10. NoSQL Databases

Slides adapted from *Principles of database management*, Lemahieu et al, 2018

Shortcomings of RDBMSs

- Relational databases traditionally emphasize consistency:
  - Guarantees ACID properties.
  - Stable schema that minimizes duplication of data and encodes integrity constraints.
- Focus on consistency may hamper flexibility and scalability:
  - Coordination between multiple sites in a DDBMS, for example.
- NoSQL use case: large databases, need for flexible schema, availability is high-priority.

NoSQL databases

- NoSQL are databases that depart from the relational model.
- NoSQL = not only SQL (because it sometimes still uses SQL).
- Purpose: near-linear horizontal scaling (data distribution) with a focus on increasing performance and availability.
- Key relaxation: consistency → eventual consistency.

Relational vs. NoSQL comparison

Feature	Relational databases	NoSQL databases
Data paradigm	Relational tables	Key-value/tuple based; document based; graph based; others: Column, XML, object, time series, probabilistic, etc.
Distribution	Single-node and distributed	Mainly distributed
Scalability	Challenges re: horizontal scaling	Easy horizontal scaling, easy data replication
Openness	Closed and open source	Mainly open source
Schema role	Schema-driven	Mainly schema-free or flexible schema
Query language	SQL	No or simple querying facilities; special-purpose languages
Transaction mechanism	ACID: Atomicity, Consistency, Isolation, Durability	BASE: Basically Available, Soft state, Eventual consistency
Feature set	Many features (triggers, views, stored procedures, etc.)	Simple API
Data volume	Normal-sized datasets	Huge datasets and/or very frequent read/write requests

Types of NoSQL databases

- Key-value stores
- Key-tuple stores and document stores
- Graph-based databases
- Others

Key-value stores

Key-value stores

- Database stores data as (key, value) pairs.
- Keys are unique.
- Keys are hashed in order to determine where they should be stored in a distributed database setup – easy **sharding**.

Simple hashing

- Distribute keys over n servers using modulus of hash.

Key
wilfried
seppe
bart
jeanne

Hash function

Hash	**Index** (= mod 3 + 1)
01	2
03	1
07	2
08	3

Server #1	Entries
03	(seppe,30)

Server #2	Entries
01	(wilfried,34)
07	(bart,46)

Server #3	Entries
08	(jeanne,19)

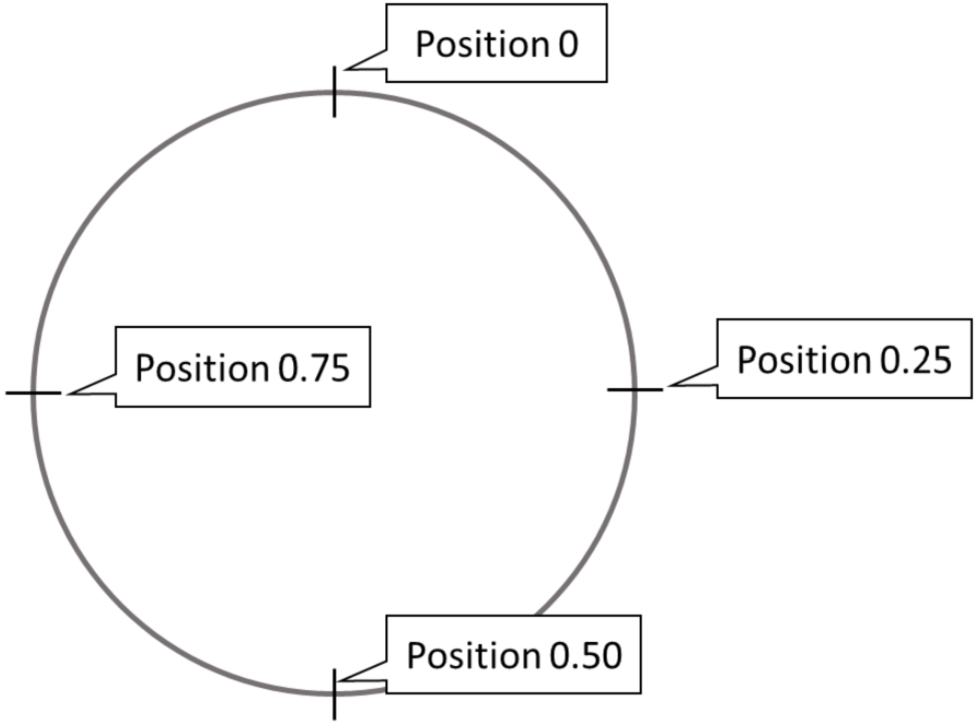
Problem with modulus-based sharding

- Removal or addition of a node results in many keys being reassigned and therefore needing to be moved.

	n		
key	3	2	4
0	0	0	0
1	1	1	1
2	2	0	2
3	0	1	3
4	1	0	0
5	2	1	1
6	0	0	2
7	1	1	3
8	2	0	0
9	0	1	1

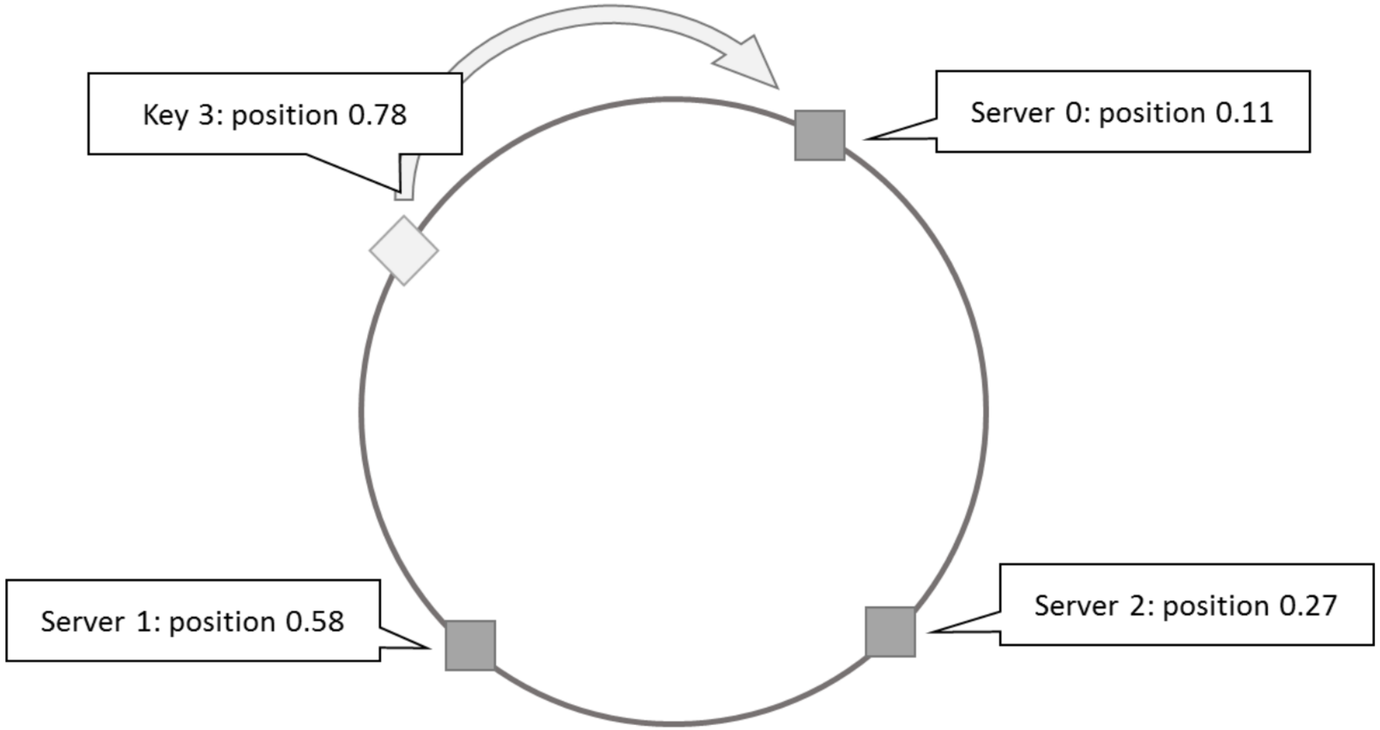
Consistent hashing

- A consistent hashing setup makes use of a ring topology that assigns nodes and keys values in the interval [0, 1).



Consistent hashing: storage

- Data stored at nearest clockwise node.
- Node addition or removal affects roughly 1/n of the values.



Request coordination and membership protocol

- Request coordinator = handles a user request and collates the data needed for a response.
- For many NoSQL databases (Cassandra, Google's BigTable, Amazon's DynamoDB), all nodes in distributed database are able to serve as request coordinator.
  - Requires all nodes to be aware of all other nodes and what is stored on them.
- Need a membership protocol that allows new nodes to inform at least one node of its existence.
  - Information about new/dropped nodes eventually propagate to other nodes.

Eventual consistency

- Membership protocol does not guarantee that every node is aware of every other node at all times.
  - It will reach a consistent state over time.
- State of the network might not be perfectly consistent at any moment in time, though will become eventually consistent at a future point in time.
- Many NoSQL databases guarantee so called **eventual consistency**.

CAP theorem

- Theorem that shows that a distributed system cannot guarantee the following three properties at the same time:
  - Consistency: all nodes see the same data at the same time (all reads return the latest write).
  - Availability: guarantees that every request receives a non-error response (even if the data contained in the response is outdated).
  - Partition tolerance: the system continues to work even if nodes go down.
- Sometimes referred to as “pick two” but in actuality, consistency, availability, and partition tolerance are all non-binary properties that can be traded off against each other.

BASE properties

- NoSQL movement and BASE stake out a position of maximal availability at the cost of consistency.
- The BASE acronym stands for:
  - Basically Available: NoSQL databases adhere to the availability guarantee of the CAP theorem.
  - Soft state: the system can change over time, even without receiving input.
  - Eventual consistency: the system will become consistent over time.

Key-value stores can be a database or a memory cache

- Multiple ways to use a key-value store:
  - As a full database (all data stored this way).
  - As a memory cache sitting in front of a database.

Features of key-value stores

- Key-value stores by themselves offer limited querying and integrity functionality.
- Querying features: mainly basic functions like put and set.
- Limited to no ability to enforce structural constraints.
  - DBMS remains agnostic to the internal structure.
- No relationships, referential integrity constraints, or database schema can be defined.

Tuple and document stores

Tuple stores

- Like a key-value store except it stores a key plus a vector of data (like a “row” in a table).
  - But no requirement for tuples to have the same length or semantic ordering (still schema-less).
  - However, some implementations allow user to organize entries into semantical groups (“collections” or “tables”).

Document stores

- The value of a key-value pair is semi-structured data.
  - JSON is currently the most popular format.

```
{
    "title": "Harry Potter",
    "authors": ["J.K. Rowling", "R.J. Kowling"],
    "price": 32.00,
    "genres": ["fantasy"],
    "dimensions": {
        "width": 8.5,
        "height": 11.0,
        "depth": 0.5
    },
    "pages": 234,
    "in_publication": true,
    "subtitle": null
}
```

Querying and filtering: challenges

- Semi-structured data may make specifying filters and queries more difficult (e.g. may require programming).
- Queries can still be slow because every filter (such as “author.last_name = Baesens“) entails a complete collection or table scan.
  - Can use indexes to speed up queries.
- Joining tables is difficult without a schema.
  - Requires data duplication or manual joining by user.

Querying and aggregation with MapReduce

- Originally developed by Google but has since become genericized and given open-source implementation in Apache Hadoop.
- Allows creating a pipeline of work to be performed on a database that allows the work to be easily parallelized.
- Map phase that extracts data for later processing and reduce phase that performs a function on extracted data.

MapReduce example: map

- Task: Get a summed count of pages for books per genre
- Create a list of input key-value pairs:

k1	v1
1	{genre: education, nrPages: 120}
2	{genre: thriller, nrPages: 100}
3	{genre: fantasy, nrPages: 20}
…	…

- Map function is a simple conversion to a genre-nrPages key-value pair:
  - ```
    function map(k1, v1)
        emit output record (v1.genre, v1.nrPages)
    end function
    ```
    ```

MapReduce example: intermediate result and reduction

- Workers have produced the following three output lists, with the keys corresponding to genres

Worker 1	
k2	v2
education	120
thriller	100
fantasy	20

Worker 2	
k2	v2
drama	500
education	200

Worker 3	
k2	v2
education	20
fantasy	10

- A working operation will be started per unique key k2, for which its associated list of values will be reduced
  - E.g., (education, [120, 200, 20]) will be reduced to its sum, 340.

- ```
  function reduce(k2, v2_list)
      emit output record (k2, sum(v2_list))
  end function
  ```
  ```

MapReduce example: final result

- Final output looks like:

k2	v3
education	340
thriller	100
drama	500
fantasy	30

- Can optionally be sorted based on k2 or v3

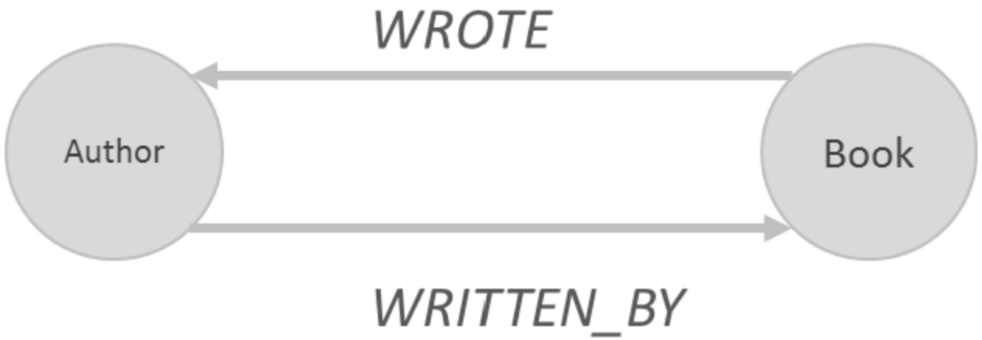
Graph-based databases

Graph-based databases

- Data structure involves nodes and edges (relationships between nodes).
- Unlike previous models which largely remove relationship information, graph-based databases are very “relationshippy”.
  - One-to-one, one-to-many, and many-to-many structures can easily be modeled in a graph
- SQL query to return all book titles for books written by a particular author would look like follows:
  - SELECT title
    FROM books, authors, books_authors
    WHERE authors.id = books_authors.author_id
    AND books.id = books_authors.book_id
    AND authors.name = "Bart Baesens"

Query language: Cypher

- Cypher: query language used by Neo4j (largest graph DBMS vendor).
- Query to return all book titles for books written by a particular author:
  - MATCH (b:Book)<-[:WRITTEN_BY]-(a:Author)
    WHERE a.name = "Bart Baesens“
    RETURN b.title



Cypher overview

- Nodes represented by parentheses () representing a circle.
  - Can also be labelled and/or filtered based on their type, e.g. (b:Book).
- Edges drawn using -- (undirected edge) or -> (directed edge).
  - Can be filtered using square brackets, e.g. [:WRITTEN_BY]

Uses for graph-based databases

- Location-based services
- Recommender systems
- Social media
- Knowledge-based systems

Other NoSQL databases

- Column-based databases: vertically-fragmented databases specializing in calculations/aggregations performed over entire columns.
- XML databases.
- OO databases.
- Database systems to deal with time series and streaming events.
- Database systems to store and query geospatial data.
- Database systems such as BayesDB which let users query the probable implication of their data.

NewSQL

- NoSQL vendors starting to focus again on robustness and durability.
- RDBMS vendors start implementing features to build schema-free, scalable data stores .
  - Focusing on horizontal scalability and distributed querying (DDBMSs).
  - Dropping schema requirements.
  - Support for nested data types or allowing storing JSON / semi-structured data directly in tables.
  - Support for map–reduce operations.
  - Support for special data types, such as geospatial data.
- Result: NewSQL: blend the scalable performance and flexibility of NoSQL systems with the robustness guarantees of a traditional RDBMS.