Jahong Liu

System Design Basics 15: NoSQL

NoSQL

NoSQL stands for “Not Only SQL”. These databases are fundamentally different from SQL databases – they don’t organize data in tables and hence, you can’t use SQL syntax for complex queries and joins. They are much more flexible and scalable than SQL databases and were designed to overcome the limitations of SQL databases, which have certain constraints in terms of scalability and performance. SQL databases can be scaled vertically, but there are limitations to this method, as we have discussed before.

NoSQL databases are specifically engineere to handle large-scale data and high-speed wordloads. They can be scaled horizontally, which is significant advantage when managing large applications. We will delve into the specifics of these benefits later in the article. For now, let’s review the different types of NoSQL databases.

Key – Value Databases

A key-value database uses a simple key-value method to store data. Just like a hashmap, the database stores a collection of key-value pairs, and the key serves as a unique identifier. Both the key and the value can range from simple objects to complex objects.

Unlike relational databases, Key-Value databases are schemmaless (meaning it manages information without the need for a blueprint) and different keys are values can have completely different structures. One key might be a string and map to a JSON object, and another be an interger and map to a list. Below is an example of what the structure for a key-value database would look like:

```
Key: 'user:123'
Value: name=John Doe, age=30, email=johndoe@example.com

Key: 'product:456'
Value: name=Widget, price=9.99, description=A useful widget for various purposes.

Key: 'order:789'
Value: customer_id=user:123, products=product:456|product:789, total_amount=29.98
```

In the example above, for dictionaries (objects) in JSON, each attribute is represented as key=value, and for arrays, the elements are separated by vertical bars(|)

A good example of a key-value store is redis. Redis is an in-memory. Key/value store and stores data in-memory. This makes Redis exceptionally fast for data retrieval since RAM is significantly faster compared to disk-based storage. Not all of the NoSQL databases store data in RAM, however.

Document Databases

Document Databases store data as “documents”. These “documents” are JSON-like. This makes it easier for developers to store and query data because the format that the database stores data in is the same document-model format develoeprs use in their application code. It provides flexibility because individual fields in a document can be added or removed independently.

```
{
  "field1": {
    "onetype" : [
      {"id": 1, "name":"John Doe"},
      {"id": 2, "name":"Don Joeh"},
    ],
    "othertype": {"id": 2, "company": "ABC"}
    },
    "field2": {
    "list1": [[1,42],[2,2]]
  }
}
```

An example of a document database is MongoDB. MongoDB will store data in JSON-like structure observed above.

Wide-Column Database

A wide-column database stores data in columns. Each column corresponds to a specific attribute or field of the data, an operation that traditional row-based high write throughput but also optimizes reads and aggregations over a particular subset of data, an operation that traditional row-based storage isn’t designed for.

The real power of wide-column databases comes into play in scenarios requiring very large scale, such as internet search and web messaging systems. If an application needs to manage a lot of timestamp data, wide-column databases will excel in handling that. While we won’t dive too deep—as these databases can get complex—it’s crucial to note that they are specialized for certain use cases and my not by the best fit for every application.

Here is what the timestamp example looks like in Apache Cassandra:

```
Row Key: Location_ID (e.g., 'location1', 'location2', etc.)

Columns:
  - Timestamp1: Temperature1
  - Timestamp2: Temperature2
  - Timestamp3: Temperature3
---
- TimestampN: TemperatureN
```

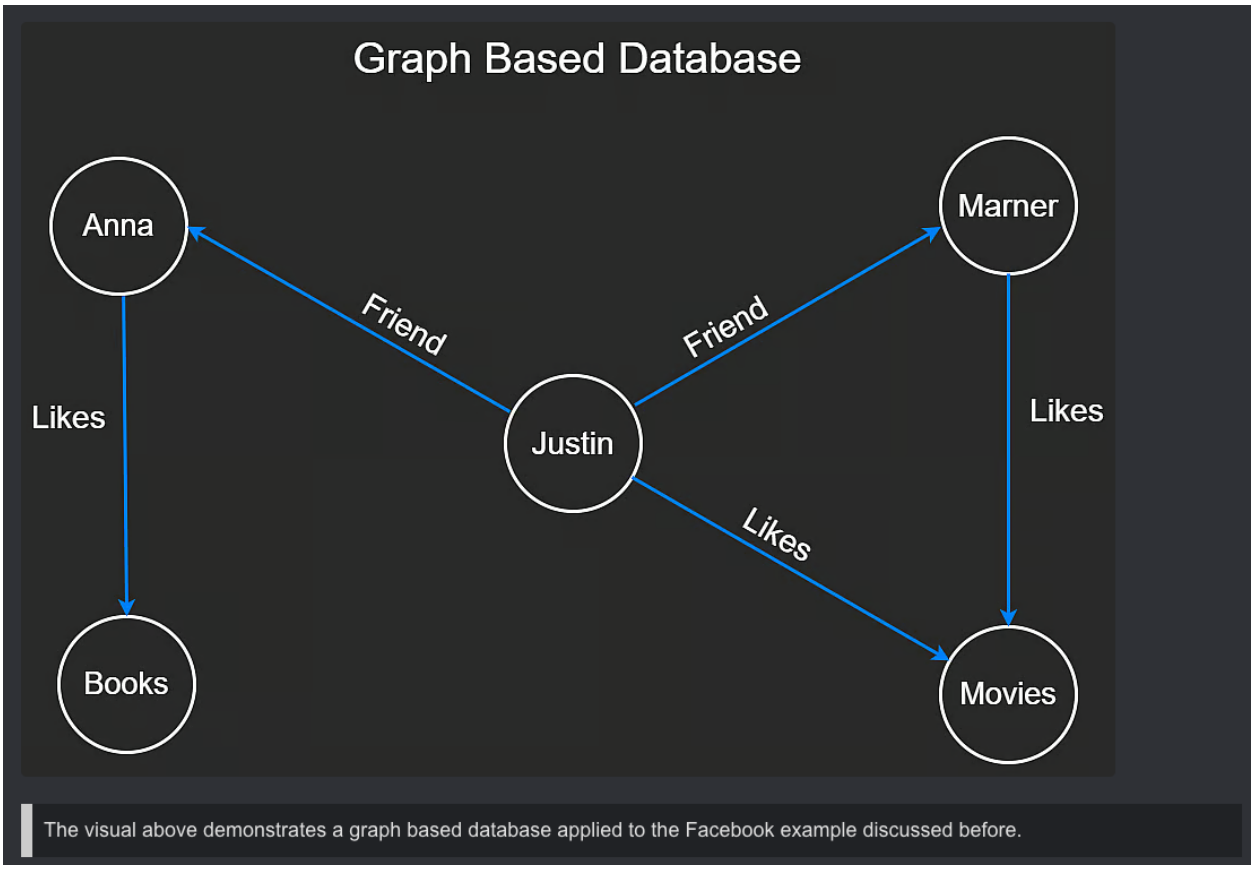
> Another example of this is Cassandra and Google's BigTable. Bigtable is a fully managed wide-column and key-value NoSQL database service for large analytical and operational workloads as part of the Google Cloud portfolio. Cassandra, similar to BigTable, is designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

Graph databases

A graph database uses a graph like structure where each node refers to an entity. Much like the graphs in algorithms, nodes in a graph database are connected to each other through edges or relationships. They can be used to represent various types of relationships, friendships and associations between entities.

By representing these entities as nodes, graph databases provide a flexible and intuitive way to model complex data structure and query the connections between entities efficiently.

Graph databases are very much useful when the data has complex relationship and interconnectedness. Facebook is a prime example of a use case where a graph database could be a good fit. Facebook’s platform involves relationships between users. Friends, friends of friends, interests, hobbies. Graph databases would be more adept at modeling and querying various types of social interactions, comments, likes, shares and relations. In that sense, we get an idea of when we might need a graph-based database.



Why do we need NoSQL databases.

A deeper dive into why we need NoSQL databases

As we discussed before, the biggest issue with SQL databases is scale and the restrictions. Because there are no foreign key or join constraints, the data can be split and stored on different servers. This means half the data can be stored on one databases, in one part of the world and the other half can be stored on another database, in a totally different part of the world and the other half can be stored on another database, in a totally different part of the world. NoSQL databases are designed with distributed architecture in mind. They are often built to handle large amounts of data across multiple nodes from the ground up.

Dropping the foreign key constrant allows this because we don't have to cross reference data between tables, which can be inefficient.

Another key concept we discussed in relational databases was the acronym ACID. Can NoSQL databases be ACID-compliant? The answer is yes. For example, MongoDB is one of the NoSQL databases that is ACID-compliant. An acronym that is used with NoSQL databases is BASE. ACID focuses on strict consistency within the databases, BASE focuses more on eventual consistency (which we will explain soon). That is not to say that one is

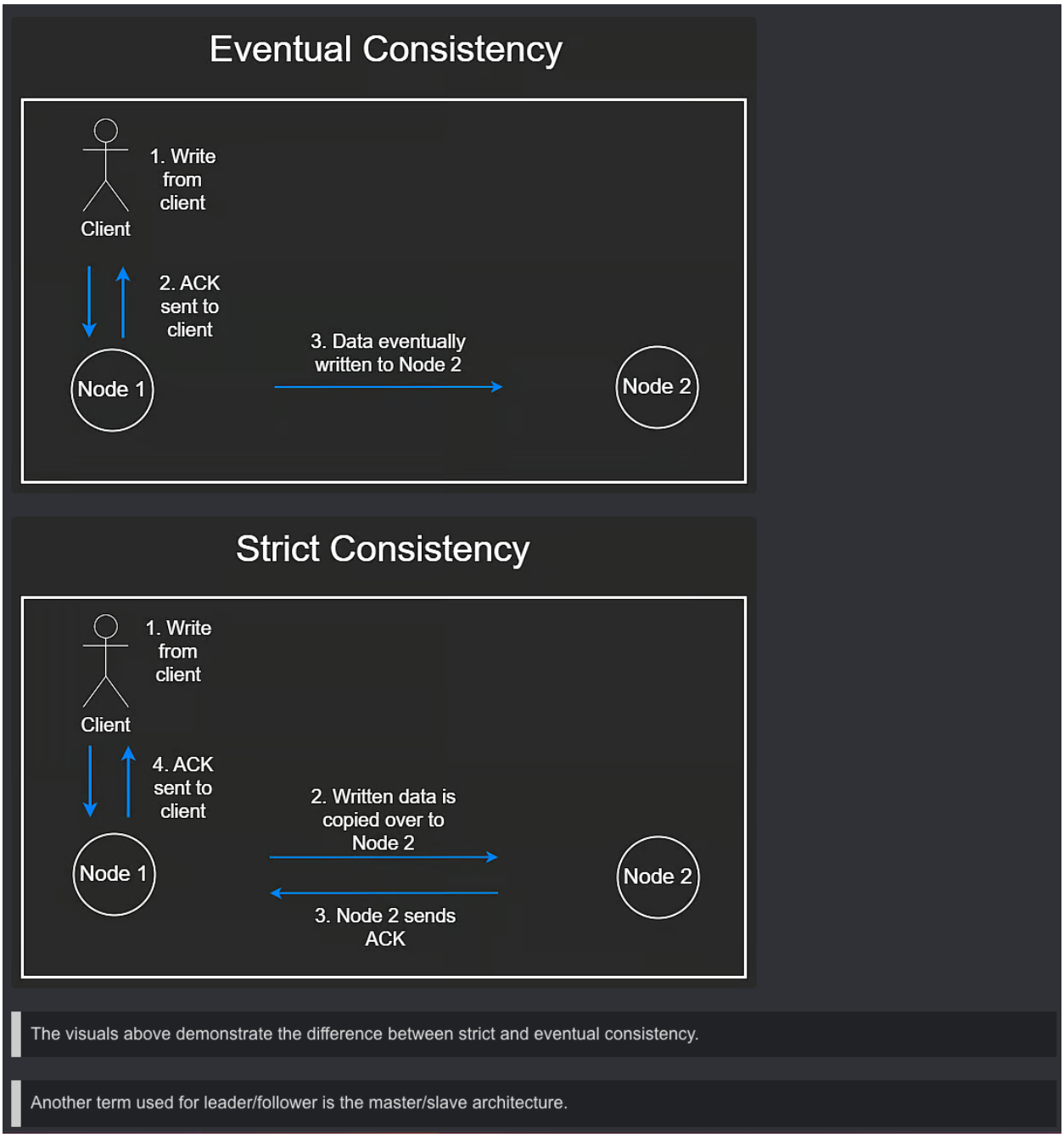
better than the other, but ACID is more common in SQL databases and BASE describes NoSQL databases better.

BASE stands for “Basically Available, Soft state, Eventual consistency” – the ‘eventual consistency’ part is what we will focus on.

Eventual Consistency

A mechanism that provides eventual consistency is the leader/follower architecture. Suppose that we have three database replicas and the query request to fetch the data can be directed to any one of these replicas. However, the first one may be said to be a primary database node. However, one of the nodes in the replica set will be a primary node. Updates and writes can only be done to the primary node and the primary node eventually updates the rest of the nodes. In this case, the primary node can be said to be the leader and the rest of the nodes being the followers.

Because of the nodes, apart from the primary node are only eventually being updated, there might be times when a user requests data and it is stale. This can be true is apps like Twitter or Instagram where updating the follower count may be delayed because the leader node has not updated the rest of the nodes.



The visuals above demonstrate the difference between strict and eventual consistency.

Another term used for leader/follower is the master/slave architecture.