Advanced Data Management

Thomas Heinis
Some Data Management History...

The Database
- Investments
- Accounts
- Customers

What about...
- Reliability?
- Security?
- Consistency?
- Response time?
- Scalability?
Relational Databases

- Structured, schema-based organization of data
- Data decomposed into tables, ex. customer data:

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Dustin</td>
</tr>
<tr>
<td>Jack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
</tr>
<tr>
<td>London</td>
</tr>
<tr>
<td>Leicester</td>
</tr>
</tbody>
</table>

- SQL - General-purpose query language
- Join tables to retrieve all data: e.g., `SELECT * FROM Customer, City WHERE City.City = Customer.City`
- Focused on strong consistency (ACID)
Transactions – Data Integrity

Why Concurrent Access to Data must be managed?
John and Jane withdraw $50 and $100 from a common account...

John:
1. get balance
2. if balance > $50
3. balance = balance - $50
4. update balance

Jane:
1. get balance
2. if balance > $100
3. balance = balance - $100
4. update balance

Initial balance $300. Final balance=?
It depends...

Need to order operations → transactions!
Indexing – Efficient Data Retrieval

• How can we answer the query: “Find the account with the balance of 920.-” efficient?

• One approach is to scan the entire customer table, check every customer, return the one with balance = 920 ... very slow for large databases
Example Index (B-Tree)
NOSQL DATABASES
Database Systems Today

New applications, more requirements!

Rapidly changing hardware, new performance hurdles!

Online Transactions

Web Apps

Scientific

Business Intelligence

Database System
Relational Databases: One Size Does Not Fit All...

New applications challenge relational databases:

1. Strong consistency (ACID) limits scalability
2. Schema evolution is challenging
3. Little optimization for novel hardware
4. Cumbersome language
5. Limited data types
NoSQL

“Not Only SQL” or “Not Relational”.
Six key features:

1. Scale horizontally “simple operations”
2. Replicate/distribute data over many servers
3. Simple call level interface (contrast w/ SQL)
4. Weaker concurrency model than ACID
5. Efficient use of distributed indexes and main memory
6. Flexible schema
Key-value Stores

• Operate on key-value pairs
• Single key to store (or retrieve) data value
• Think “file system” more than “database”
• Consistent hashing (DHT)
• Only primary index: lookup by key
• No secondary indexes
Key-Value Store – Basic Idea

Table T:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>k1</td>
<td>v1</td>
</tr>
<tr>
<td>k2</td>
<td>v2</td>
</tr>
<tr>
<td>k3</td>
<td>v3</td>
</tr>
<tr>
<td>k4</td>
<td>v4</td>
</tr>
</tbody>
</table>

- No query language!
- API with simple operations:
  - lookup(key) \(\rightarrow\) value
  - lookup(key range) \(\rightarrow\) values
  - getNext \(\rightarrow\) value
  - insert(key, value)
  - delete(key)
- Each row has timestamp
- No multi-key transactions
Document Stores

• A "document" = a pointerless object = e.g. JSON = nested or not = schema-less

• In addition to KV stores, may have secondary indexes

• SimpleDB, CouchDB, MongoDB, Terrastore

• Scalability:
  – Replication (e.g. SimpleDB, CouchDB – means entire db is replicated),
  – Sharding (MongoDB);
  – Both
Document Store (MongoDB)

```javascript
> db.user.insert({
    first: "John",
    last: "Doe",
    age: 39
})

> db.user.find ({"first": "John"})
{
    "_id": ObjectId("51..."),
    "first": "John",
    "last": "Doe",
    "age": 39
}

> db.user.update({
    "_id": ObjectId("51...")},
    {
        $set: {
            age: 40,
            salary: 7000
        }
    }
}

> db.user.remove({
    "first": /^J/}
})
```
Scalable Relational Systems (NewSQL)

- Means relational databases that offer sharding

- **Key difference to NoSQL:**
  - NoSQL difficult or impossible to perform large-scope operations and transactions
  - NewSQL systems do not **preclude** these operations, but users pay a price only when they need them.

- MySQL Cluster, VoltDB, Clusterix, ScaleDB, Megastore (the new BigTable)

- Many more **NewSQL** systems coming online...
Scalable Data Processing

• Parallel execution achieves greater efficiency
• But, parallel programming is hard
  - Parallelization
  - Fault Tolerance
  - Data Distribution
  - Load Balancing
MapReduce (Hadoop and others)

• “MapReduce is a programming model and an associated implementation for processing and generating large data sets”

• Programming model
  – Abstractions to express simple computations

• Library
  – Takes care of the gory stuff: Parallelization, Fault Tolerance, Data Distribution and Load Balancing
Programming Model

• To generate a set of output key-value pairs from a set of input key-value pairs
  \[ \{ <k_i, v_i> \} \rightarrow \{ <k_o, v_o> \} \]

• Expressed using two abstractions:
  - Map task
    \[ <k_i, v_i> \rightarrow \{ <k_{int}, v_{int}> \} \]
  - Reduce task
    \[ <k_{int}, \{v_{int}\}> \rightarrow <k_o, v_o> \]

• Library
  - aggregates all the all intermediate values associated with the same intermediate key
  - passes the intermediate key-value pairs to \textit{reduce} function
MapReduce Architecture
NoSQL Systems

• Key Value Stores

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW3 3TB</td>
<td>London</td>
</tr>
<tr>
<td>B18 4BJ</td>
<td>Birmingham</td>
</tr>
</tbody>
</table>

• Document Stores

• Scalable SQL Systems

• Data Processing Systems
NoSQL Challenges

E.g., Flexible Schemas in MongoDB

```javascript
db.inventory.insert(
    {
        category: "vacuum",
        details: {
            model: "14Q3",
            manufacturer: "XYZ Company"
        },
        stock: [ { size: "S", qty: 25 }],
        category: "clothing"
    }
)
```

```javascript
db.inventory.insert(
    {
        category: "vacuum",
        details: {
            model: "14Q3",
            manufacturer: "XYZ Company"
        },
        color: "blue"
    }
)
```

What fields does `db.user.find ({"category": "vacuum"})` have?
Data Management Landscape*

"Structuredness"

very

not all

Query/Analysis Frequency

low

high

* according to me

* ORACLE

* VOLTDB

* MySQL

* cassandra

* mongoDB

* CouchDB

* Hadoop