THE UNIVERSITY of EDINBURGH

Advanced Database Systems

Spring 2025

Lecture #24:

NoSQL

1

NoSQL Motivation

Driven by Web 2.0 Applications

- Emergence of massive-scale applications (e.g., Facebook, Amazon, Instagram)
- Need for handling high-volume, real-time data operations (OLTP)
- Load can increase rapidly with web traffic and unpredictably

Scaling transactions across multiple nodes is hard

- Traditional protocols like 2PC are too slow

Consistency is hard to enforce when data is partitioned & replicated

2

Scaling through Partitioning

Partition (shard) data across multiple machines

- Enables data to fit into main memory for faster access
- User queries / transactions are spread across multiple machines

Advantages

- **Higher throughput:** can handle more clients simultaneously
- **Efficient writes:** updates impact only a single data copy

Disadvantages

- **Expensive reads:** retrieving data may need accessing many machines, increasing latency
- **Concurrency challenges:** reads need locks on each machine to handle concurrent writes

3

Scaling through Replication

Create multiple copies (replicas) across machines

- Each database partition is replicated across multiple nodes
- Queries can be distributed among replicas for load balancing

Advantages

- **Better throughput & latency:** clients can query different replicas, reducing latency
- **Improved fault tolerance:** if a machine fails, another replica can serve the request
- **Efficient reads:** multiple replicas make read operations faster and more scalable

Disadvantages

- **Expensive writes:** every write must update all replicas to maintain consistency
- **Potentially stale reads:** If updates aren't synced properly, replicas may serve outdated data

4

NoSQL: "Not Only SQL"

A paradigm shift
- Focus on scalability and performance
- Complements, rather than replaces, RDBMS

Trade-off
- Scalability and performance through horizontal scaling (sharding and replication)
- Sacrifice consistency and complex analytics (OLAP)

Core principles
- **Flexible schema:** Adapts to evolving data needs
- **Simplified data models:** Designed for speed and efficiency
- **Efficient but restricted update operations:** Optimises for high-volume transactions

5

Recap: ACID in Relational DBMS

Atomicity: *All* actions in the txn happen, or *none* happen — *"all or nothing"*

Consistency: If each txn is consistent and the DB *starts* consistent, then it *ends* up consistent — *"it looks correct to me"*

Isolation: Execution of one txn is isolated from that of other txns — *"as if alone"*

Durability: If a txn commits, its effects persist — *"survive failures"*

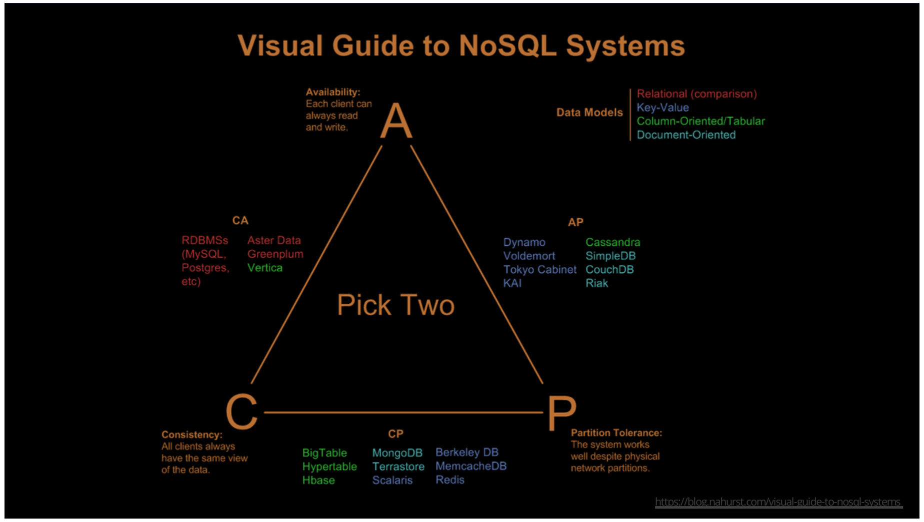
6

CAP Theorem

> *"Of three properties of shared-data systems – data Consistency, system Availability, and tolerance to network Partitions – only two can be achieved at any given moment in time" — Brewer, 1999*

Consistency	All nodes see the same data at the same time
Availability	Guarantee that every request receives a response about whether it was successful or failed
Partition tolerance	System continues to operate despite arbitrary message loss or failure of part of the system

7



8

NoSQL Paradigm: BASE

Basically Available

Guarantees availability even during failures

The system can still respond, though the data might not be fully consistent

Soft State

The state of the system can change over time, even without updates

Replicas may temporarily have different data until synchronised

Eventually Consistent

Data will eventually become consistent across all nodes

No guarantee of immediate consistency, but eventual convergence

Taxonomy of NoSQL Systems

Key-Value Stores

Description: Data is stored as key-value pairs (simple lookup)

Examples: Redis, DynamoDB, Memcached

Use cases: Caching, session storage, simple data storage

Document Stores

Description: Data is stored as documents (often JSON, BSON, or XML)

Examples: MongoDB, CouchDB

Use cases: Content management, e-commerce applications, user profiles

Taxonomy of NoSQL Systems (Cont.)

Column-Family Stores

Description: Data organised into columns and column families

Examples: Cassandra, HBase (open-source implementation of Google's BigTable)

Use cases: Large-scale analytics, time-series data, log processing

Graph Databases

Description: Data is stored as nodes and edges (relationships)

Examples: Neo4j, ArangoDB, Amazon Neptune

Use cases: Social networks, recommendation engines, fraud detection

Key-Value Stores

Data model: `(key,value)` pairs

Key = string/integer, unique for the entire data

Value = can be anything (very complex object)

Operations: `get(key)`, `put(key,value)`

Operations on value not supported

Partitioning & Replication: using hashing

Partitioning: key *k* is stored at server *h(k)*

Multiway replication: e.g., key *k* stored at *h1(k), h2(k), h3(k)*

On update, propagate changes to the other servers (eventual consistency)

Issue: when an app reads one replica, it may be stale

Document Stores

Motivation

In key-value stores, the ***value*** is often a very complex object

Example: `key = '18/05/2024', value = [ all flights that date ]`

Better approach: store the *value* as structure data

Formats like JSON, Protobuf, or XML are commonly used

"Document" is simply structured data

A document database is a collection of documents

Each document can represent a complex data model

JSON: Semi-Structured Data Model

Human-readable data interchange

Text-based, open standard for exchanging data between systems

Core structures

Object: A collection of key-value pairs

Array: An ordered list of values

```json
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```

Data types in JSON

Atomic values: e.g., strings, numbers

Objects: Nested JSON objects

Arrays: A list of values, can include objects or other arrays

Relational Data Model	Semi-Structured Data Model (JSON)
Rigid, Flat Structure: Data is stored in tables	**Flexible, Nested Structure:** Data is organised in trees (objects and arrays)
Fixed Schema: Schema must be defined in advance	**No Predefined Schema:** JSON is "self-describing", allowing flexibility
Binary Representation: Good for performance, bad for exchange	**Text Representation:** Good for exchange, bad for performance
Query Language: Based on Relational Algebra	**Query Language:** NoSQL use their own query languages, RDBMS use SQL with extensions

Summary

NoSQL: Emerged for modern data challenges

Initially perceived as a potential replacement for SQL

Reality: NoSQL and SQL databases now coexist, each excelling in its niche

Modern RDBMSs now support storing and querying JSON data

SQL-based systems remain essential for strong consistency