CS 245: Principles of Data-Intensive Systems

Instructor: Matei Zaharia

cs245.stanford.edu
Outline

Why study data-intensive systems?

Course logistics

Key issues and themes

A bit of history
My Background

PhD in 2013

Open source distributed data processing framework

Cofounder of analytics company

Research in systems for ML
Why Study Data-Intensive Systems?

Most important computer applications must manage, update and query datasets
  » Bank, store, fleet controller, search app, …

Data quality, quantity & timeliness becoming even more important with AI
  » Machine learning = algorithms that generalize from data
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
Practical Benefits of Studying These Systems

Learn how to select & tune data systems

Learn how to build them

Learn how to build apps that have to tackle some of these same challenges

» E.g. cross-geographic-region billing app, custom search engine, etc
Scientific Interest

Interesting algorithmic and design ideas

In many ways, data systems are the highest-level successful programming abstractions
Programming: The Dream

High-level spec

∀i \sum_{x \in S_i \cup T_i} \lambda x. x^2(...)

Working application
∀ " # ∈ & ' ∪ *+. ( …) Working application High-level spec
Programming: The Reality

How to horizontally center a `<div>`?

Why calling `setState` method doesn't mutate the state immediately?

How to find which version of TensorFlow is installed in my system?

Why does HTML think “chucknorris” is a color?

How come certain random strings produce colors when entered as background colors in HTML? For example:

```
<body bgcolor="chucknorris"> test </body>
```

...produces a document with a red background across all browsers and platforms.

Interestingly, while `chucknorris` produces a red background as well, `chucknorr` produces a yellow background.
Programming with Databases

High-level spec

Relational algebra

Actually manages:
- Durability
- Concurrency
- Query optimization
- Security
- …
Outline

Why study data-intensive systems?

Course logistics

Key issues and themes

A bit of history
Teaching Assistants

Ben Braun
Edward Gan
Leo Mehr
Deepak Narayanan
Pratiksha Thaker
James Thomas
Course Format

Lectures in class

Assigned paper readings (Q&A in class)

3 programming assignments

Midterm and final

This is the 1st run of my version of the course, so we’re still figuring some things out
Paper Readings

A few classic or recent research papers

Read the paper **before** the class: we want to discuss it together!

We’ll post discussion questions on the class website a week before lecture
How Should You Read a Paper?

Read: “How to Read a Paper”

TLDR: don’t just go through end to end; focus on key ideas/sections.
Our First Paper

We’ll be reading part of “A History and Evaluation of System R” for next class!

Find instructions and questions on website
Programming Assignments

Three assignments implemented in Java or Scala, and submitted online

1. Storage and access methods
2. Query optimization
3. Transactions and recovery

Done individually; A1 posted next week
Midterm and Final

Written tests based on material covered in lectures, assignments and readings

Final will cover the entire course but focus on the second half
Grading

45% Assignments (15% each)

25% Midterm

30% Final
Keeping in Touch

Sign up for Piazza on the course website to receive announcements!

cs245.stanford.edu
Outline

Why study data-intensive systems?

Course logistics

Key issues and themes

A bit of history
Recall: Examples of 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?
Basic Components

Clients / users

Queries

Data mgmt. system

Logical dataset
(e.g. table, graph)

Physical storage
(data structures)

Administrator
# Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, ...</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, ...</td>
</tr>
</tbody>
</table>
# Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, …</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, …</td>
</tr>
<tr>
<td>TensorFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, …</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, …</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Tensors</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, ...</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, ...</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Tensors</td>
<td>NCHW, NHWC, sparse arrays, ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, …</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, …</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Tensors</td>
<td>NCHW, NHWC, sparse arrays, …</td>
<td>Python DAG construction</td>
<td></td>
</tr>
</tbody>
</table>
Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, ...</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, ...</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Tensors</td>
<td>NCHW, NHWC, sparse arrays, ...</td>
<td>Python DAG construction</td>
<td>query planning, distribution, specialized HW</td>
</tr>
</tbody>
</table>
## Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, ...</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, ...</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Tensors</td>
<td>NCHW, NHWC, sparse arrays, ...</td>
<td>Python DAG construction</td>
<td>query planning, distribution, specialized HW</td>
</tr>
<tr>
<td>Apache Kafka</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, …</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, …</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Tensors</td>
<td>NCHW, NHWC, sparse arrays, …</td>
<td>Python DAG construction</td>
<td>query planning, distribution, specialized HW</td>
</tr>
<tr>
<td>Apache Kafka</td>
<td>Streams of opaque records</td>
<td>Partitions, compaction</td>
<td>Publish, subscribe</td>
<td>Durability, rescaling</td>
</tr>
</tbody>
</table>
# Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes,...</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations,...</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Tensors</td>
<td>NCHW, NHWC, sparse arrays, ...</td>
<td>Python DAG construction</td>
<td>query planning, distribution, specialized HW</td>
</tr>
<tr>
<td>Apache Kafka</td>
<td>Streams of opaque records</td>
<td>Partitions, compaction</td>
<td>Publish, subscribe</td>
<td>Durability, rescaling</td>
</tr>
<tr>
<td>Apache Spark RDDs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Examples

<table>
<thead>
<tr>
<th>System</th>
<th>Logical Data Model</th>
<th>Physical Storage</th>
<th>API</th>
<th>Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational databases</td>
<td>Relations (i.e. tables)</td>
<td>B-trees, column stores, indexes, ...</td>
<td>SQL, ODBC</td>
<td>Durability, transactions, query planning, migrations, ...</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Tensors</td>
<td>NCHW, NHWC, sparse arrays, ...</td>
<td>Python DAG construction</td>
<td>query planning, distribution, specialized HW</td>
</tr>
<tr>
<td>Apache Kafka</td>
<td>Streams of opaque records</td>
<td>Partitions, compaction</td>
<td>Publish, subscribe</td>
<td>Durability, rescaling</td>
</tr>
<tr>
<td>Apache Spark RDDs</td>
<td>Collections of Java objects</td>
<td>Read external systems, cache</td>
<td>Functional API, SQL</td>
<td>Distribution, query planning, transactions*</td>
</tr>
</tbody>
</table>
Some Typical Concerns

**Access interface** from many, changing apps

**Performance:** throughput, latency, etc

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

**Concurrency:** access by multiple users

**Security** and data privacy
What should happen if two consumers read() at the same time?
Example

Message queue system

Producers

Consumers

What should happen if a consumer reads a message but then immediately crashes?
Example

Message queue system

Can a producer put in 2 messages atomically?
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
Declarative Interface Examples

SQL
  » Abstract “table” data model, many physical implementations
  » Specify queries in a restricted language that the database can optimize

TensorFlow
  » Operator graph gets mapped & optimized to different hardware devices

Functional programming (e.g. MapReduce)
  » Says what to run but not how to do scheduling
Transaction Examples

SQL databases
  » Commands to start, abort or end transactions based on multiple SQL statements

Apache Spark, MapReduce
  » Make the multi-part output of a job appear atomically when all partitions are done

Stream processing systems
  » Count each input record exactly once despite crashes, network failures, etc
Outline

Why study data-intensive systems?

Course logistics

Key issues and themes

A bit of history
Early Data Management

At first, each application did its own data management directly against storage

Ye Olde Bank

I’d like a computerized account system

write_block()

I have just the thing

read_block()

Stores 5 MB!
Problems with App Storage Management

How should we lay out and navigate data?

How do we keep the application reliable?

What if we want to share data across apps?

Every app is solving the same problems!
Navigational Databases (1964)

CODASYL, IDS

Data is graph of records

Procedural API based on navigating links:

get department with name='Sales'
get first employee in set department-employees
until end-of-set do {
    get next employee in set department-employees
    process employee
}

“Data independence”: app code not tied to storage details
I raise the example of Copernicus today to illustrate a parallel that I believe exists in the computing or, more properly, the information systems world. We have spent the last 50 years with almost Ptolemaic information systems. These systems, and most of the thinking about systems, were based on a “computer centered” concept.

A new basis for understanding is available in the area of information systems. It is achieved by a shift from a computer-centered to the database-centered point of view. This new understanding will lead to new solutions to our database problems and speed our conquest of the $n$-dimensional data structures which best model the complexities of the real world.

Charles W. Bachman, “The Programmer as Navigator”
Edgar F. (Ted) Codd

Proposed the relational DB model, with declarative queries & storage (1970)

Relation = table with unique key identifying each row

Data independence++: apps don’t even specify how to execute query
Key Ideas in Relational DBMS

Logical data model: tables with references across them (foreign keys)

Relational algebra (e.g. SQL)

Data mgmt. system

Physical storage: raw files, B-trees, hash indexes, etc

Clients / users

Administrator

Query planning, access methods, transactions, etc
Early Relational DBMS

IBM System R (1974): research system
  » Led to IBM SQL/DS in 1981

Ingres (1974): Mike Stonebraker at Berkeley
  » Led to PostgreSQL

Oracle database (released 1979)

Next class, we’ll cover database architecture by looking at System R
Rest of the Course

We’ll explore both “big ideas” we saw, focusing on relational DBs but showing examples in other areas

- Declarative interfaces
  - Data independence and data storage formats
  - Query languages and optimization
- Transactions, concurrency & recovery
  - Concurrency models
  - Failure recovery
  - Distributed storage and consistency

Don’t forget to sign up for Piazza!