Data Informatics

Seon Ho Kim, Ph.D.
seonkim@usc.edu
NoSQL and Big Data Processing
Database

- Relational Databases – mainstay of business
- Web-based applications caused spikes
  - Especially true for public-facing e-Commerce sites
- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
- Known as ‘scaling out’ or ‘horizontal scaling’

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relation
Short Review of SQL

Standard language for querying and manipulating data

Structured Query Language

• Data Definition Language (DDL)
  – Create/alter/delete tables and their attributes

• Data Manipulation Language (DML)
  – Query one or more tables – discussed next!
  – Insert/delete/modify tuples in tables
### Tables in SQL

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Tables Explained

- The *schema* of a table is the table name and its attributes:

  $Product(PName, \ Price, Category, Manufacturer)$

- A *key* is an attribute whose values are unique; we underline a key

  $Product(PName, \ Price, Category, Manufacturer)$
SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT <attributes>
FROM  <one or more relations>
WHERE <conditions>
```
Simple SQL Query

```
SELECT * 
FROM Product 
WHERE category= 'Gadgets'
```

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

“selection”
### Simple SQL Query

**SELECT** PName, Price, Manufacturer  
**FROM** Product  
**WHERE** Price > 100
Joins

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

---

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer=CName AND Country= ‘Japan’ AND Price <= 200
```
SQL Physical Layer Abstraction

• Applications specify what, not how
• Query optimization engine
• Physical layer can change without modifying applications
  – Create indexes to support queries
  – In Memory databases
Transactions – ACID Properties

- **Atomic** – All of the work in a transaction completes (commit) or none of it completes.
- **Consistent** – A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
- **Isolated** – The results of any changes made during a transaction are not visible until the transaction has committed.
- **Durable** – The results of a committed transaction survive failures.
OLTP vs. OLAP

• We can divide IT systems into transactional (OLTP) and analytical (OLAP). In general we can assume that OLTP systems provide source data to data warehouses, whereas OLAP systems help to analyze it.
Challenges of Scale Differ
Motives Behind NoSQL

• Big data.
• Scalability.
• Data format.
• Manageability.
Big Data

- Collect.
- Store.
- Organize.
- Analyze.
- Share.

- Data growth outruns the ability to manage it so we need scalable solutions.
Scalability

• Scale up, Vertical scalability.
  – Increasing server capacity.
  – Adding more CPU, RAM.
  – Managing is hard.
  – Possible down times
Scalability

- Scale out, Horizontal scalability.
  - Adding servers to existing system with little effort, aka Elastically scalable.
    - Bugs, hardware errors, things fail all the time.
    - It should become cheaper. Cost efficiency.
  - Shared nothing.
  - Use of commodity/cheap hardware.
  - Heterogeneous systems.
  - Service Oriented Architecture. Local states.
    - Decentralized to reduce bottlenecks.
    - Avoid single point of failures.
What is Wrong With RDBMS?

• Nothing. One size fits all? Not really.
• Rigid schema design → Efficiency issue in big scale.
• Harder to scale.
• Replication → Expensive.
• Joins across multiple nodes? Hard.
• How does RDMS handle data growth? Hard.
• Need for a DBA.
• Many programmers already familiar with it → Good.
• Transactions/ACID make development easy → Good.
• Lots of tools to use → Good.
ACID Semantics

• Atomicity, Consistency, Isolation, Durability
• Any data store can achieve Atomicity, Isolation and Durability but do you always need consistency? No.
  
• *By giving up ACID properties, one can achieve higher performance and scalability.*
The Perfect Storm

• Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm
• Not a backlash/rebellion against RDBMS
• SQL is a rich query language that cannot be rivaled by the current list of NoSQL offerings
CAP Theorem

• Three properties of a system:
  – consistency, availability and partitions
• You can have at most two of these three properties for any shared-data system
• To scale out, you have to partition. That leaves either consistency or availability to choose from
  – In almost all cases, you would choose availability over consistency
Theorem: You can have at most **two** of these properties for any shared-data system.
BASE, an ACID Alternative

• Almost the opposite of ACID.
• Basically Available: Nodes in the a distributed environment can go down, but the whole system shouldn’t be affected.
• Soft State (scalable): The state of the system and data changes over time.
• Eventual Consistency: Given enough time, data will be consistent across the distributed system.
A Clash of cultures

ACID:
• Strong consistency.
• Less availability.
• Pessimistic concurrency.
• Complex.

BASE:
• Availability is the most important thing. Willing to sacrifice for this (CAP).
• Weaker consistency (Eventual).
• Best effort.
• Simple and fast.
• Optimistic.
NoSQL Definition

• From www.nosql-database.org:
  – Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly.
  – Often more characteristics apply as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge data amount, and more.
What is NoSQL?

- Stands for **Not Only SQL**
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins
- All NoSQL offerings relax one or more of the ACID properties (following CAP theorem)
Why NoSQL?

• For data storage, an RDBMS cannot be the be-all/end-all
• Just as there are different programming languages, need to have other data storage tools in the toolbox
• A NoSQL solution is more acceptable to a client now than even a year ago
NoSQL Distinguishing Characteristics

- Large data volumes
  - Google’s “big data”
- Scalable replication and distribution
  - Potentially thousands of machines
  - Potentially distributed around the world
- Queries need to return answers quickly
- Mostly query, few updates

- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE
- CAP Theorem
- Open source development
What kinds of NoSQL

- NoSQL solutions fall into two major areas:
  - Key/Value or ‘the big hash table’.
    - Amazon S3 (Dynamo)
    - Voldemort
    - Scalaris
    - Memcached (in-memory key/value store)
    - Redis
  - Schema-less which comes in multiple flavors, column-based, document-based or graph-based.
    - Cassandra (column-based)
    - CouchDB (document-based)
    - MongoDB (document-based)
    - Neo4J (graph-based)
    - HBase (column-based)
NoSQL Databases

• The **key-value data model** is based on a structure composed of two data elements: a key and a value, in which every key has a corresponding value or set of values.

• The key value model is also referred to as the attribute-value or associative data model.

• Consider the example shown on the next slide which illustrates a small truck-driving company called Trucks-R-Us. Each of its three drivers has one or more certifications and other general information.

• Based on this example we can define the following important points:
NoSQL Databases

Data stored using traditional relational model

<table>
<thead>
<tr>
<th>DID</th>
<th>CERT1</th>
<th>CERT2</th>
<th>CERT3</th>
<th>DOB</th>
<th>LICTYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2732</td>
<td>80</td>
<td>90</td>
<td></td>
<td>1/24/1962</td>
<td>P</td>
</tr>
<tr>
<td>2946</td>
<td></td>
<td>92</td>
<td>90</td>
<td>4/11/1970</td>
<td></td>
</tr>
<tr>
<td>3650</td>
<td>86</td>
<td></td>
<td></td>
<td>6/27/1968</td>
<td>C</td>
</tr>
</tbody>
</table>

Data stored using key-value model

<table>
<thead>
<tr>
<th>DID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2732</td>
<td>CERT1</td>
<td>80</td>
</tr>
<tr>
<td>2732</td>
<td>CERT3</td>
<td>90</td>
</tr>
<tr>
<td>2732</td>
<td>DOB</td>
<td>1/24/1962</td>
</tr>
<tr>
<td>2732</td>
<td>LICTYPE</td>
<td>P</td>
</tr>
<tr>
<td>2946</td>
<td>CERT2</td>
<td>92</td>
</tr>
<tr>
<td>2946</td>
<td>CERT3</td>
<td>90</td>
</tr>
<tr>
<td>2946</td>
<td>DOB</td>
<td>4/11/1970</td>
</tr>
<tr>
<td>3650</td>
<td>CERT1</td>
<td>86</td>
</tr>
<tr>
<td>3650</td>
<td>DOB</td>
<td>6/27/1968</td>
</tr>
<tr>
<td>3650</td>
<td>LICTYPE</td>
<td>C</td>
</tr>
</tbody>
</table>

- In the relational model:
  - Each row represents one entity instance
  - Each column represents one attribute of the entity
  - The values in a column are of the same data type
- In the key-value model:
  - Each row represents one attribute/value of one entity instance
  - The “key” column could represent any entity’s attribute
  - The values in the “value” column could be of any data type and therefore it is generally assigned a long string data type
NoSQL Databases

• In the key-value model, each row represents one attribute of one entity instance. The “key” column points to an attribute and the “value” column contains the actual value for the attribute.

• The data type of the “value” column is generally a long string to accommodate the variety of actual data types of the values placed in the column.

• To add a new entity attribute in the relational model, you need to modify the table definition (schema modification). To add a new attribute in the key-value model, you add a row to the key-value store, which is why it is said to be “schema-less”.
NoSQL Databases

- NoSQL databases do not store or enforce relationships among entities. The programmer is required to manage the relationships in the code. Also, all data and integrity validations must be done in the code.

- NoSQL databases use their own native application programming interface (API) with simple data access commands, such as `put`, `read`, and `delete`. Because there is no declarative SQL-like syntax to retrieve data, the program code must take care of retrieving related data in the correct way.

- Indexing and searches can be difficult. Because the “value” column in the key-value model could contain many different data types, it is often difficult to create indices on the data. At the same time, searches can become very complex.
NoSQL Databases

• One of the big advantages of NoSQL databases is that they generally use a distributed architecture.

• NoSQL databases can handle very large volumes of data. In particular, they are suited for sparse data.

• **Sparse data** occurs when the number of attributes is very large, but the number of actual data instances is low. Using the truck driving company as an example, drivers can take any certification exam, but they are not required to take them all. In this case there were three drivers and three possible certificates for each driver, so there will be nine possible data points (we illustrated only 4). Extrapolate this to a hospital with 50,000 patients and more than 2500 possible medical tests that could be performed. We would not expect to see 50000 x 2500 data points, in reality we would see far less.
NoSQL Databases

• Most NoSQL databases are **geared toward performance** rather than transaction consistency.

• Enforcing data consistency on a distributed database is a very difficult problem. Distributed databases make copies of data elements at multiple nodes to ensure high availability and fault tolerance. Updating values and requiring consistency amongst all copies is a very tough problem to solve. NoSQL database sacrifice this consistency to attain high levels of performance.

• Some NoSQL databases provide a feature called **eventual consistency**, which means that updates to data will propagate through the system and eventually all copies will be consistent. With eventual consistency, data are not guaranteed to be consistent across all copies immediately after an update.
Key/Value Data Model

• Pros:
  – very fast
  – very scalable
  – simple model
  – able to distribute horizontally

• Cons:
  – many data structures (objects) can't be easily modeled as key value pairs
Schema-Less Data Model

• Pros:
  – Schema-less data model is richer than key/value pairs
  – eventual consistency
  – many are distributed
  – still provide excellent performance and scalability

• Cons:
  – typically no ACID transactions or joins
Common Advantages

• Cheap, easy to implement (open source)
• Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
  – Down nodes easily replaced
  – No single point of failure
• Easy to distribute
• Don't require a schema
• Can scale up and down
• Relax the data consistency requirement (CAP)
What am I giving up?

• Features
  – joins
  – group by
  – order by
  – ACID transactions

• SQL is sometimes frustrating but still a powerful query language

• Easy integration with other applications that support SQL
Storing and Modifying Data

• Syntax varies
  – HTML
  – Java Script
  – Etc.
• Asynchronous – Inserts and updates do not wait for confirmation
• Versioned
• Optimistic Concurrency
Retrieving Data

- Syntax Varies
  - No set-based query language
  - Procedural program languages such as Java, C, etc.
- Application specifies retrieval path
- No query optimizer
- Quick answer is important
- May not be a single “right” answer
Open Source

• Small upfront software costs
• Suitable for large scale distribution on commodity hardware
NoSQL Summary

- NoSQL databases reject:
  - Overhead of ACID transactions
  - “Complexity” of SQL
  - Burden of up-front schema design
  - Declarative query expression
  - Yesterday’s technology

- Programmer responsible for
  - Step-by-step procedural language
  - Navigating access path
Summary

• SQL Databases
  – Predefined Schema
  – Standard definition and interface language
  – Tight consistency
  – Well defined semantics

• NoSQL Database
  – No predefined Schema
  – Per-product definition and interface language
  – Getting an answer quickly is more important than getting a correct answer
Web References

• “NoSQL -- Your Ultimate Guide to the Non - Relational Universe!”
  http://nosql-database.org/links.html
• “NoSQL (RDBMS)”
  http://en.wikipedia.org/wiki/NoSQL
• “Scalable SQL”, ACM Queue, Michael Rys, April 19, 2011
  http://queue.acm.org/detail.cfm?id=1971597
• “a practical guide to noSQL”, Posted by Denise Miura on March 17, 2011 at