Systems for Big Data

Mike Carey
Information Systems Group
Computer Science Department
UC Irvine
Plan For Today’s Talk

- Raising the level: towards declarative tools
  - On saying *what*, not *how!*
- Systems for declarative data management
  - Database management systems
  - Structured query language (SQL)
- Moving from data to Big Data
  - Definition and challenges
  - Current systems (SQL, NoSQL, data analytics platforms)
- A bigger picture: the data lifecycle
  - From ingestion to insights and/or production
Suppose we wanted to make a pizza:

*Imperative* instructions might say...

1. Get a 3” ball of pizza dough.
2. Using a rolling pin, flatten the ball until it is 12” in diameter.
3. Open and spread 4 3oz cans of pizza sauce over the dough.
4. Hand grate 3 oz of mozzarella cheese evenly over the dough.
5. Starting slightly inside the dough, encircle the pizza with evenly spaced 1” pepperoni slices; repeat again and again, moving inwards, until you reach the center of the pizza.
6. Preheat the oven to 350F.
7. When the oven is ready, bake the pizza for 25 minutes.
Suppose we wanted to make a pizza:

**Declarative** instructions would say...

1. Create a pizza with a 12” diameter crust,
2. covered with a 12oz layer of pizza sauce,
3. a 3oz layer of mozzarella cheese,
4. and a layer of 1” pepperoni slices,
5. and baked for 25 minutes at 350F.

Notice how we said *what* we wanted this time, but didn’t have to specify *how* to make it...!
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equipment item</td>
<td>Approximate</td>
<td>quantity</td>
<td>=sum(D2:D8)</td>
<td>2 PCs, one of which is mainly used for backup, but there are also two laptops and a smartphone.</td>
</tr>
<tr>
<td>Desktop PC</td>
<td>£ 400.00</td>
<td>2</td>
<td>£ 800.00</td>
<td></td>
</tr>
<tr>
<td>LCD monitor</td>
<td>£ 120.00</td>
<td>2</td>
<td>£ 240.00</td>
<td></td>
</tr>
<tr>
<td>Laser printer</td>
<td>£ 100.00</td>
<td>1</td>
<td>£ 100.00</td>
<td></td>
</tr>
<tr>
<td>Scanner/colour print</td>
<td>£ 70.00</td>
<td>1</td>
<td>£ 70.00</td>
<td></td>
</tr>
<tr>
<td>Router</td>
<td>£ 50.00</td>
<td>1</td>
<td>£ 50.00</td>
<td></td>
</tr>
<tr>
<td>Laptop</td>
<td>£ 450.00</td>
<td>1</td>
<td>£ 450.00</td>
<td>Use mainly for working</td>
</tr>
<tr>
<td>Smartphone</td>
<td>£ 150.00</td>
<td>1</td>
<td>£ 150.00</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>£ 1,860.00</td>
</tr>
</tbody>
</table>
What *is* a Database System?

- So what’s a *database*?
  - A (very) large, integrated collection of data
- Often a model of a *real-world enterprise* or a history of *real-world events*
  - *Entities* (e.g., students, courses, Facebook users, …)
  - *Relationships* (e.g., Susan is taking CS 234, Susan is a friend of Lynn, Mike filed a grade change for Lynn, …)
- What’s a *database management system* (DBMS)?
  - A software system designed to store, manage, and provide access to one or more such databases
Evolution of DBMS

Files

CODASYL/IMS

Early DBMS Technologies

- Records and pointers
- Large, carefully tuned data access programs that have dependencies on physical access paths, indexes, etc.

Manual Coding

- Byte streams
- Majority of application development effort goes towards building and then maintaining data access logic

Relational

Relational DB Systems

- Declarative approach
- Tables + views bring “data independence”
- Details left to system

New Data

- ...
Why Use a DBMS?

- Data independence
- *Efficient (automatic) data access*
- *Reduced development time*
- *Data integrity and security*
- Uniform data administration
- Concurrent access and recovery from crashes
Data Models

- A **data model** is a collection of concepts for describing data (to one another, or to a DBMS)
- A **schema** is a description of a particular collection of data, using a given data model
- The **relational model** is the most widely used data model today
  - **Relation** – basically a **table** with rows and (named) columns
  - **Schema** – describes the tables and their columns
Many *views* of one *conceptual (logical) schema* and an underlying *physical schema*

- **Views** describe how different users or groups see the data
- **Conceptual schema** defines the logical structure of the database
- **Physical schema** describes the files and indexes used “under the covers”
Example: University DB

- Conceptual schema (a.k.a. stored tables):
  - $\textbf{Students}(\text{sid}: \text{string}, \text{name}: \text{string}, \text{login}: \text{string}, \text{age}: \text{integer})$
  - $\textbf{Courses}(\text{cid}: \text{string}, \text{cname}: \text{string}, \text{credits}: \text{integer})$
  - $\textbf{Enrolled}(\text{sid}: \text{string}, \text{cid}: \text{string}, \text{grade}: \text{string})$

- Physical schema (a.k.a. storage and indexing):
  - Tables each stored internally as unordered files
  - Have indexes on first and third columns of $\text{Students}$

- External schema (a.k.a. views):
  - $\textbf{CourseInfo}(\text{cid}: \text{string}, \text{cname}: \text{string}, \text{enrollment}: \text{integer})$
## Example: University DB (cont.)

### Students

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td><a href="mailto:dusty@aol.com">dusty@aol.com</a></td>
<td>22</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td><a href="mailto:bbrute@gmail.com">bbrute@gmail.com</a></td>
<td>19</td>
</tr>
<tr>
<td>31</td>
<td>Rusty</td>
<td><a href="mailto:rust@hotmail.com">rust@hotmail.com</a></td>
<td>23</td>
</tr>
<tr>
<td>32</td>
<td>Andrew</td>
<td><a href="mailto:andyman@aol.com">andyman@aol.com</a></td>
<td>18</td>
</tr>
<tr>
<td>58</td>
<td>Suzy</td>
<td><a href="mailto:susan@yahoo.com">susan@yahoo.com</a></td>
<td>22</td>
</tr>
<tr>
<td>71</td>
<td>Rosie</td>
<td><a href="mailto:flower@fb.com">flower@fb.com</a></td>
<td>20</td>
</tr>
</tbody>
</table>

### Courses

<table>
<thead>
<tr>
<th>cid</th>
<th>course</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS 101</td>
<td>Programming I</td>
<td>6</td>
</tr>
<tr>
<td>CS 102</td>
<td>Programming II</td>
<td>6</td>
</tr>
<tr>
<td>CS 103</td>
<td>Computer Games</td>
<td>4</td>
</tr>
<tr>
<td>Stat 101</td>
<td>Statistics 1</td>
<td>4</td>
</tr>
</tbody>
</table>

### Enrolled

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>CS 101</td>
<td>B-</td>
</tr>
<tr>
<td>22</td>
<td>CS 103</td>
<td>A-</td>
</tr>
<tr>
<td>58</td>
<td>Stat 101</td>
<td>A</td>
</tr>
<tr>
<td>29</td>
<td>CS 101</td>
<td>C+</td>
</tr>
<tr>
<td>71</td>
<td>CS 103</td>
<td>A+</td>
</tr>
</tbody>
</table>
Example: University DB (cont.)

- User query (in SQL, against the view):
  - SELECT c.cid, c.enrollment
    FROM CourseInfo c
    WHERE c.cname = 'Computer Games'

- Equivalent query (against the stored tables):
  - SELECT e.cid, count(e.*)
    FROM Enrolled e, Courses c
    WHERE e.cid = c.cid AND c.cname = 'Computer Games'
    GROUP BY c.cid

- Under the hood (against the physical schema)
  - Access Courses
    - Use index on cname to find associated cid
  - Access Enrolled
    - Use index on cid to count the enrollments
User query (in SQL, against the view):

- SELECT c.cid, c.enrollment
  FROM CourseInfo c
  WHERE c.cname = 'Computer Games'

Result

<table>
<thead>
<tr>
<th>cid</th>
<th>enrollment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS 103</td>
<td>2</td>
</tr>
</tbody>
</table>

**Declarative!**

IBM DB2, Oracle, Microsoft SQL Server, MySQL, SQLite, PostgreSQL, and others...
So what went on – and why?

What’s going on right now?
How Big is “Big Data”?

This is big data!
DB Systems: Under the Hood

SQL

Query plans

API calls

Query Parser
Query Optimizer
Plan Executor
Relational Operators (+ Utilities)
Files of Records
Access Methods (Indices)
Buffer Manager
Disk Space and I/O Manager

Transaction Manager
Lock Manager
Log Manager

WAL

Data Files
Index Files
Catalog Files
Let’s consider a grocery shopping analogy:

- List 1 = \{milk, cheerios, ice cream, bread\}
Continuing our grocery shopping analogy:

- List 2 = \{milk, cheerios, ice cream, bread, cream, cat food, chicken, coffee, napkins, coke, jelly, kleenex, ...(87)..., water\}

Think of disk accesses as being similar to aisle visits!

Sorting by aisle could help a lot!
Enterprises wanted to store and query historical business data (data warehouses)

- 1970’s: Relational databases appeared (w/SQL)
- Late 1970’s: Database machines based on novel hardware and early (brute force) parallelism
- 1980’s: Parallel database systems based on “shared-nothing” architectures (Gamma, GRACE, Teradata)
- 2000’s: Netezza, Aster Data, DATAlegro, Greenplum, Vertica, ParAccel, ... (Serious “Big $” acquisitions!)

(Each node runs an instance of an indexed database data storage and runtime system)
Late 1990’s brought a need to index and query the rapidly exploding content of the Web
- SQL-based databases didn’t fit the problem(s)
- Google, Yahoo! et al had to do something

Google responded by laying a new foundation
- Google File System (GFS)
  - OS-level byte stream files spanning 1000’s of machines
  - 3-way replication for fault-tolerance (and high availability)
- MapReduce (MR) programming model
  - User writes just two simple functions: Map and Reduce
  - “Parallel programming for dummies” – MR runtime does all the heavy lifting (using partitioned parallelism)
MapReduce: A Quick Example

Romeo, Romeo, wherefore art thou Romeo?
What, art thou hurt?

```
Romeo, 1
Romeo, 1
wherefore, 1
art, 1
thou, 1
Romeo, 1

art, (1, 1)
hurt (1),
thou (1, 1)

art, 2
hurt, 1
thou, 2
```

```
map

reduce

map

reduce

```

```
What, 1
art, 1
thou, 1
hurt, 1

Romeo, (1, 1, 1)
wherefore, (1)
what, (1)

Romeo, 3
wherefore, 1
what, 1
```

Input Splits (distributed)
Mapper Outputs
Reducer Inputs
Reducer Outputs (distributed)

SHUFFLE PHASE (based on keys)

"Partitioned Parallelism" (can scale up to 1000’s of nodes)
MapReduce Programming model

- Inputs and outputs are sets of key/value pairs
- Programmers simply provide two functions
  - map($K_1, V_1$) -> list($K_2, V_2$)
    - Produces list of intermediate key/value pairs for each input key/value pair
  - reduce($K_2, \text{list}(V_2)$) -> list($K_3, V_3$)
    - Produces a list of result values for all intermediate values that are associated with the same intermediate key
- In our word count example, notice that
  - The keys were the words and the counts were the values
  - *We never had to think about parallelism!* 
Yahoo!, Facebook, and friends cloned Google’s “Big Data” infrastructure from papers
- GFS → Hadoop Distributed File System (HDFS)
- MapReduce → Hadoop MapReduce
- Widely used for Web indexing, click stream analysis, log analysis, information extraction, some machine learning

Tired of puzzle-solving with just two moves, higher-level languages were developed to “hide” MR
- E.g., Pig (Yahoo!), Hive (Facebook), Jaql (IBM)
- Now in heavy use over MR (Pig > 60%, HiveQL > 90%)

Similar happenings at Microsoft
- Cosmos, Dryad, and SCOPE (which powers Bing)
Other Up-and-Coming Platforms

- Bulk Synchronous Programming (BSP) platforms, e.g., Pregel, Giraph, GraphLab, ..., for doing Big* Graph analysis
  
  “Think Like a Vertex”
  - Receive messages
  - Update state
  - Send messages

  (* Big is the platform's problem)

- Spark for in-memory cluster computing – for repetitive data analyses, iterative machine learning tasks, ...

\[\text{Input} \rightarrow \text{Distributed memory} \rightarrow \text{query 1} \rightarrow \text{query 2} \rightarrow \text{query 3} \rightarrow \text{Iter. 1} \rightarrow \text{Iter. 2} \rightarrow \text{Iter. 3} \rightarrow \text{...} \]
AsterixDB System (UCI / UCR)
(http://asterixdb.incubator.apache.org)

ASTERIX Goal:
Ingest, digest, persist, index, manage, query, analyze, and publish massive quantities of semi-structured information...

(ADM = ASTERIX Data Model, AQL = ASTERIX Query Language)
create dataverse TinySocial;
use dataverse TinySocial;

create type MugshotUserType as {
  id: int32,
  alias: string,
  name: string,
  user-since: datetime,
  address: {
    street: string,
    city: string,
    state: string,
    zip: string,
    country: string
  },
  friend-ids: {{ int32 }},
  employment: [EmploymentType]
}

create type EmploymentType as open {
  organization-name: string,
  start-date: date,
  end-date: date?
}

“NoSQL” characteristics include...
- Objects can be nested
- Fields can be multivalued (plural)
- Content can vary from object to object
- Schemas are not mandatory

(Other examples: MongoDB, Couchbase,...)

- Rich type support (space, time, ...)
- Records, lists, bags
- Open vs. closed types
Ex: MugshotUsers Data

```json
{ "id":1, "alias":"Margarita", "name":"MargaritaStoddard", "address":{
  "street":"234 Thomas Ave", "city":"San Hugo", "zip":"98765",
  "state":"CA", "country":"USA" }
"user-since":datetime("2012-08-20T10:10:00"),
"friend-ids":{{ 2, 3, 6, 10 }}, "employment":[
  "organization-name":"Codetechno", "start-date":date("2006-08-06") }
}

{ "id":2, "alias":"Isbel", "name":"IsbelDull", "address":{
  "street":"345 James Ave", "city":"San Hugo", "zip":"98765",
  "state":"CA", "country":"USA" },
"user-since":datetime("2011-01-22T10:10:00"),
"friend-ids":{{ 1, 4 }}, "employment":[
  "organization-name":"Hexviafind", "start-date":date("2010-04-27") ] }

{ "id":3, "alias":"Emory", "name":"EmoryUnk", "address":{
  "street":"456 Jose Ave", "city":"San Hugo", "zip":"98765",
  "state":"CA", "country":"USA" },
"user-since": datetime("2012-07-10T10:10:00"),
"friend-ids":{{ 1, 5, 8, 9 }}, "employment":[
  "organization-name":"geomed",
  "start-date":date("2010-06-17"), "end-date":date("2010-01-26") ] }
```
Ex: Identify active users and group/count them by country:

```plaintext
with $end := current-datetime()
with $start ::= $end - duration("P30D")
from $user in dataset MugshotUsers
where some $logrecord in dataset AccessLog
  satisfies $user.alias = $logrecord.user
    and datetime($logrecord.time) >= $start
    and datetime($logrecord.time) <= $end

group by $country ::= $user.address.country with $user
select {
  "country" : $country,
  "active users" : count($user)
}
```
AsterixDB Status

- 4 year initial NSF project (250+ KLOC @ UCI+UCR)
- AsterixDB BDMS is here! (First shared June 6th, 2013)
  - Semistructured “NoSQL“ style data model
  - Declarative parallel queries, inserts, deletes, ...
  - LSM-based storage/indexes (primary & secondary)
  - Internal and external datasets both supported
  - Rich set of data types (including text, time, location)
  - Fuzzy and spatial query processing
  - NoSQL-like transactions (for inserts/deletes)
  - Data feeds and external indexes in next release
- Performance competitive w/parallel relational DBMS, MongoDB, and Hive (see papers)
- *Now in Apache open source incubation mode!*
Some Use Case Examples

- Recent or projected use case areas include...
  - Behavioral science (at UCI)
  - Social data analytics
  - Cell phone event analytics
  - Power usage monitoring
  - Public health (joint effort with UCIPT@UCLA)
  - Cluster management log analytics
  - Your future use cases go here... 😊
I’ve just described one piece of the Data Science “Big Data puzzle”...
What We’ve Touched On

- Raising the level: towards *declarative* tools
  - It’s all about saying *what*, not *how*!
- Systems for declarative *data management*
  - Database management systems
  - Structured query language (SQL) in particular
- Moving from data to *Big Data*
  - Definition of “big” and some of the challenges
  - Current systems (SQL, NoSQL, data analytics platforms)
- The bigger picture: the *data lifecycle*
  - From ingestion to insights and production (and repeat!)
Questions?