NoSQL

### SQL & Relational Databases

SQL Databases:

* Depend on a pre-filter.
* Assume monolithic memory.
* Assume a single disk farm.
* Hard to partition.
* ACID properties.

Limitations of Relational Databases:

* Impedance mismatch.
* Application and integration.
* Scale up vs. scale out.

### NoSQL

Why NoSQL:

1. Velocity
2. Variety
3. Volume

NoSQL **scales better** than SQL / relational databases.

NoSQL:

* Non-relational.
* No SQL as query language.
* Schema-less
* Usually open source.
* Distributed (exception of graph DBs).
* BASE properties.

Database goals:

* Data durability.
* Consistent performance.
* Graceful degradation under load.

Big Data workloads require **distributed computing**.

Distribution Models

### Replication

**Replication** takes the same data and copies it over multiple nodes:

|  |
| --- |
| **Master-slave**: One node is the **primary**, responsible for processing the update to data while the other nodes are **secondaries**.   * Scaling by adding slaves / secondaries. * Processing incoming data is limited by the master / primary. * Read resilience. * Inconsistency problem (read). |
| **Peer-to-peer**: All replicas have equal weight and can accept writing.   * Scaling by adding nodes. * Node failure without losing write capability. * Inconsistency problem (write). |

### Sharding

**Sharding** puts different data on different nodes.

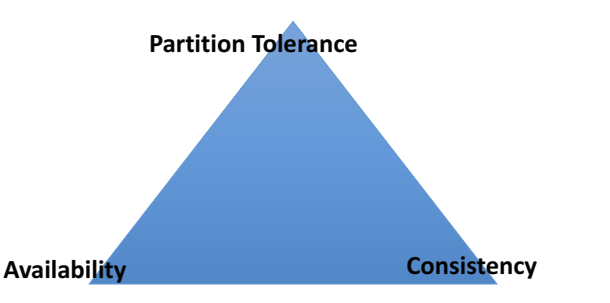
* Each server acts as a **single source** for its subset of the data.
* Offers **horizontal scalability**.
* Access can be based on physical location. Place data to the nearest server.
* Scales read and write on the different nodes of the same **cluster**.
* No resilience if used alone. Node failure ⇒ data unavailability.

Replication and sharding can be used in combination or alone:

|  |
| --- |
| **Master-slave & sharding**: Multiples masters, each data has a single master. |
| **P2P & sharding**:  Common for column-family databases.  Replication of the shard. |

CAP Theorem

|  |  |
| --- | --- |
| Consistency | All nodes see the **same data** at the **same time**. |
| Availability | Every request receives a success or failure **response**. |
| Partition Tolerance | The system **continues to operate** despite arbitrary partitioning due to network failures. |



MongoDB prioritises CP.

|  |  |
| --- | --- |
| **ACID** (SQL) | **BASE** (NoSQL) |
| **Atomic**:  Everything in a transaction succeeds, or the entire transaction is rolled back.  **Consistent**:  A transaction cannot leave the database in an inconsistent state.  **Isolated**:  Transactions cannot interfere with each other.  **Durable**:  Completed transactions persist even when servers restart. | **Basic Availability**:  An application works basically all the time.  **Soft-State**:  It does not have to be consistent all the time.  **Eventual Consistency**:  It will be in some known state eventually.  Each node is **always available** to serve requests. As a trade-off, data modifications are **propagated** in the background to other nodes. The system may be **inconsistent**, but the data is **largely accurate**. |

Data Models

A **data model** is a **representation** we perceive and use to manipulate our data.

It describes how we interact with the data, and how these elements interact with each other.

Common types of NoSQL:

1. Key-value stores *aggregate*
2. Document stores *aggregate*
3. Column stores *aggregate*
4. Graph stores

There are many hybrid approaches.

|  |  |
| --- | --- |
| Key-Value | Maps keys to values.  Values are treated as **blobs**, and can be complex compound objects.  Single index.  **Fast** and **scalable**.  Inefficient for **aggregate queries** e.g. “all the carts worth $100 or more”.  shopping carts, user profiles & preferences, session information. |
| Document | Like a hash: one id → many values.  **JSON**.  Data is implicitly **de-normalised**. |
| Column | Store data as columns rather than row.   * Columns are organised in **column families**. * Each column belongs to a single column family. * A column acts as a unit for access. * Particular column families are accessed together.   Efficient to do column-ordered operations.  Inefficient for row-based queries.  Adding columns is **inexpensive**, done on a row-by-row basis.  Each row can have a different set of columns ⇒ Tables can remain sparse.  Advantages:   |  |  | | --- | --- | | **Relational** | **Column** | | Queries that return small subsets of rows.  Queries that use a large subset of row data. | Queries that just require a column of data.  Queries that require a small subset of row data. | |
| Graph | **Nodes**: Entities.  **Edges**: Relationships between entities.  Querying a graph database involves traversing the graph’s relationships.  Advantages:   * Representing highly interconnected objects. * Cheap traversal of relationships.   Relational databases are not suited for representing relationships - implemented with foreign keys and expensive joins. |

**Hadoop** is a map-reduce framework used to **partition computation** on large datasets for analysis.

### NoSQL vs. Relational Databases

|  |  |
| --- | --- |
| Advantages: | Disadvantages: |
| Flexible schema.  Simple API.  Scalable  Distributed & replicated storage  Cheap | Not ACID compliant.  No standards.  Eventually consistent.  Poor documentation for certain products. |