Course Review

Instructor: Matei Zaharia

[cs245.stanford.edu]
What Are Data-Intensive Systems?

Relational databases: most popular type of data-intensive system (MySQL, Oracle, etc)

Many systems facing similar concerns: message queues, key-value stores, streaming systems, ML frameworks, your custom app?

Goal: learn the main issues and principles that span all data-intensive systems
Typical System Challenges

**Reliability** in the face of hardware crashes, bugs, bad user input, etc

**Concurrency**: access by multiple users

**Performance**: throughput, latency, etc

**Access interface** from many, changing apps

**Security** and data privacy
Basic Components

Clients / users

Queries

Data mgmt. system

Logical dataset (e.g. table, graph)

Physical storage (data structures)

Administrator

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Address</th>
<th>City</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Philly</td>
<td>Mouse</td>
<td>123 Fantasy Way</td>
<td>Arabel</td>
<td>73</td>
</tr>
<tr>
<td>Red</td>
<td>Hax</td>
<td>562 Capeve Ave</td>
<td>Gotham</td>
<td>94</td>
</tr>
<tr>
<td>Wooner</td>
<td>H CAMERA</td>
<td>187 Truth Way</td>
<td>Portland</td>
<td>39</td>
</tr>
<tr>
<td>Donald</td>
<td>Dick</td>
<td>555 Quest Street</td>
<td>Hudson</td>
<td>65</td>
</tr>
<tr>
<td>Bono</td>
<td>Barry</td>
<td>587 Central Street</td>
<td>Dallas</td>
<td>36</td>
</tr>
<tr>
<td>Wicky</td>
<td>Crytak</td>
<td>988 Snow Way</td>
<td>Ceylon</td>
<td>44</td>
</tr>
<tr>
<td>Cel</td>
<td>Wonesia</td>
<td>244 Pacific Street</td>
<td>Haskell</td>
<td>32</td>
</tr>
<tr>
<td>avirus</td>
<td>Bird</td>
<td>945</td>
<td>Bolivia</td>
<td>20</td>
</tr>
</tbody>
</table>
Two Big Ideas

Declarative interfaces
» Apps specify *what* they want, not *how* to do it
» Example: “store a table with 2 integer columns”, but not how to encode it on disk
» Example: “count records where column1 = 5”

Transactions
» Encapsulate multiple app actions into one *atomic* request (fails or succeeds as a whole)
» Concurrency models for multiple users
» Clear interactions with failure recovery
Key Concepts: Architecture

**Traditional RDBMS:** self-contained end to end system

**Data lake:** separate storage from compute engines to let many engines use same data
Key Concepts: Hardware

- Latency, throughput, capacity
- Random vs sequential I/Os
- Caching & 5-minute rule
Key Concepts: Data Storage

Field encoding

Record encoding: fixed/variable format, etc

Table encoding: row or column oriented

Data ordering

Indexes: dense, sparse, B+ trees, hashing, multi-dimensional
Key Concepts: Query Execution

Query representation  
(e.g. SQL)

Logical query plan  
(e.g. relational algebra)

Optimized logical plan

Physical plan  
(code/operators to run)

Many execution methods: per-record exec, vectorization, compilation
Key Concepts: Relational Algebra

∩, U, −, ×, σ, π, ⊙, G

Algebraic rules involving these
Key Concepts: Optimization

**Rule-based:** systematically replace some expressions with other expressions

**Cost-based:** propose several execution plans and pick best based on a **cost model**

**Adaptive:** update execution plan at runtime

**Data statistics:** can be computed or estimated cheaply to guide decisions
Key Concepts: Correctness

Consistency constraints: generic way to define correctness with Boolean predicates

Transaction: collection of actions that preserve consistency

Transaction API: commit, abort, etc
Key Concepts: Recovery

Failure models

Undo, redo, and undo/redo logging

Recovery rules for various algorithms (including handling crashes during recovery)

Checkpointing and its effect on recovery

External actions → idempotence, 2PC
Key Concepts: Concurrency

**Isolation levels**, especially **serializability**

» Testing for serializability: conflict serializability, precedence graphs

**Locking**: lock modes, hierarchical locks, and lock schedules (well formed, legal, 2PL)

**Optimistic validation**: rules and pros+cons

**Recoverable, ACR & strict** schedules
Categories of Schedules

- Serializable
  - Conflict serializable
    - 2PL
    - Val
    - Serial
Key Concepts: Distributed

Partitioning and replication

Consensus: nodes eventually agree on one value despite up to F failures

2-Phase commit: parties all agree to commit unless one aborts (no permanent failures)

Parallel queries: comm cost, load balance, faults

BASE and relaxing consistency
Key Concepts: Security and Data Privacy

Threat models

**Security goals:** authentication, authorization, auditing, confidentiality, integrity etc

**Differential privacy:** definitions, computing sensitivity & stability
Putting These Concepts Together

How can you integrate these different concepts into a coherent system design?

How to change system to meet various goals (performance, concurrency, security, etc)?