w_name VARCHAR UNIQUE,
  ...
);

CREATE TABLE DISTRICT (
  d_id INT,
  d_w_id INT REFERENCES WAREHOUSE (w_id),
  ...
  PRIMARY KEY (d_w_id, d_id)
);
DATABASE PARTITIONING

Schema

WAREHOUSE
  ↓
DISTRICT
  ↓
CUSTOMER
  ↓
ORDERS
  ↓
ORDER_ITEM
  ↓
ITEM

Schema Tree

WAREHOUSE
  ↓
DISTRICT
  ↓
CUSTOMER
  ↓
ORDERS
  ↓
ORDER_ITEM
  ↓
ITEM
    Replicated
DATABASE PARTITIONING

WAREHOUSE
  ↓
DISTRICT  STOCK
  ↓
CUSTOMER  ORDERS
  ↓
ORDER_ITEM

ITEM Replicated

Partitions
DATABASE PARTITIONING

Partitions

![Diagram showing database partitioning with replicated item]

CMU 15-445/645 (Fall 2017)
21
DATABASE PARTITIONING

WAREHOUSE

DISTRICT

STOCK

CUSTOMER

ORDERS

ORDER_ITEM

ITEM

Replicated

Partitions

P1

ITEM

P2

ITEM

P3

ITEM

P4

ITEM
Get W_ID=1

Application Server

Partitions

P1
ITEM

P2
ITEM

P3
ITEM

P4
ITEM
SIMPLE EXAMPLE

Application Server

Commit

Partitions

- P1
- P2
- P3
- P4
**SINGLE-NODE VS. DISTRIBUTED TRANSACTIONS**

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.
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

Application Server

Coordinator

Lock Request

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Coordinator

Lock Request

Application Server

Partitions

P1

P2

P3

P4
CENTRALIZED COORDINATOR

Coordinator

Lock Request

Acknowledgement

Partitions

Application Server
Centralized Coordinator

Commit Request

Coordinator

Partitions

Application Server
**Centralized Coordinator**

- **Application Server**
  - Commit Request
  - Safe to commit?

- **Coordinator**
  - Acknowledgement

- **Partitions**
  - P1
  - P2
  - P3
  - P4
CEN TRA LIZED COORDIN AT OR

Query Requests

Application Server

Middleware

Partitions

P1

P2

P3

P4

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

Application Server

Query Requests

Middleware

Partitions

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

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Application Server

Middleware

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?

Partition P1

Partition P2

Partition P3

Partition P4

Commit Request

Safe to commit?
DECENTRALIZED COORDINATOR

Application Server

Partitions
P1
P2
P3
P4
DECENTRALIZED COORDINATOR

Application Server

Partitions

P1

P2

P3

P4

Begin Request
DECENTRALIZED COORDINATOR

Application Server

Query Request

Partitions

P1

P2

P3

P4
DECENTRALIZED COORDINATOR

Application Server

Commit Request

Safe to commit?

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

Set A=2

A=2

A=2

Set B=7

B=7

Application Server

Node 1

Node 2

NETWORK
DISTRIBUTED 2PL

Application Server

Set A=2
Set B=9

Node 1

A=2

Node 2

B=7

Application Server

Set B=7
Set A=0

NETWORK

CMU 15-445/645 (Fall 2017)
DISTRIBUTED 2PL

Application Server

Set A = 2

Waits-For Graph

T₁ -> T₂

T₂ -> T₁

Application Server

Set B = 7

Node 1

A = 2

Node 2

B = 7

NETWORK
OBSERVATION

We have not discussed how to ensure that all nodes agree to commit a txn and then to make sure it does commit if we decide that it should.
→ What happens if a node fails?
→ What happens if our messages show up late?
ATOMIC COMMIT PROTOCOL

When a multi-nodetxn finishes, the DBMS needs to ask all of the nodes involved whether it is safe to commit.
→ All nodes must agree on the outcome

Examples:
→ Two-Phase Commit
→ Three-Phase Commit (not used)
→ Paxos
→ Raft
→ ZAB (Apache Zookeeper)
TWO-PHASE COMMIT (SUCCESS)

Application Server

Coordinator

Node 1

Commit Request

Node 2

Node 3

Participant

Participant

CMU 15-445/645 (Fall 2017)
TWO-PHASE COMMIT (SUCCESS)

Application Server

Coordinator

Node 1

Phase 1: Prepare

Commit Request

Node 2

Node 3

Participant

Participant
TWO-PHASE COMMIT (SUCCESS)

Application Server

Commit Request

Phase 1: Prepare

Node 1

OK

Node 2

OK

Node 3
TWO-PHASE COMMIT (SUCCESS)

Application Server

Commit Request

Phase1: Prepare

Node 1

Phase2: Commit

Node 2

OK

Node 3

Participant

Participant

Participant
TWO-PHASE COMMIT (SUCCESS)

- Application Server
- Coordinator
- Node 1
- Node 2
- Node 3

Success!
TWO-PHASE COMMIT (ABORT)

Application Server

Coordinator

Node 1

Commit Request

Node 2

Node 3

Participant

Participant

Participant
TWO-PHASE COMMIT (ABORT)

Application Server

Commit Request

Phase1: Prepare

Coordinator

Node 1

Node 2

Node 3

Participant

Participant

Participant
TWO-PHASE COMMIT (ABORT)

Application Server

Phase 1: Prepare

Commit Request

Node 1

Coordinator

ABORT!

Node 2

Participant

Node 3

Participant
TWO-PHASE COMMIT (ABORT)

Application Server

Coordinator

Node 1

Aborted

Node 2

Node 3

ABORT!

Participant

Participant

Participant
TWO-PHASE COMMIT (ABORT)

Application Server

Aborted

Coordinator

Phase2: Abort

Node 1

ABORT!

Node 2

Participant

Participant

Node 3
TWO-PHASE COMMIT (ABORT)

Application Server

Aborted

Phase2: Abort

OK

Node 2

Participant

Node 1

Coordinator

Node 3

Participant

ABORT!

OK
2PC OPTIMIZATIONS

Early Prepare Voting
→ If you send a query to a remote node that you know will be the last one you execute there, then that node will also return their vote for the prepare phase with the query result.

Early Acknowledgement After Prepare
→ If all nodes vote to commit a txn, the coordinator can send the client an acknowledgement that their txn was successful before the commit phase finishes.
EARLY ACKNOWLEDGEMENT

Application Server

Coordinator

Node 1

Commit Request

Node 2

Node 3

Participant

Participant
EARLY ACKNOWLEDGEMENT

Commit Request

Application Server

Phase 1: Prepare

Coordinator

Node 1

Node 2

Node 3

Participant

Participant

Participant
EARLY ACKNOWLEDGEMENT

Commit Request

Phase1: Prepare

Application Server

Node 1

Coordinator

Node 2

OK

OK

Node 3

Participant

Participant

Participant
**EARLY ACKNOWLEDGEMENT**

Application Server

![Success!](image)

**Phase 1: Prepare**

Node 1

Coordinator

Node 2

Participant

OK

Node 3

Participant

OK

CMU 15-445/645 (Fall 2017)
EARLY ACKNOWLEDGEMENT

Phase 1: Prepare
Phase 2: Commit

Success!

Node 1
Coordinator

Node 2
Participant

Node 3
Participant
TWO-PHASE COMMIT

Each node has to record the outcome of each phase in a stable storage log.

**What happens if coordinator crashes?**
→ Participants have to decide what to do.

**What happens if participant crashes?**
→ Coordinator assumes that it responded with an abort if it hasn't sent an acknowledgement yet.

The nodes have to block until they can figure out the correct action to take.
PAXOS

Consensus protocol where a coordinator proposes an outcome (e.g., commit or abort) and then the participants vote on whether that outcome should succeed.

Does not block if a majority of participants are available and has provably minimal message delays in the best case.

→ First correct protocol that was provably resilient in the face asynchronous networks
2PC VS. PAXOS

Two-Phase Commit
→ Blocks if coordinator fails after the prepare message is sent, until coordinator recovers.

Paxos
→ Non-blocking as long as a majority participants are alive, provided there is a sufficiently long period without further failures.
REPLICATION

The DBMS can replicate data across redundant nodes to increase availability.

System Configurations:
→ Master/Slave (aka Leader-Follower)
→ Multi-Master (aka Multi-Home)

Propagation Protocols:
→ Physical Logging
→ Logical Logging
REPLICATION CONFIGURATIONS

Master-Slave

- Writes
- Reads

Master

| P1
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Replicas

| P1
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Multi-Master

- Writes
- Reads

Node 1

| P1
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Node 2

| P1
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
REPLICATION PROPAGATION

Synchronous
→ The master sends updates to replicas and then waits for them to acknowledge that they fully applied (i.e., logged) the changes.

Asynchronous
→ The master immediately returns the acknowledgement to the client without waiting for replicas to apply the changes.

Semi-Synchronous
→ Replicas immediately send acknowledgements without logging them.
Recovery

What do we do if a node crashes in CA/CP DBMS?
If node is replicated, use Paxos to elect a new primary.
→ If node is last replica, halt the DBMS.
Node can recover from checkpoints + logs and then catch up with primary.
CAP THEOREM

Proposed by Eric Brewer that it is impossible for a distributed system to always be:

→ Consistent
→ Always Available
→ Network Partition Tolerant

Proved in 2002.
Linearizability

All up nodes can satisfy all requests.

Impossible

Still operate correctly despite message loss.

CAP THEOREM

Consistency
Availability
Partition Tolerant
CAP – CONSISTENCY

Application Server → Master

Set A=2

A=2
B=8

NETWORK

Replica → Application Server

A=1
B=8
Set A=2

Application Server

A=2
B=8

Master

Network

A=2
B=8

Replica

Application Server

CAP – Consistency
CAP – CONSISTENCY

Application Server

Set A=2
ACK

Master

A=2
B=8

Replica

A=2
B=8

Application Server

NETWORK
CAP – CONSISTENCY

Application Server → Set A=2 → ACK → Application Server

Master: A=2, B=8

Replica: A=2, B=8

NETWORK
CAP – CONSISTENCY

Application Server

Set A=2
ACK

A=2
B=8

NETWORK

Master

Replica

A=2
B=8

Application Server

Read A
A=2

A=2

If master says the txn committed, then it should be immediately visible on replicas.
CAP – AVAILABILITY
CAP – AVAILABILITY

Application Server

Read B

A=1
B=8

Master

NETWORK

Replica

Application Server

A=1
B=8
CAP – AVAILABILITY

Application Server

Read B
B=8

Master

A=1
B=8

NETWORK

Replica

Application Server
CAP – AVAILABILITY

Application Server

Network

Master

A=1
B=8

X

Read A

Application Server

Replica
CAP – AVAILABILITY

Application Server

A=1

B=8

Master

NETWORK

Replica

A=1

Application Server

Read A

CMU 15-445/645 (Fall 2017)
47
CAP – PARTITION TOLERANCE

Application Server

A=1
B=8

Master

A=1
B=8

Network

Replica

Application Server
CAP – PARTITION TOLERANCE

Application Server

A=1
B=8

Master

NETWORK

Master

A=1
B=8

Application Server
CAP – PARTITION TOLERANCE

Application Server

Set A=2

Master

A=2
B=8

Set A=3

Master

A=3
B=8

Application Server

NETWORK
Set $A=2$

ACK

Set $A=3$

ACK

Application Server

Application Server

Master

Master

Network

CMU 15-445/645 (Fall 2017)
48
CAP FOR OLTO DBMSs

How a DBMS handles failures determines which elements of the CAP theorem they support.

Traditional/NewSQL DBMSs
→ Stop allowing updates until a majority of nodes are reconnected.

NoSQL DBMSs
→ Provide mechanisms to resolve conflicts after nodes are reconnected.
CONCLUSION

I have barely scratched the surface on distributed txn processing...

It is **really** hard to get right.

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

Distributed OLAP Systems