Architecting Big Data Ecosystems

An overview of NOSQL Systems and Apache Spark

Dr Asma ZGOLLI

Senior Machine Learning Engineer

Guest Lecturer at SRH University of Applied Sciences Hamburg

Invited by:

Prof Dr Nnamdi Oguji

8 November 2023

Big data ecosystem

- Applications:
  - Demands and sales forecast
  - Predictive maintenance
  - Fraud and anomaly detection
  - Efficient data management

- Data:
  - Orders and payment
  - Product catalogues and clients profile
  - Smart meters, network, and sensors
  - Weather and geospatial information

✔High volume, variety, veracity, and velocity

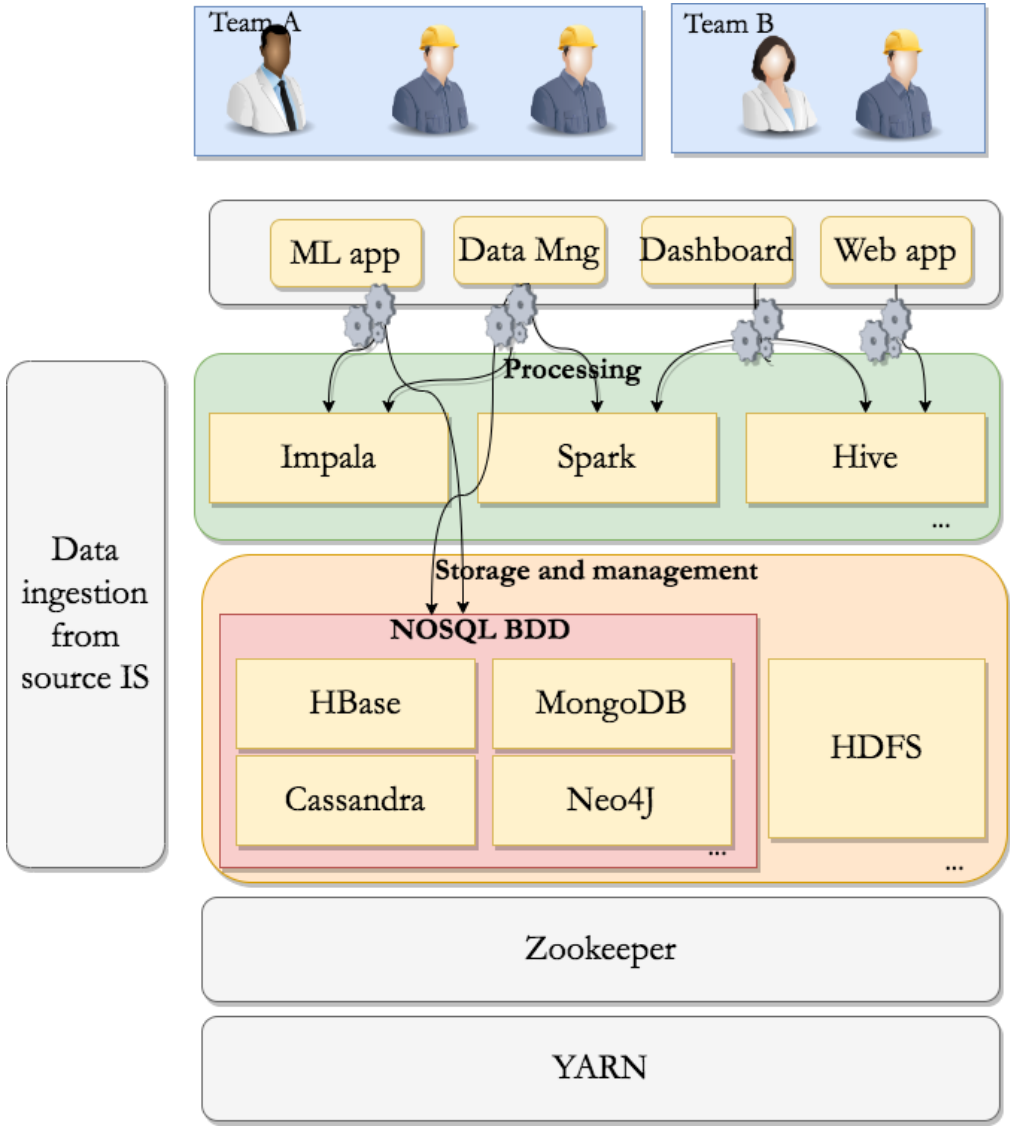

Figure 1: Example of a Big Data Architecture

Roadmap

Big Data Ecosystems

- Introduction

NOSQL Datastores

- NOSQL
- NOSQL Data Models
- Key-Value Datastores
- Document Datastores
- Column-Oriented Datastores
- Graph Datastores
- Comparing NoSQL Data Models: A Snapshot

Apache Spark

- Spark Internals
- Spark APIs

NOSQL

- NOSQL: **N**ot **O**nly **SQL**
- ***Scalability:*** *horizontal scalability on commodity hardware using distribution techniques.*
- ***Performance vs Transactions:*** NOSQL prioritizes performance over transactional semantics.
- ***Complex Queries:*** *In addition to to key-only access, NOSQL supports richer queries e.g.*
  - Multi-property indexing and sophisticated data retrieval. (MongoDB)
  - Column-based filtering. (BigTable)
- ***Big Data Integration***
  - Bindings to Hadoop or other parallel and distributed processing frameworks, such as MapReduce.
- ***Schema-Free Storage***
  - *Supports semi/unstructured data*
  - *(+)* Faster application development and less costly on-the-fly schema updates
  - *(-)* More responsibility on application developers and the need for versioning

NOSQL Data Models

Data model	Data structure	Relation	Record	Attribute	Data view
Document	Hierarchical tree e.g. : map , collection	Collection	Document	Attribute	Physical view
Key-value	Blob , text , json , xml	Bucket	Key-value pair	X	Physical view
Column family	Tabular data	Column family	Row	Column	Logical view
Graph	Graph , tree	Node	Node	Attribute	Logical view
		Link	link	Attribute	

Key-Value Datastores …

- **Physical Data Model:** *< key , value >* pairs.
- **Key**
  - Hash key
- **Value**
  - Blob
  - Complex, non-typed: Text, structured, multivalued (array), or atomic
- **Key-Based Data Access:**
  - Supported operators:
    - put(key, value);
    - value= get(key);
    - delete(key);
  - Very efficient data access.
  - Limited API
    - Increased complexity related to data processing at the application level.
- **Use Cases :** Sessions management, data indexing

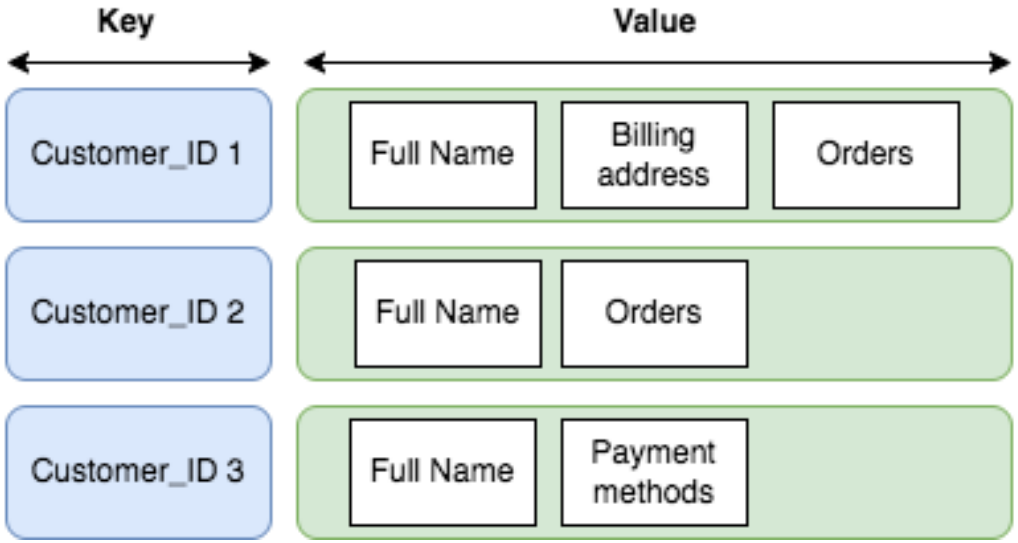

Figure 2: Example of a key value pairs collection

… Key-Value Datastores

Case study: Dynamo

- A pure key-value store [G23,DHJK+07]
  - Scalable
  - Decentralized (Peer-to-peer architecture)
  - Highly available
- Data partitioning & Replication:
  - Consistent hashing : Keys are stored at the next node in the ring with a higher ID.
  - Virtual Nodes: Serve as an abstract physical nodes for enhanced fault tolerance and load balancing.
- Amazon Use Cases:
  - Powers critical services: shopping carts, bestseller lists, sales rankings.
  - Distinct from DynamoDB, which is offered by AWS.

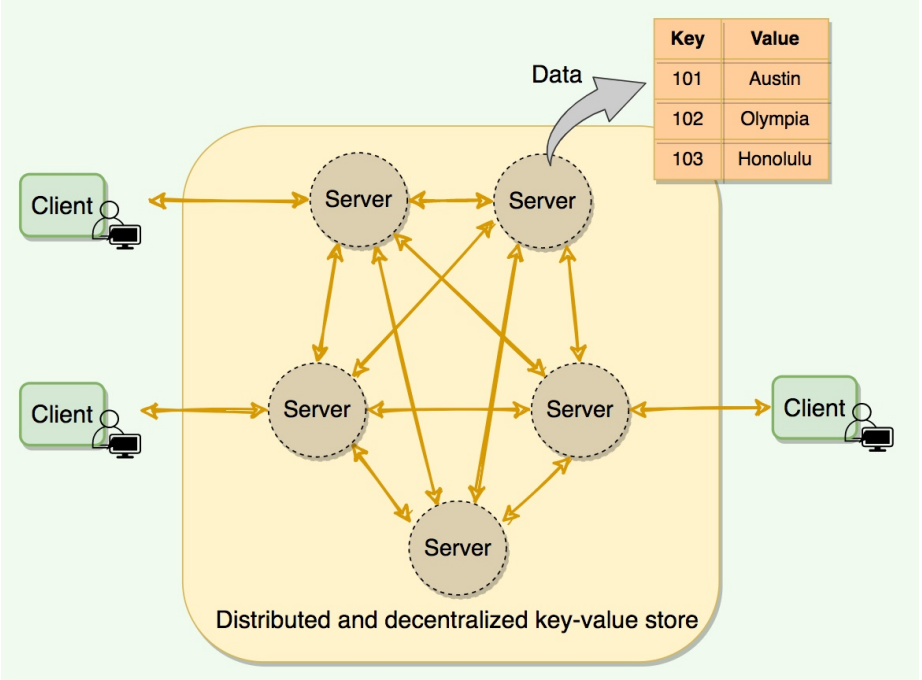

Figure 3: key-value data store distribution model [G23]

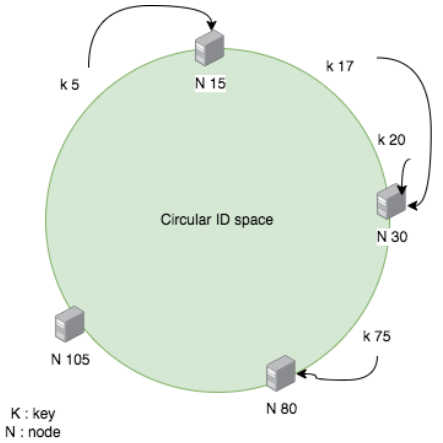

Figure 4: Consistent Hashing

Column-Oriented Datastores …

- **Physical Data Model:**
  - Data is stored by column
- **Logical Data Model:**
  - Big table
- **Column-Family:**
  - A group of columns stored together in the same server optimizing data retrieval
- **Column:**
  - Basic data storage unit
  - atomic (single data type) or structured (nested data).
- **Column-Oriented Data access:**
  - Reads only the **relevant columns** which is suitable for read-intensive data access.
  - Writing data requires significant **IO**.
  - Supports operations such as filtering and aggregation, but does **not support joins** or **subqueries**.

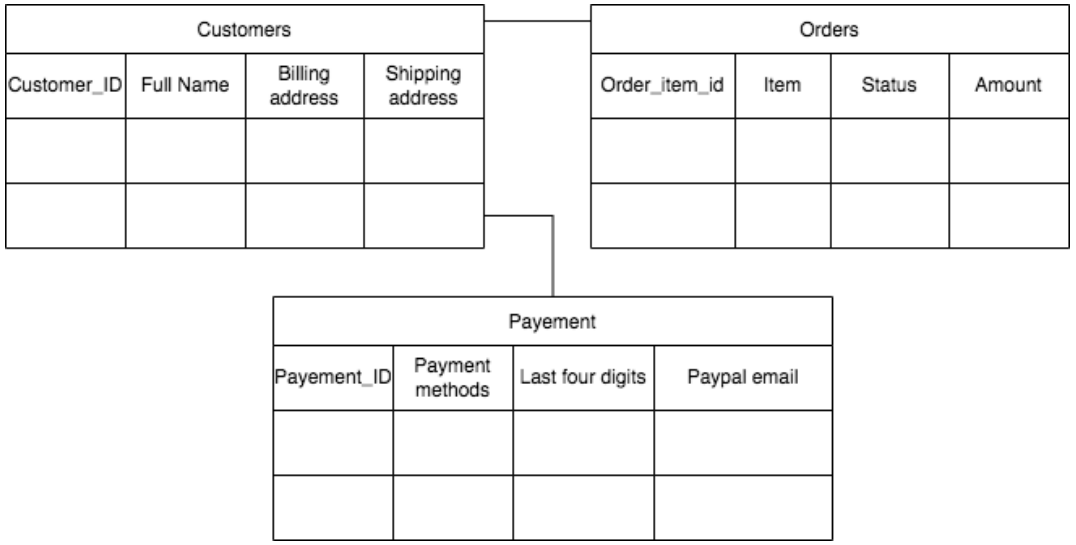

Figure 5: Example of tables (Relational data model)

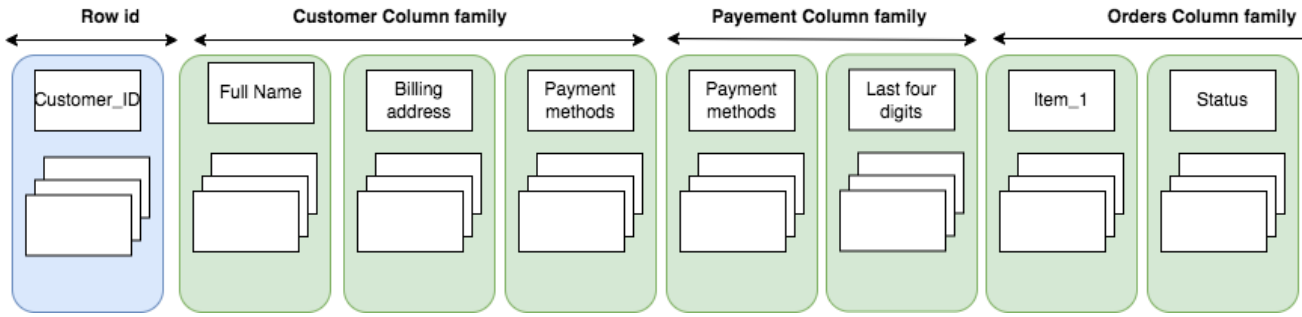

Figure 6: Example of a column-family

Column-Oriented Datastores …

Case study: Hbase

- Hybrid Data Model :
  - key-value + column oriented.
- Foundation: Built on top of of the Hadoop Distributed File System (HDFS).
- Distributed Architecture:
  - Replication : e.g. master-slave, p2p clusters, etc.
  - Partitioning : auto-sharding.
  - Parallelism : Write-Ahead Logging (WAL) and Bloom filters.
- System components [Tea16]:
  - HMaster: Load balancing, catalogue management, and orchestration.
  - RegionServer: Manages data partitions (Regions)
  - HFile: Physical storage format.
  - MemStore: In-memory write buffer.

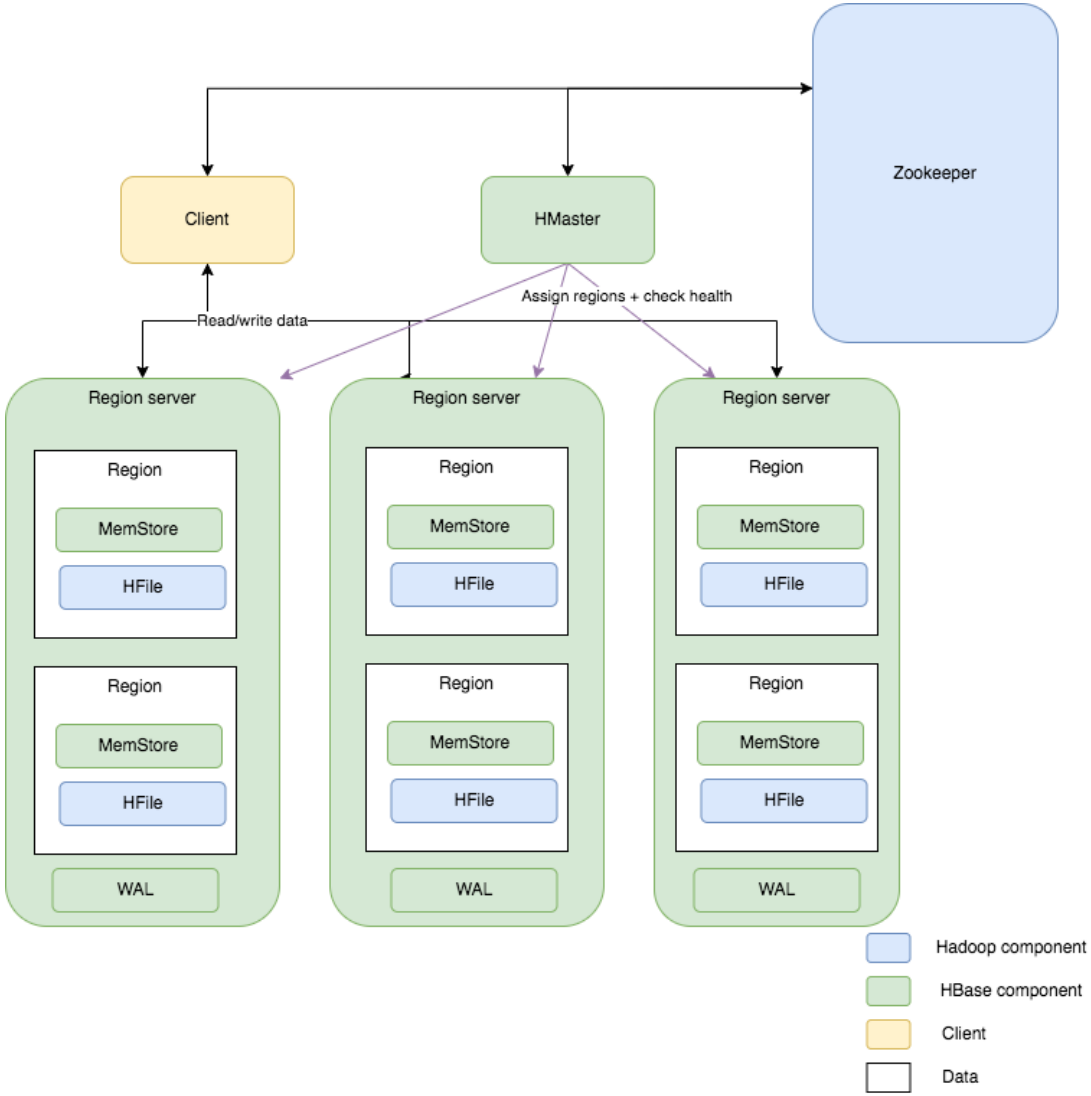

Figure 7: Apache Hbase Architecture

Document-Oriented Datastores …

- **Physical Data Model:**
  - JSON like
  - Flexible schema
  - Hierarchical
  - Stores of complex data in fields called **aggregates**.
- **Rich API:**
  - Advanced data access & filtering
  - Built-in aggregation functions
  - No joins
- **Use cases:**
  - Products catalogues
  - Clients profiles
  - Content management systems (e.g. CRM)

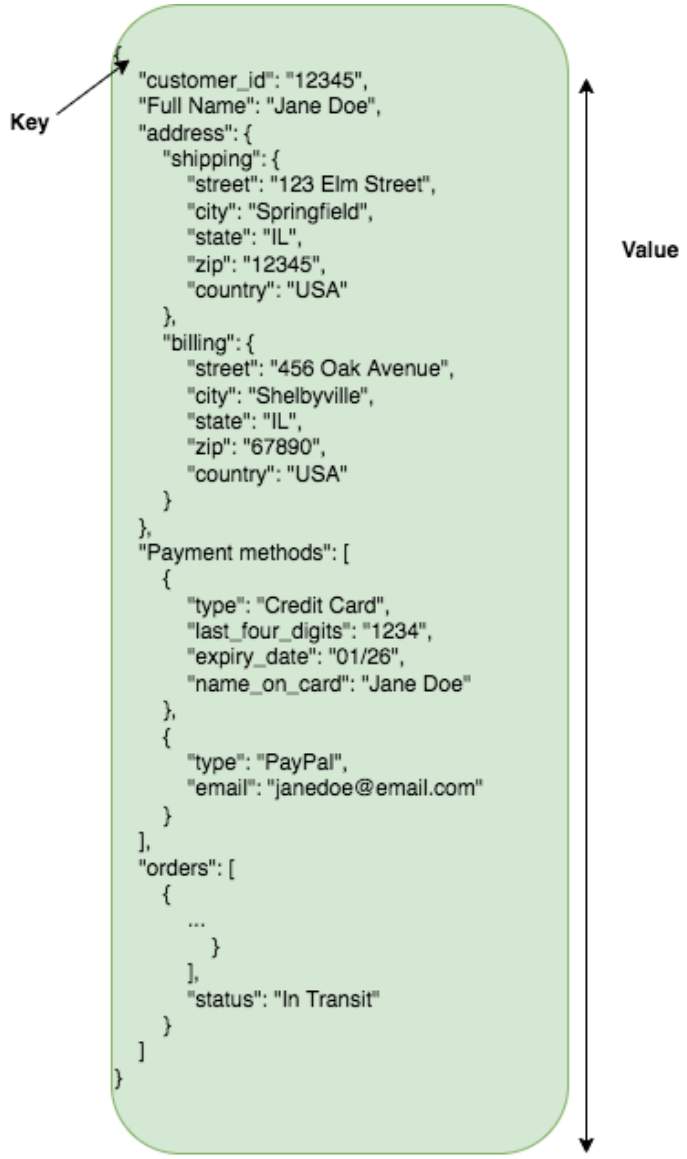

Figure 8: Example of a document

… Document-Oriented Datastores

Case study: Mongodb

- Data model:
  - BSON (Binary JSON)
- Documents:
  - Typed fields
  - Arrays and nested documents
- Data partitioning & Replication:
  - Master / slave replication
    - Relica set
    - Automatic failover
  - Partitioning : sharding
- Indexing:
  - Utilizes a B-tree data structure.

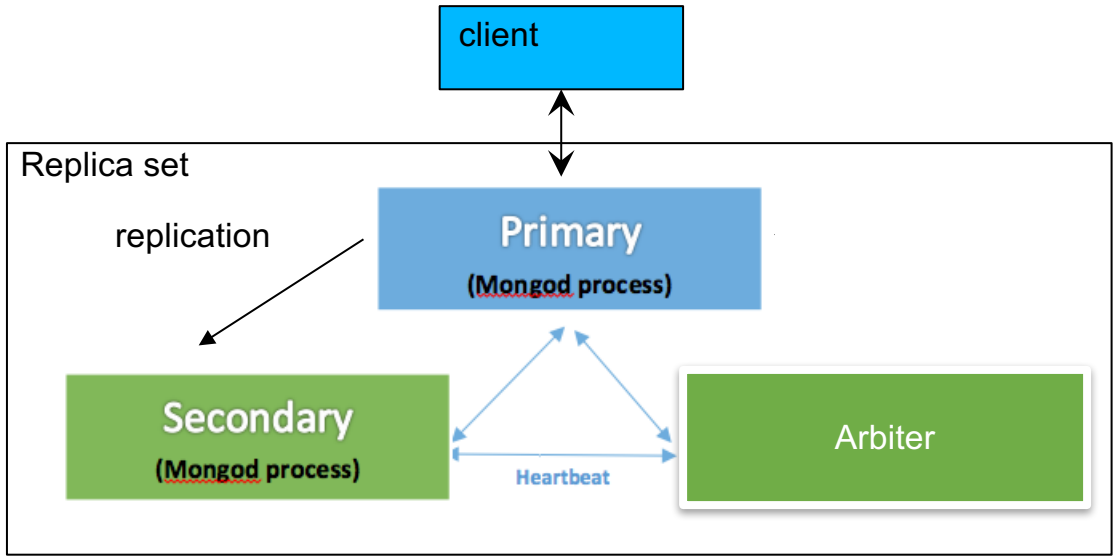

Figure 9: MongoDB replication architecture

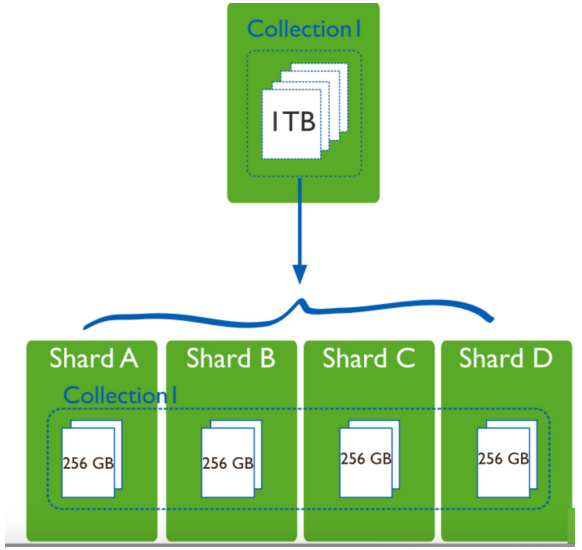

Figure 10: MongoDB partitioning architecture [Mon23]

Graph Datastores …

- Graphs are data structures that help modelling complex relationships between different entities (real-world objects).
- **Nodes:**
  - Also called vertices
  - Represents the entities (real-world objects)
- **Links:**
  - Also called edges
  - represents the relationship between these nodes
- **Physical data model:**
  - Nodes and links typically stored separately.
- **Use cases:**
  - Search and pathfinding in a graph , recommender system, social media data management

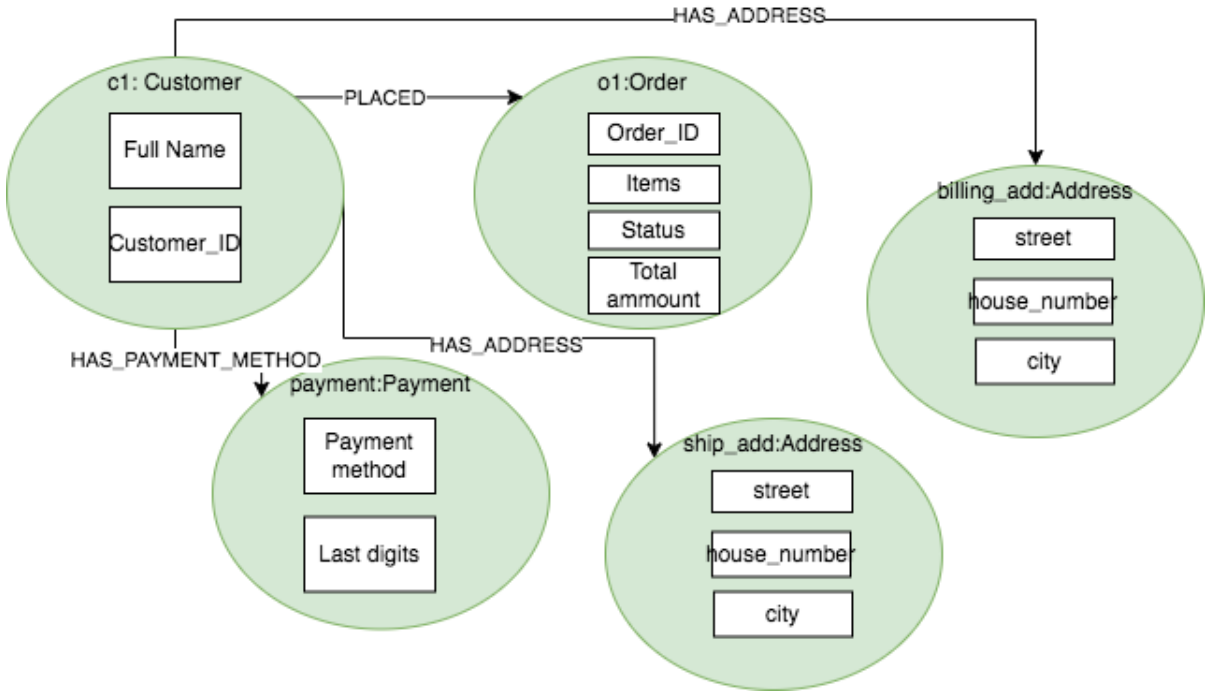

Figure 11: Example of a graph

… Graph Datastores

Case study: Tigergraph

- Data model:
  - Vertices and Edges with attribute support.
  - Attributes can be atomic or structured.
- **GSQL:**
  - Graph-specific, SQL-like declarative language.
  - Supports graph pattern matching.
- **Scalability:**
  - Horizontally scalable architecture.
- **Processing:**
  - Massively Parallel Processing (MPP).

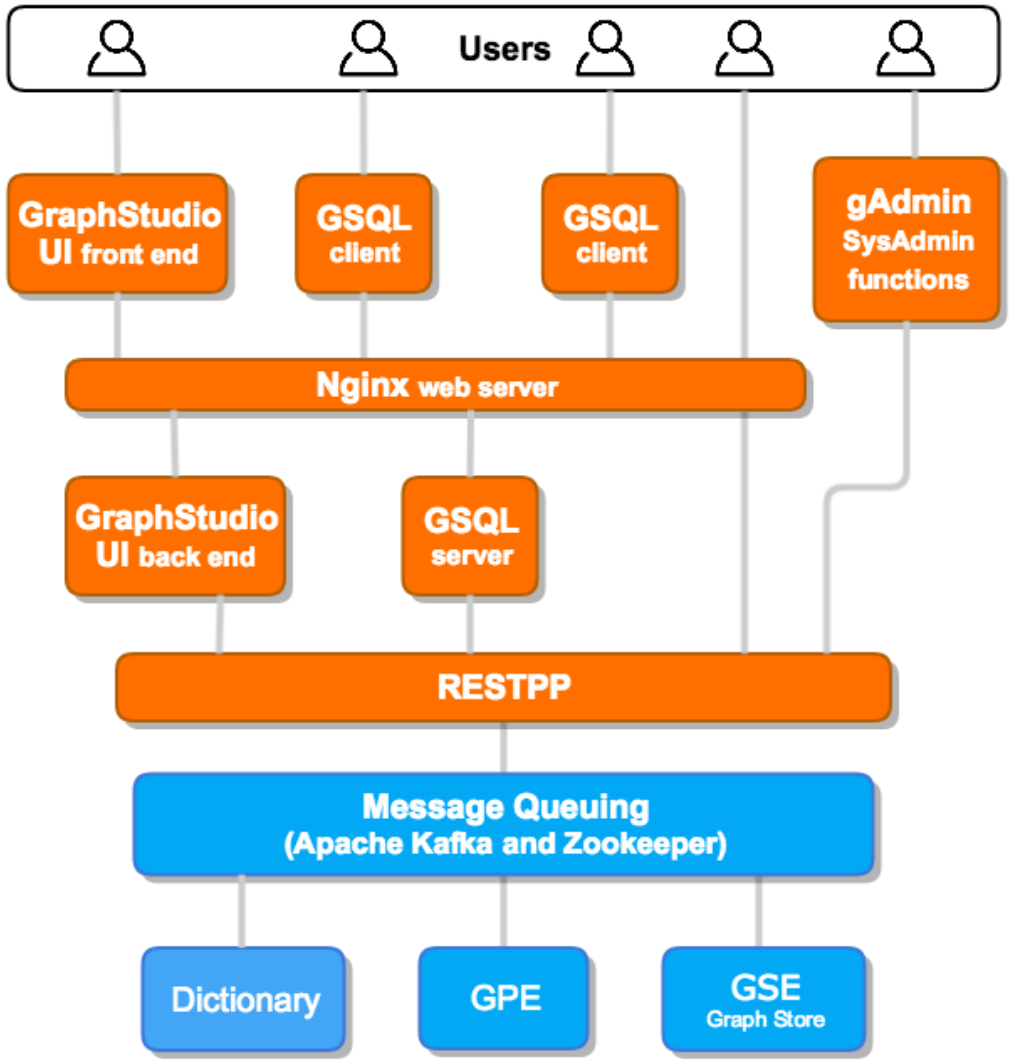

Figure 12: Tigergraph architecture [Tig23]

Comparing NoSQL Data Models: A Snapshot [ZBV21] …

Data model	Principle	Advantage	Disadvantage
Key-value	Data stored and accessed as a key-value pair: Key a unique identifier and Complex data as value	✔ Efficient index on data ✔ Simple data model	Limited access API
Document oriented	Aggregates stored as documents	✔ Complex queries ✔ Rich filters	Slow writes, lack of consistency
Column family oriented	Data organised as columns. Columns store aggregates and are organised as column families	✔ Evolving data schema ✔ Efficient column oriented storage	Slow writes, bad performance with small data and row based access
Graph oriented	Representing the connexions between data records	✔Graph algorithms for efficient connected data querying	Does not support well distribution

… Comparing NoSQL Data Models: A Snapshot [ZBV21]

Operator / data model	**Key value** (e.g Dynamo)	**Document based** (e.g MongoDB)	**Column family** (e.g Apache HBase)	**Graph based** (e.g Neo4J)
Scan	XXX	X	XX	X
Filter	X	XX	X	XX
Join	-	X	-	XXX
Aggregate	-	XXX	X	XX
Group by	-	XXX	X	XX
Order by	-	X	X	X

The number of x denotes the performance of the operator (x, xx, xxx)

Roadmap

Big Data Ecosystems

- Introduction

NOSQL Datastores

- NOSQL
- NOSQL Data Models
- Key-Value Datastores
- Document Datastores
- Column-Oriented Datastores
- Graph Datastores
- Comparing NoSQL Data Models: A Snapshot

Apache Spark

- Spark Internals
- Spark APIs

Spark Internals

- **Apache Spark**
  - **Cluster computing** framework **integrated** in the Hadoop **ecosystem**:
    - Horizontally scalable data processing
    - Parallel data processing
  - Characteristics :
    - **In memory processing**: it addresses the limitation of MapReduce that stores intermediate results on disk after every task by using a caching technique
    - **Distributed processing**: a **master** node receives the workload from the driver program and coordinates the processing by sends it to the **worker** nodes for execution.
    - **Lazy evaluation**: tasks are classified as actions and transformations and structured as a **DAG**. The processing is only triggered when an action is called.

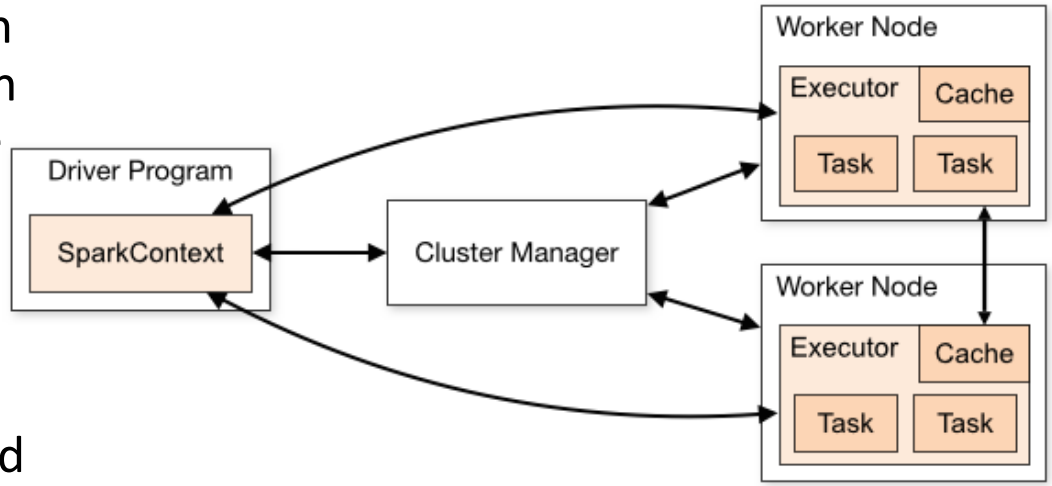

Figure 13: Apache Spark Architecture [Spa]

Spark APIs …

- **Spark RDD API**
  - RDDs are a collection of immutable (in-memory read-only) objects that are distributed across the cluster
  - Low level API: RDDs are the physical data model of Spark, other APIs are built on top of it to facilitate the development in spark

- Example:

```
# Read the input file and Calculating words count
text_file = sc.textFile("firstprogram.txt")
counts = text_file.flatMap(lambda line: line.split(" ")) \
                                        .map(lambda word: (word, 1)) \
                                        .reduceByKey(lambda x, y: x + y)
# Printing each word with its respective count
output = counts.collect()
for (word, count) in output:
    print("%s: %i" % (word, count))
```

… Spark APIs

- **Spark DataFrame**
  - Collection of immutable and distributed data with named columns
  - Similar data model to a relational table
  - Has a Domain Specific Language API that allows querying and transforming the dataframes
- **SparkSQL**
  - Interface that allows executing an SQL-like query language on dataframes
  - Supports query optimization (Catalyst optimizer)
- **Data Science libraries**
  - MLlib: scalable and simple ML library that is integrated with spark APIs
  - Graphx: scalable and parallel processing for graphs

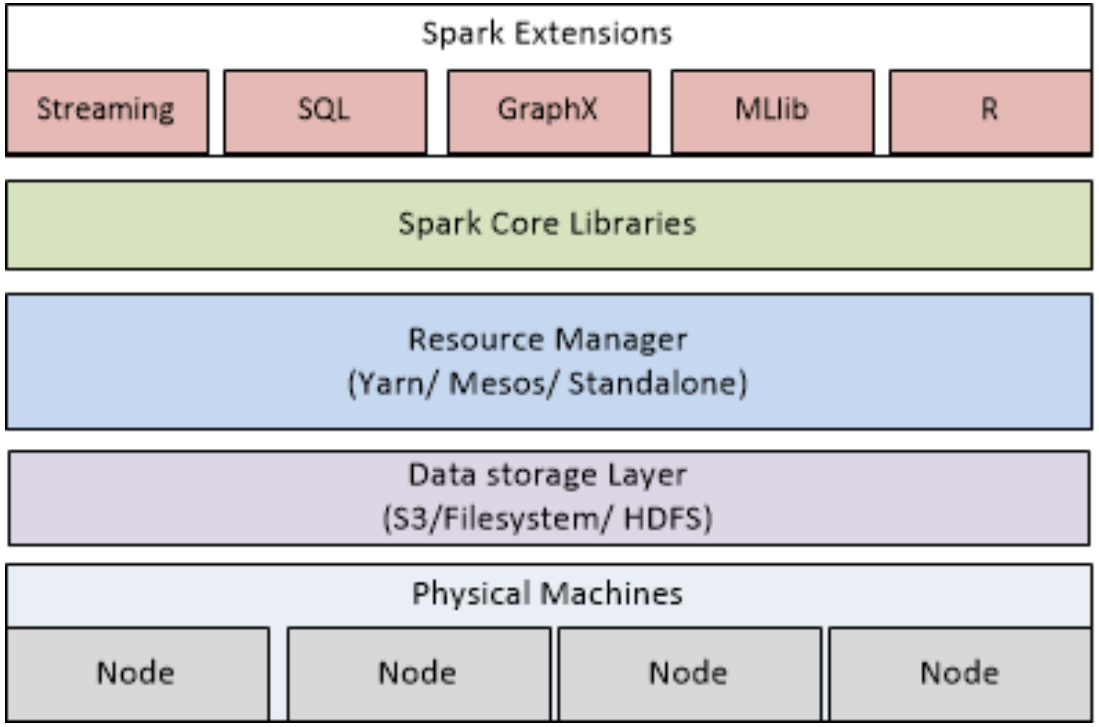

Figure 14: Apache Spark Components [SG17]

Thank you for your attention

Bibliography…

- [G23] Design Gurus. (n.d.). Dynamo: Introduction. Design Gurus: One-Stop Portal For Tech Interviews. Retrieved November 2, 2023, from https://www.designgurus.io/course-play/grokking-the-advanced-system-design-interview/doc/636ce5b668ee620a5aa631d2
- [DHJK+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. 41, 6 (December 2007), 205–220. https://doi.org/10.1145/1323293.1294281
- [Mon23] MongoDB. MongoDB documentation. Available at https://docs.MongoDB.com, Accessed: 2023-10-10, 2023.
- [SF12] Pramod J. Sadalage and Martin Fowler. 2012. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence (1st. ed.). Addison-Wesley Professional.
- [SG17] Saxena, Shilpi, and Saurabh Gupta. 2017. Practical Real-time Data Processing and Analytics. Packt Publishing.

… Bibliography

- [Spa]Team Spark. Cluster Mode Overview - Spark 3.1.1 Documentation. Available at https://spark.Apache.org/docs/latest/cluster-overview.html, Accessed: 2023-10-10.
- [Tea16] Apache HBase Team. Hbase™ reference guide. Available at https://hbase.Apache.org/1.2/book.html, Accessed: 2023-10-10,2016.
- [Tig23] TigerGraph. TigerGraph docs. Available at https://docs.tigergraph.com/home/, Accessed: 2023-10-10, 2023.
- [ZBV21] Asma Zgolli, Christophe Bobineau, Genoveva Vargas-Solar. 2021. Recommandation de placement de données pour les traitements dans des lacs de données smart grids. [Data placement recommendation for processing in smart grid data lakes]. (Thesis Presentation).