NoSQL Systems

Lipeng WANG, Qiaomu SHEN
April 22, 2016
Outline

• What is NoSQL databases
• Why NoSQL databases
• Basic Concepts
• Data models
• Distribution
• Mainstream NoSQL Database
• References
What is NoSQL Databases

“A NoSQL (originally referring to ‘non SQL’ or ‘non relational’) database provides a mechanism for storage and retrieval of data which is modeled in means other than in the tabular relations in relational databases”

-Wikipedia
What does NoSQL mean and how do you categorize these databases?

Common observations of NoSQL:

• Not using the relational model
• Running well on clusters
• Mostly open-source
• Built for the 21st century web estates
• Schema-less
NoSQL databases can broadly be categorized in four types:

- Key-Value databases
- Document databases
- Column family stores
- Graph databases
Key-Value databases

Database Examples:

- Riak
- Redis
- Memcached
- Berkeley DB
- upscaledb
- Amazon DynamoDB
- Couchbase
Document databases

Database Examples:

- MongoDB
- CouchDB
- Terrastore
- OrientDB
- RavenDB
Column family stores

Database Examples:

- BigTable
- Hbase
- Hypertable
Graph databases

Database Examples:

- Neo4J
- Infinite Graph
- OrientDB
- FlockDB
Advantages of NoSQL Databases over Relational Databases

• The Growth of Big Data
• Continuous Data Availability
• Real Location Independence
• Modern Transactional Capabilities
• Flexible Data Models
• Better Architecture
• Analytics and Business Intelligence
• Relational databases were born in the era of mainframes and business applications – long before the Internet, Cloud, big data and mobile
• RDMS were engineered to run on a single server – the bigger, the better
• NoSQL databases emerged as a result of the exponential growth of the Internet and the rise of web applications
• Develop with *agility* and to operate at *any scale*
Flexibility for Faster Development

**RDBMS** – An explicit schema prevents the addition of new attributes on demand
Flexibility for Faster Development

**JSON** – The data model evolves as new attributes are added on demand
Simplicity for Easier Development

RDBMS – Applications “shred” objects into rows of data stored in multiple tables
**NoSQL VS RDBMS**

Simplicity for Easier Development

<table>
<thead>
<tr>
<th>Shane</th>
<th>Johnson</th>
<th>Big Data</th>
<th>Product Marketing</th>
<th>Couchbase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shane</td>
<td>Johnson</td>
<td>Big Data</td>
<td>Technical Marketing</td>
<td>Red Hat</td>
</tr>
<tr>
<td>Shane</td>
<td>Johnson</td>
<td>Java</td>
<td>Product Marketing</td>
<td>Couchbase</td>
</tr>
<tr>
<td>Shane</td>
<td>Johnson</td>
<td>Java</td>
<td>Technical Marketing</td>
<td>Red Hat</td>
</tr>
<tr>
<td>Shane</td>
<td>Johnson</td>
<td>NoSQL</td>
<td>Product Marketing</td>
<td>Couchbase</td>
</tr>
<tr>
<td>Shane</td>
<td>Johnson</td>
<td>NoSQL</td>
<td>Technical Marketing</td>
<td>Red Hat</td>
</tr>
</tbody>
</table>

*RDBMS – Queries return duplicate data, applications have to filter it out*
Simplicity for Easier Development

JSON – Applications can store objects with nested data as single documents
Elasticity for Performance at Scale

**NoSQL VS RDBMS**

*RDBMS*—The server is too big or too small, leading to unnecessary costs or poor performance.
NoSQL VS RDBMS

Elasticity for Performance at Scale

NoSQL – Add commodity server on demand so the hardware resources match the application load
Availability for Always-on, Global Deployment

NoSQL VS RDBMS

RDBMS—Requires separate software to replicate data to other data centers
Availability for Always-on, Global Deployment

NoSQL – Replication between data centers is fully built-in and can be bi-directional
Basic Concepts

The CAP Theorem:

• Consistency
  All nodes see the same data at the same time.

• Availability
  A guarantee that every request receives a response about whether it succeeded or failed.

• Partition Tolerance
  The system continues to operate despite arbitrary partitioning due to network failures.
The CAP Theorem

Combinations of CA, CP and AP:

• CA
  Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.

• CP
  Some data may not be accessible, but the rest is still consistent/accurate.

• AP
  System is still available under partitioning, but some of the data returned may be inaccurate.
The CAP Theorem

Availability

Consistency

Partition Tolerance
Ippolite summarizes the BASE properties in the following way:

An application works basically all the time (basically available), does not have to be consistent all the time (soft-state) but will be in some known-state eventually (eventual consistency) [Ipp09].
The CAP:
- Consistency
- Availability
- Partition tolerance

• The BASE:
- Basically available
- Soft-state
- Eventual consistency
ACID and CAP

The relationship between CAP and ACID is more complex and often misunderstood, in part because the C and A in ACID represent different concepts than the same letters in CAP and in part because choosing availability affects only some of the ACID guarantees.
Data Models

- Key-value database
- Document database
- Column-family database
- Graph database

Aggregate data models

Aggregate ignorant models
Data Models

- Key-value database
- Document database
- Column-family database
- Graph database

Aggregate data models

Aggregate ignorant models
Aggregate Data Model

- **Aggregate**: a collection of related objects that we wish to treat as a unit for data manipulation, management and consistency [Evans03]

- Aggregate oriented database:
  - Each data unit is more complex: list, map or hierarchical structure.
  - Aggregates match up with in-memory data structure very well.
  - The update of an aggregate is an atomic operation.
  - Good for using in distributed storage systems (all the related data are aggregated together)
Data Models

Aggregate Data Model  Example: e-commerce website. [Nosql Distilled]

The database needs to store the following information

- Customer
- Billing Address
- Address
- Order
- Order Payment
- Order Item
- Product

If using relational database, each of the information will be stored in a table. And there is no data repeat.

Relational
Data Models

Aggregate Data Model  Example: e-commerce website. [Nosql Distilled]

The database needs to store the following information

- Customer
- Billing Address
- Address
- Order
- Order Payment
- Order Item
- Product

If using a relational database, each of the information will be stored in a table. And there is no data repeat.
Data Models

Aggregate Data Model

The database needs to store the following information:

- Customer
- Billing Address
- Address
- Order
- Order Payment
- Order Item
- Product

If using a relational database, each of the information will be stored in a table. And there is no data repeat.
Data Models

Aggregate Data Model

```javascript
//Customer
{
    "id": 1,
    "name": "Martin",
    "billingAddress": [{
        "city": "HongKong",
    },{
        "city": "Washington"
    }]
}

//orders
{
    "id": 99,
    "customerId": 1,
    "orderItems": [{
        "productId": 27,
        "price": 32.45,
        "productName": "NoSQL Distilled"
    }],
    "shippingAddress": [{
        "city": "Washington"
    }],
    "orderPayment": [{
        "ccinfo": "1000-1000-1000-1000",
        "txnId": "abelif879rfc",
        "billingAddress": {
            "city": "Washington"
        }
    }]
}
```
Data Models

Aggregate Data Model

Customer -- Order

```json
//Customer
{
  "id": 1,
  "name": "Martin",
  "billingAddress": [{
    "city": "HongKong",
  }, {
    "city": "Washington"
  }],
  "shippingAddress": [{
    "city": "Washington"
  }],
  "orderPayment": [{
    "ccinfo": "1000-1000-1000-1000",
    "txnId": "abelif879rFr",
    "billingAddress": {
      "city": "Washington"
    }
  }]
}
```

```json
//orders
{
  "id": 99,
  "customerId": 1,
  "orderItems": [{
    "productId": 27,
    "price": 32.45,
    "productName": "NoSQL Distilled"
  }, {
    "city": "Washington"
  }],
  "shippingAddress": [{
    "city": "Washington"
  }],
  "orderPayment": [{
    "ccinfo": "1000-1000-1000-1000",
    "txnId": "abelif879rFr",
    "billingAddress": {
      "city": "Washington"
    }
  }]
}
```

Link between two aggregates
When using nosql database, we want to minimize the number of aggregates we access during a data interaction.
Data Models

Aggregate Data Model

Customer only

Put all the orders into the customer aggregate.

```javascript
//Customers
{
  "customer": {
    "id": 1,
    "name": "Martin",
    "billingAddress": {
      "city": "Washington"
    },
    "orders": [
      {
        "id": 99,
        "customerId": 1,
        "orderItems": [
          {
            "productID": 27,
            "price": 32.45,
            "productName": "NoSQL Distilled"
          }
        ],
        "shippingAddress": {
          "city": "Washington"
        },
        "orderPayment": {
          "coinfo": "1000-1000-1000-1000",
          "txnId": "abelif879rft",
          "billingAddress": {
            "city": "Washington"
          }
        }
      }
    }
  }
}
```
Data Models

Aggregate Data Model

An aggregate structure may help with some data interactions but be an obstacle for others.

Aggregate

Reviews orders for a specific customer.

For each customer, we only need to visit one aggregate.

Analysis all the orders of the last year.

We need visit all the aggregates to get the orders in a specific time interval.
Data Models

Aggregate Data Model

Key-Value

Key 1 → Value 1
Key 2 → Value 2
Key 3 → Value 3
Key 4 → Value 4

Aggregate

Document

```javascript
//orders
{
    "id": 99,
    "customerId": 1,
    "orderItems": [{
        "productId": 27,
        "price": 32.45,
        "productName": "NoSQL Distilled"
    },
    "shippingAddress": {
        "city": "Washington"
    },
    "orderPayment": {
        "ccInfo": "1000-1000-1000-1000",
        "txnId": "abelif879rf",
        "billingAddress": {
            "city": "Washington"
        }
    }
}
```

Column-family

RKey 1 → Value 1
RKey 2 → Value 2
CKey 1 → CValue 1
CKey 2 → CValue 2
CKey 3 → CValue 3
Distribution

Nosql makes the distribution of data on large cluster easier. Because all the related data is contained in one aggregate.

Two solutions:

- **Sharding**: put different data on different node.
  - Master-slave
  - Peer-to-peer
    - Copy the same data over different nodes

Example: A database with four sets of data is deployed on a single server

Write

A
B
C
D

Read
Distribution

Nosql makes the distribution of data on large cluster easier. Because all the related data is contained in one aggregate.

Two solutions:

- **Sharding**: put different data on different nodes.

- **Replication**
  - Master-slave
  - Peer-to-peer

Copy the same data over different nodes.
Distribution

Nosql makes the distribution of data on large cluster easier. Because all the related data is contained in one aggregate.

Two solutions:

**Replication**
- Copy the same data over different nodes

**Sharding**
- Put different data on different nodes

How to assign the data to the node? **Hashing?**
Consistent hashing:
- Do not need to rehash all the data
- Balance, Load, Smoothness, Balance, Monotonicity
Consistent hashing:

- Do not need to rehash all the data
- Balance, Load, Smoothness, Balance, Monotonicity
Distribution

Consistent hashing:
- Do not need to rehash all the data
- Balance, Load, Smoothness, Balance, Monotonicity
Consistent hashing:

- Do not need to rehash all the data
- Balance, Load, Smoothness, Balance, Monotonicity
Consistent hashing:
- Do not need to rehash all the data
- Balance, Load, Smoothness, Balance, Monotonicity

Amazon's Dynamo
Nosql makes the distribution of data on large cluster easier. Because all the related data is contained in one aggregate.

Two solutions:

- **Sharding**: put different data on different nodes.
  - Master-slave
  - Peer-to-peer
    - Copy the same data over different nodes

- **Replication**
  - Write
  - Read
Distribution

Nosql makes the distribution of data on large cluster easier. Because all the related data is contained in one aggregate.

**Two solutions:**

- **Sharding:** put different data on different nodes.
- **Replication**
  - Peer-to-peer
  - Copy the same data over different nodes

**Master-slave:** one master which is node responsible for writing and reading, the slave nodes are responsible for reading only.
Nosql makes the distribution of data on large cluster easier. Because all the related data is contained in one aggregate.

Two solutions:

**Sharding**: put different data on different nodes.

**Replication**

- **Master-slave**: replicate data across multiple nodes
- **Peer-to-peer**: replicate the same data over different nodes
Distribution

Consistency in peer to peer

• Vector clocks (node, counter)
Distribution

Consistency in peer to peer

- Vector clocks (node, counter)
- Quorum NRW
  - W: the number of nodes participating in writing
  - R: the number of nodes participating in read
  - N: the number of nodes involved in replication
  - **Write Quorum**: \( W > \frac{N}{2} \)
  - **Read Quorum**: \( R + W > N \)
Distribution

Consistency in peer to peer

• Vector clocks (node, counter)
• Quorum NRW
• Gossip Protocol
Nosql makes the distribution of data on large cluster easier. Because all the related data is contained in one aggregate.

**Two solutions:**

- **Sharding**: put different data on different nodes.
- **Replication**: Copy the same data over different nodes.

**Combine replication and sharding**

- **Master-slave**
- **Peer-to-peer**
  - Copy the same data over different nodes
Mainstream NoSQL Database

**Key-value**
- Riak
- Redis
- Memcached DB
- Berkeley DB
- HamsterDb
- DynamoDB

**Document**
- DocumentDB
- CouchDB
- MongoDB
- PostgreSQL
- OrientDB
- Informix
Mainstream NoSQL Database

**Column-family**
- Hbase
- Hypertable
- Cassandra
- Accumulo
- Parquet

**Graph Database**
- Neo4j
- AllegroGraph
- Teradata Aster
- GraphBase
- InfiniteGraph
Mainstream NoSQL Database

Dynamo DB (key-value)

Compared with the relational database

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>Dynamo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Bucket</td>
</tr>
<tr>
<td>Row</td>
<td>Key-value</td>
</tr>
<tr>
<td>Row_id</td>
<td>Key</td>
</tr>
</tbody>
</table>

Basic Features

• The simplest nosql model
• The store is a hash table, the users access database through primary key
• Each value is an aggregate and could be any kinds of data structure
• Operations: get, put and delete
## Dynamo DB (key-value)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Technique</th>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning</td>
<td>Consistent Hashing</td>
<td>Incremental Scalability</td>
</tr>
<tr>
<td>High Availability for writes</td>
<td>Vector clocks with reconciliation during reads</td>
<td>Version size is decoupled from update rates.</td>
</tr>
<tr>
<td>Handling temporary failures</td>
<td>Sloppy Quorum</td>
<td>Provides high availability and durability guarantee when some of the replicas are not available.</td>
</tr>
<tr>
<td>Recovering from permanent failures</td>
<td>Anti-entropy using Merkle trees</td>
<td>Synchronizes divergent replicas in the background.</td>
</tr>
<tr>
<td>Membership and failure detection</td>
<td>Gossip-based membership protocol and failure detection.</td>
<td>Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.</td>
</tr>
</tbody>
</table>
Mainstream NoSQL Database

Dynamo DB (key-value)

- Consistency
- Availability
- Partition tolerance

DynamoDB
Mainstream NoSQL Database

**Basic Features**

- The storage unit (aggregate) is document (JSON, BSON)
- Self-describing
- Hierarchical structure
- Flexible schema

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Collection</td>
</tr>
<tr>
<td>Row</td>
<td>Document</td>
</tr>
<tr>
<td>Column</td>
<td>Field</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td>Join</td>
<td>Embedded Document</td>
</tr>
<tr>
<td>Foreign Key</td>
<td>Reference</td>
</tr>
</tbody>
</table>
Mainstream NoSQL Database

**MongoDB (document)**

*Operations:*

**CRUD**
- Create
  - `db.collection.insert(<document>)`
- Read
  - `db.collection.find(<query>, <projection>)`
  - `db.collection.findOne(<query>, <projection>)`
- Update
  - `db.collection.update(<query>, <update>, <options>)`
- Delete
  - `db.collection.remove(<query>, <justOne>)`

**Index**
- `db.collection.createIndex({ id: 1 })`
Mainstream NoSQL Database

MongoDB (document):

And

Sharding
Mainstream NoSQL Database

MongoDB (document):

Consistency

Availability

Partition tolerance
Mainstream NoSQL Database

MongoDB (document)

Two scenarios to consider

- Heavy write (sharding based)
- Heavy read (master-slave based)
Cassandra(column-family)

**Compared with the relational database**

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>Cassandra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Keyspace</td>
</tr>
<tr>
<td>Row</td>
<td>Row</td>
</tr>
<tr>
<td>Table</td>
<td>Column-family</td>
</tr>
<tr>
<td>Column</td>
<td>Column</td>
</tr>
</tbody>
</table>

**Basic Features**

- Each row has an unique key(row-key)
- Each row consists of several columns and each column has name, value and a time stamp
- A collection of similar rows(often used together) makes up a column family.
- Each row doesn’t not have to contains the same columns.
- The row key could be some complex data structure.
Mainstream NoSQL Database

Cassandra(column-family)

**Query**

- A SQL like language: **CQL**

  ```sql
  SELECT name, address from Customer
  ```

  ```sql
  SELECT * name, address from Customer
  ```

  ```sql
  INSERT INTO Customer(KEY, name, address, gender)
  VALUES('51044420', 'Cobi', 'Los Angeles', 'male')
  ```
Mainstream NoSQL Database

Cassandra(column-family)

Query

- A SQL like language: **CQL**
  
  \[
  \text{SELECT name, address from Customer}
  \]
  
  \[
  \text{SELECT * name, address from Customer}
  \]
  
  \[
  \text{INSERT INTO Customer(K VALUES('51044420', 'C}
  \]

Scaling

- Peer to peer and sharding
- Consistent hashing
Thank you and any questions?
References

• https://en.wikipedia.org/wiki/NoSQL
• http://www.couchbase.com/nosql-resources/what-is-no-sql
• https://www.thoughtworks.com/insights/blog/nosql-databases-overview
• http://www.planetcassandra.org/what-is-nosql/
• http://webpages.uncc.edu/xwu/5160/nosqldbs.pdf
• https://en.wikipedia.org/wiki/CAP_theorem
• http://www.w3resource.com/mongodb/nosql.php