Distributed Non-Relational Databases

Pelle Jakovits

Tartu, 7 December 2018
Outline

• Relational model
• NoSQL Movement
• Non-relational data models
  – Key-value
  – Document-oriented
  – Column family
  – Graph
• Non-relational database examples
• When Relational vs Non-Relational
The Relational Model

• Data is stored in tables
  – Different entity types in different tables

• Strict relationships between tables
  – Foreign key references between columns

• The expected format of the data is specified with a restrictive schema
  – Fixed data types
  – Allow/deny null values

• Data is typically accessed using Structured Query Language (SQL)
Database Scaling

• Vertical scaling – scale the server
• Horizontal scaling – across a cluster of server
• Relational model does not scale well horizontally
• There are too many dependencies in relational model
• Database sharding is one approach for scaling relational databases
Sharding

https://www.mysql.com/products/cluster/scalability.html
Dependencies in relational databases

The NoSQL Movement

• Driven by the rise of „Big Data“ and Cloud Computing
• NoSQL is a broad term with no clear boundaries
  – The term NoSQL itself is very misleading
  – NoSQL -> NonSQL -> Non Relational
• Emergence of persistence solutions using non-relational data models
• Non-relational data models are based on **Key-Value** structure
• Simpler schema-less key-value based data models scale better than the relational model
• Tradeoff between data consistency and high availability
CAP Theorem

- It is impossible for a distributed computer system to simultaneously guarantee all three of the following:
  - **Consistency** - every read receives the most recent write or an error
  - **Availability** - every request receives a proper response
  - **Partition/Fault tolerance** - the system continues to operate despite arbitrary partitioning (network failures, dropped packets)
- Choice is usually between **consistency** or **availability** when failures happen.
- NoSQL solutions which are more focused on **availability** try to achieve **eventual consistency**
- When trying to aim for both **consistency** or **availability**
  - Will have high latency
CAP Theorem

Availability

Consistency

Partition Tolerance

MySQL, Postgre, Oracle SQL

Dynamo, Cassandra, Riak

BigTable, HyperTable, HBase, MongoDB, BerkeleyDB, Redis
Non-relational Data Models
Aggregate-orientedness

• Non-relational data models aim to store data that needs to be aggregated together
• Aggregate is a collection of data that is treated as a unit
  – E.g. a customer and all his orders
• In normalized relational databases aggregates are computed using GroupBy operations
• Goal is to de-normalize data so minimal (or no) grouping and joining is needed!
• Keyed aggregates make for a natural unit of data sharding
Indexing and Partitioning

• In Key-value model, Key acts as an index
• Secondary indexes can be created in some solutions
  – Implementations vary from partitioning to local distributed indexes
• Data is partitioned between different machines in the cluster
  – Similar to sharding
• Usually data is partitioned by rows or/and column families
• Users can often specify partitioning parameters
  – Gives control over how data is distributed
  – Very important for optimizing query speed
The Key-value Model

- Data stored as key-value pairs
- The value is an opaque blob to the database
- Examples: Dynamo, Riak, Apache Ignite, ArangoDB, Berkeley DB, Couchbase, FoundationDB, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL, OrientDB, Redis, SciDB, ZooKeeper
Benefits of the Key-value Model

• Horizontal scalability
  – Data with the same Key stored close to each other
  – Suitable for cloud computing
• Flexible schema-less models suitable for unstructured data
• Fetching data by key can be very fast
The Document-oriented Model

• Data is also stored as key-value pairs
  – Value is a „document“ and has further structure
• No strict schema
  – Expectation is that documents contain schema
  – Not enforced in any way
• Query data based on document structure
• Examples: CouchDB, MongoDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, IBM Domino, MarkLogic, OrientDB, RethinkDB
Example JSON document

{  
  "name":"Asheville Veedub Volkswagon Repair & Restoration",  
  "address":"851 Charlotte Hwy",  
  "attributes":{    
    "BusinessAcceptsCreditCards":"True",    
    "ByAppointmentOnly":"False",    
    "GoodForKids":"True"  
  },  
  "business_id":"0KwutFa520HgPLWtFv02EQ",  
  "categories":"Auto Repair, Automotive",  
  "city":"Fairview",  
  "is_open":1,  
  "latitude":35.5431561,  
  "longitude":-82.4419299,  
  "neighborhood":",  
  "postal_code":"28730",  
  "review_count":7,  
  "stars":4.5,  
  "state":"NC"  
}  

• Aggregates are described in JSON using map and array data structures
The Column Family Model

• Data stored in large sparse tabular structures
• Columns are grouped into column families
  – Column family is a meaningful group of columns
  – Similar concept as a table in relational database
• A record can be thought of as a two-level map
  – Column Family -> Column -> Value
• New columns can be added at any time
• Examples: BigTable, Cassandra, HBase, Accumulo
### Column Family Example

<table>
<thead>
<tr>
<th>_id</th>
<th>names</th>
<th>contacts</th>
<th>messages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>username</td>
<td>firstname</td>
<td>lastname</td>
</tr>
<tr>
<td>a001</td>
<td>jsmith01</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>b014</td>
<td>pauljones</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Names**
  - username: jsmith
  - firstname: John
  - lastname: Smith

- **Contacts**
  - phone: 555 001
  - email: jsmith@example.com

- **Messages**
  - item1: Message1
  - item2: Message2
  - ...: ...
  - itemN: MessageN

- **Names**
  - username: pauljones

- **Contacts**
  - item1: new Message
Graph Databases

- Data stored as nodes and edges of a graph
- The nodes and edges can have fields and values
- Aimed at graph traversal queries in connected data
- Examples: Neo4J, FlockDB, AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph

Non Relational Database Examples
Riak

• Key-value model
• Distributed decentralized persistent hash table
  – Keyspace is partitioned between nodes
• Consistent hashing
  – Avoid hash-to-node remapping when the number of nodes changes
• Eventual Consistency using Multi Version Concurrency Control (MVCC)
  – Do not overwrite data item, create a newer version
Riak

• RESTful query interface
  – Basic PUT, GET, POST, and DELETE functions

• Links and link walking
  – One-way relationships between data objects
  – Turns Key-Value store into a simple Graph database

• Higher level querying on-top of Key-Value structure:
  – Turns Riak into Key->Document database
  – Riak search is based on Apache Solr search engine
  – MapReduce in Erlang and JavaScript
Query example

- **HTTP**
  
curl -v http://127.0.0.1:8098/buckets/user/keys/pellejakovits

- **Python API**

  ```python
  client = riak.RiakClient()
  user_bucket = client.bucket('user')
  johndoe = user_bucket.get('pellejakovits')
  johndoe_dict = johndoe.get_data()
  ```
MongoDB

- Document oriented (BSON)
- Query language based on JavaScript
- GridFS for BLOB file storage
- Master-slave architecture
- Linking between documents
- Supports MapReduce in JavaScript
Query example

db.inventory.find({
    status: "A",
    $or: [ {qty: { $lt: 30 }}, { item: /^p/? } ]
})

• Matches the following SQL query:
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
CouchDB

- Document oriented (JSON)
- RESTful query interface
- Built in web server
- Web based query front-end Futon
CouchDB

• MapReduce is the primary query method
  – JavaScript and Erlang
• Materialized views as results of incremental MapReduce jobs
• CouchApps – JavaScript heavy web applications
  – Built entirely inside CouchDB
  – Without a separate web server for the logic layer
Query example

Documents:

{ "id": 2, "position": "programmer", "first_name": "James", "salary": 100 }
{ "id": 7, "position": "support", "first_name": "John", "salary": 23 }

Let's extract average salary for each unique position

Map:

function(doc, meta) {
  if (doc.position && doc.salary) { emit(doc.position, doc.salary); }
}

Reduce:

function(key, values, rereduce) {
  return sum(values)/values.length;
}

Some Reduce functions have been pre-defined: _sum(), _count(), _stats()
Cassandra

• High scalability and high availability
• Column family model
• Uses SQL-Like Cassandra Query Language (CQL)

• Powered By Cassandra:
  – Apple's: 75,000 nodes storing over 10 PB of data
  – Netflix: 2,500 nodes, 420 TB, 1 trillion requests per day
Cassandra

• Static column family
  – Fixed number and types of columns
  – Rows are not required to have values

• Dynamic column family
  – No fixed column names or types
  – Each row can contain different number of columns
  – Columns can be added on the fly

• Super-column families
  – contains super columns that contains columns

• Composite columns
  – Combination of columns labels separated by colon (:) as a column label
  – Like using combined Keys from MapReduce
Cassandra query example

```
SELECT race_name, point_id, lat_long
AS CITY_LATITUDE_LONGITUDE
FROM cycling.route;
```

<table>
<thead>
<tr>
<th>race_name</th>
<th>point_id</th>
<th>city_latitude_longitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>47th Tour du Pays de Vaud</td>
<td>1</td>
<td>('Onnens', (46.8444, 6.6667))</td>
</tr>
<tr>
<td>47th Tour du Pays de Vaud</td>
<td>2</td>
<td>('Champagne', (46.8333, 6.65))</td>
</tr>
<tr>
<td>47th Tour du Pays de Vaud</td>
<td>3</td>
<td>('Novalle', (46.8333, 6.6))</td>
</tr>
<tr>
<td>47th Tour du Pays de Vaud</td>
<td>4</td>
<td>('Vuiteboeuf', (46.8, 6.55))</td>
</tr>
<tr>
<td>47th Tour du Pays de Vaud</td>
<td>5</td>
<td>('Baulmes', (46.7833, 6.5333))</td>
</tr>
<tr>
<td>47th Tour du Pays de Vaud</td>
<td>6</td>
<td>('Les Clées', (46.7222, 6.5222))</td>
</tr>
</tbody>
</table>

Neo4J

• Open source NoSQL Graph Database
• Uses the Cypher Query Language for querying graph data
• Graph consists of Nodes, Edges and Attributes
Neo4J Query example

MATCH (you {name:"You"})
MATCH (expert)-[:WORKED_WITH]-> (db:Database {name:"Neo4j"})
MATCH path = shortestPath(
  (you)-[:FRIEND*..5]-(expert)
)
RETURN db, expert, path

https://neo4j.com/developer/cypher-query-language/
Conclusions

• There has been a rise in non-relational (NoSQL) data stores
• This is related to the rise of cloud computing
  – Horizontal scaling is cheap
  – key-value models offer better scalability
• The NoSQL landscape is extremely varied
  – The key-value model
  – The document-oriented model
  – The column family model
  – Graph databases
• In reality the classification of databases into different models is not straight forward
• Non-relational databases are aggregation oriented
  – Avoid joins, keep logically grouped data together
Why use Relational databases?

• Highly available and highly consistent
  - Old, tried and tested technologies
  - Well optimized for performance
• Strict schema means less mistakes
  - Easier to control the quality of data
  - Easier to avoid missing data, wrong data types, etc.
• Very good performance as long as data fits into a single machine
When use Non-Relational?

• When volume of Data grows too large
  – Easier and cheaper to scale
• When the schema is not fixed
  – Store a query unstructured data
  – Prototyping: Build applications without pre-defining schema
  – Updating schemas is very simple
• When using structured documents (JSON, XML)
  – Similar schema inside documents, but can vary greatly
  – Nested sub-structures
• Many open source options
That is All

• Next week's practice session
  – Using NoSQL databases: CouchDB

• Next Lecture
  – Distributed Machine learning
  – Distributed data processing with R