ADMINISTRIVIA

Project #4: Tuesday Dec 10th @ 11:59pm

Extra Credit: Tuesday Dec 10th @ 11:59pm

Final Exam: Monday Dec 9th @ 5:30pm
Who: You
What: http://cmudb.io/f19-final
When: Monday Dec 9th @ 5:30pm
Where: Porter Hall 100
Why: https://youtu.be/6yOH_FjeSAQ
What to bring:
→ CMU ID
→ One page of handwritten notes (double-sided)
→ Extra Credit Coupon

Optional:
→ Spare change of clothes
→ Food

What not to bring:
→ Your roommate
COURSE EVALS

Your feedback is strongly needed:
→ https://cmu.smartevals.com

Things that we want feedback on:
→ Homework Assignments
→ Projects
→ Reading Materials
→ Lectures
OFFICE HOURS

Andy's hours:
→ Friday Dec 6th @ 3:30-4:30pm
→ Monday Dec 9th @ 1:30-2:30pm

All TAs will have their regular office hours up to and including Saturday Dec 14th
STUFF BEFORE MID-TERM

SQL
Buffer Pool Management
Hash Tables
B+Trees
Storage Models
Inter-Query Parallelism
TRANSACTIONS

ACID
Conflict Serializability:
→ How to check?
→ How to ensure?

View Serializability
Recoverable Schedules
Isolation Levels / Anomalies
TRANSACTIONS

Two-Phase Locking
→ Rigorous vs. Non-Rigorous
→ Deadlock Detection & Prevention

Multiple Granularity Locking
→ Intention Locks
TRANSACTIONS

Timestamp Ordering Concurrency Control
→ Thomas Write Rule

Optimistic Concurrency Control
→ Read Phase
→ Validation Phase
→ Write Phase

Multi-Version Concurrency Control
→ Version Storage / Ordering
→ Garbage Collection
Buffer Pool Policies:
→ STEAL vs. NO-STEAL
→ FORCE vs. NO-FORCE

Write-Ahead Logging

Logging Schemes

Checkpoints

ARIES Recovery
→ Log Sequence Numbers
→ CLRs
DISTRIBUTED DATABASES

System Architectures
Replication
Partitioning Schemes
Two-Phase Commit
<table>
<thead>
<tr>
<th></th>
<th>2018</th>
<th>2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cockroach Labs</td>
<td>26</td>
<td>18</td>
</tr>
<tr>
<td>Google Spanner</td>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>MongoDB</td>
<td>24</td>
<td>18</td>
</tr>
<tr>
<td>Amazon Aurora</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>Redis</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>Cassandra</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>12</td>
<td>17</td>
</tr>
<tr>
<td>Hive</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>Scuba</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>MySQL</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Internal DBMS designed for real-time data analysis of performance monitoring data.
→ Columnar Storage Model
→ Distributed / Shared-Nothing
→ Tiered-Storage
→ No Joins or Global Sorting
→ Heterogeneous Hierarchical Distributed Architecture

Designed for low-latency ingestion and queries.
Redundant deployments with lossy fault-tolerance.
FACEBOOK LOG PIPELINE

Application Servers

Structured Debug Logs

Streaming Platform

Record Batcher

Record Batcher

Record Batcher

Combined Logs Per Category

Execution Layer

Aggregator

Leaf Node

SQL Queries

Validation Service

Columnar Batch Data

Source: Stavros Harizopoulos
SCUBA ARCHITECTURE

Leaf Nodes:
→ Store columnar data on local SSDs.
→ Leaf nodes may or may not contain data needed for a query.
→ No indexes. All scanning is done on time ranges.

Aggregator Nodes:
→ Dispatch plan fragments to all its children leaf nodes.
→ Combine the results from children.
→ If a leaf node does not produce results before a timeout, then they are omitted.
SELECT COUNT(*) FROM events
WHERE type = 'crash'
AND time = 'Monday'
SCUBA ARCHITECTURE

```
SELECT COUNT(*) FROM events
WHERE type = 'crash'
AND time = 'Monday'
```
SCUBA ARCHITECTURE

```sql
SELECT COUNT(*) FROM events
WHERE type = 'crash'
AND time = 'Monday'
```
SELECT COUNT(*) FROM events
WHERE type = 'crash'
AND time = 'Monday'

SCUBA ARCHITECTURE

Root
30+40+20=90

Aggregator
10+20=30

Leaf Node
10

Aggregator
25+15=40

Leaf Node
20

Aggregator
20

Leaf Node
20
Facebook maintains multiple Scuba clusters that contain the same databases.

Every query is executed on all the clusters at the same time.

It compares the amount of missing data each cluster had when executing the query to determine which one produced the most accurate result.

→ Track the number of tuples examined vs. number of tuples inserted via Validation Service.
Distributed **document** DBMS started in 2007.

→ Document → Tuple
→ Collection → Table/Relation

Open-source (Server Side Public License)

Centralized shared-nothing architecture.

Concurrency Control:
→ OCC with multi-granular locking
PHYSICAL DENORMALIZATION

A customer has orders and each order has order items.

- **Customers**: $R_1$ (custId, name, ...)
- **Orders**: $R_2$ (orderId, custId, ...)
- **Order Items**: $R_3$ (itemId, orderId, ...)

*Note: The diagram shows a coalesced structure where the relationships between the three relations are combined into a single set.*
A customer has orders and each order has order items.
A customer has orders and each order has order items.
QUERY EXECUTION

JSON-only query API

No cost-based query planner / optimizer. → Heuristic-based + "random walk" optimization.

JavaScript UDFs (not encouraged).

Supports server-side joins (only left-outer?).

Multi-document transactions.
DISTRIBUTED ARCHITECTURE

Heterogeneous distributed components.
→ Shared nothing architecture
→ Centralized query router.

Master-slave replication.

Auto-sharding:
→ Define 'partitioning' attributes for each collection (hash or range).
→ When a shard gets too big, the DBMS automatically splits the shard and rebalances.
MongoDB Cluster Architecture

Application Server

Get Id=101

Router (mongos)

Router (mongos)

Config Server (mongod)

<table>
<thead>
<tr>
<th>Shard</th>
<th>ID Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1-100</td>
</tr>
<tr>
<td>P2</td>
<td>101-200</td>
</tr>
<tr>
<td>P3</td>
<td>201-300</td>
</tr>
<tr>
<td>P4</td>
<td>301-400</td>
</tr>
</tbody>
</table>
STORAGE ARCHITECTURE

Originally used `mmap` storage manager
→ No buffer pool.
→ Let the OS decide when to flush pages.
→ Single lock per database.

MongoDB v3 supports pluggable storage backends
→ **WiredTiger** from BerkeleyDB alumni.
→ **RocksDB** from Facebook (“MongoRocks”)
CockroachDB

Started in 2015 by ex-Google employees.
Open-source (BSL – MariaDB)
Decentralized shared-nothing architecture using range partitioning.
Log-structured on-disk storage (RocksDB)
Concurrency Control:
→ MVCC + OCC
→ Serializable isolation only
Multi-layer architecture on top of a replicated key-value store.

→ All tables and indexes are stored in a giant sorted map in the k/v store.

Uses RocksDB as the storage manager at each node.

Raft protocol (variant of Paxos) for replication and consensus.
DBMS uses **hybrid clocks** (physical + logical) to order transactions globally.

→ Synchronized wall clock with local counter.

Txns stage writes as "intents" and then checks for conflicts on commit.

All meta-data about txns state resides in the key-value store.
COCKROACHDB OVERVIEW

Application

Node 1

Node 2

Node 3

Node n

ID: 1-100 → Node 1
ID: 101-200 → Node 2
ID: 201-300 → Node 3

Id=50
CockroachDB Overview

Application

Node 1
- Leader

Node 2

Node 3

Node n

Raft

Update Id=50

ID: 1-100 → Node 1
ID: 101-200 → Node 2
ID: 201-300 → Node 3
COCKROACHDB OVERVIEW

Application

Node 1
- 1-100
- 101-200
- 201-300

Node 2
- 1-100
- 101-200
- 201-300

Node 3
- 1-100
- 101-200
- 201-300

Node n

ID: 1-100 → Node1
ID: 101-200 → Node2
ID: 201-300 → Node3

Id=150
CockroachDB Overview

Application

Node 1
1-100
101-200
201-300

Node 2
1-100
101-200
201-300

Node 3
1-100
101-200
201-300

Node n

Get Id=150

Leader
ANDY'S CONCLUDING REMARKS

Databases are awesome.
→ They cover all facets of computer science.
→ We have barely scratched the surface...

Going forth, you should now have a good understanding how these systems work.

This will allow you to make informed decisions throughout your entire career.
→ Avoid premature optimizations.