Rajat Kumar Pradhan
19111045
6th semester, Biomed

NoSQL databases for big data

January 26, 2022

1 Abstract

Many challenges encountered while dealing with particular specific applications, such as the storing of very large datasets, have led to the development of NoSQL solutions. Traditional RDMS, on the other hand, ensure data integrity and transaction consistency. However, this comes at the expense of a tight storage architecture and difficult management. Data consistency and integrity are necessary in many circumstances, such as financial applications, however they are not always required. The purpose of this paper is to provide a detailed picture of NoSQL's evolution and mechanics, as well as the benefits and drawbacks of the most popular NoSQL data models and frameworks. To that end, a detailed comparison of SQL and NoSQL databases is offered first. Scalability, performance, consistency, security, analytical capabilities, and fault-tolerance techniques are all explored. Second, the four major types of NoSQL databases: key-value stores, document databases, column-oriented databases, and graph databases are defined and compared. Third, we compare the key technical solutions for each NoSQL data model.

2 Introduction

There are a number of phenomena that have aided the proliferation of NoSQL databases and bolstered their appeal. For example, the advent of new web technologies and low-cost storage, as well as tendencies toward Web 2.0, Web 3.0, and big data, have all contributed to the internet's rapid growth. Data storage, processing, analysis, and display face increased demands and obstacles as a result of these trends.

Indeed, massive data sets contain a variety of organized, semi-structured, and unstructured data that relational database management systems RDBMS and SQL struggle to handle. In fact, standard data storage and querying languages necessitate data organization in a specific and specified way, which is insufficient in the context of big data (where rapid and huge volumes of data are generated in heterogeneous formats).

To meet this problem, NoSQL databases (a phrase that encompasses all non-relational databases) were created to address big data challenges and issues such as large-scale storage of a variety of data types, the requirement for flexible schemas, and the need for scalable, quick, and distributed databases.

In fact, they offer a variety of categories to meet the needs of various use cases, including key-value pairs, documents, graphs, columnar databases, and geospatial databases. As a result, NoSQL is a database management system that is well-suited to distributed systems and non-relational data storage systems such as HDFS. It's important to note that NoSQL systems supplement rather than replace RDBMSs with their common query language SQL.

It's also worth noting that NoSQL solutions were not designed with the same goals in mind as SQL-based solutions. While relational databases are designed to store structured data and perform transactions, NoSQL databases were built to address the storage issues associated with large unstructured datasets. Both systems, in reality, have advantages and downsides.

NoSQL was created a long time ago before it was widely adopted. According to some sources, the word NoSQL was initially used in 1998 for relational databases that did not use SQL, and then again in 2009 for a non-relational database conference in San Francisco. Indeed, the adoption of NoSQL databases was aided by the proliferation of clouds, particularly Databases as a Service, and the pressing need for fast, scalable, and less expensive databases to handle massive data.

There were other factors that aided the spread of NoSQL. That is, to support object-oriented principles while avoiding costly object-relational mapping, data must be stored in a simpler structure. As a result, NoSQL was developed to meet the needs of simple, non-complex applications. Another trend is the rise of online technologies and cloud computing, both of which require low maintenance and scalability. There was also a trend in programming languages and frameworks to mask the complexities of SQL and relational databases in order to provide more flexible and convenient solutions (e.g., Ruby, Java Persistent API, ADO.net, etc.).

The purpose of this paper is to provide a clear picture of NoSQL's evolution and mechanics, as well as the benefits and drawbacks of the most popular NoSQL data models and frameworks. To that end, a detailed comparison of SQL and NoSQL databases is offered first. The four major types of NoSQL databases are defined and compared in the second section. Finally, we present and compare the various technical solutions for each NoSQL data model.

3 Comparing SQL and NoSQL databases features

This section gives a quick overview of the most essential characteristics of several NoSQL systems, allowing you to compare their performance, benefits, and drawbacks to traditional relational database systems.

For many years, RDMS and SQL databases were the standard. However, they can only support structured datasets and can only manage one type of preconfigured schema. They ensure atomicity, consistency, isolation, and durability (ACID) qualities, which are critical in a variety of applications. In fact, ACID properties are one of the most important features for ensuring the security of online transactions.

NoSQL, on the other hand, is better suited to non-relational, unstructured datasets, such as big data. They support a wide range of data schemas. They are, however, slow to handle massive queries, whereas SQL databases are faster and better suited to complicated and intense queries. Using commodity servers and clusters, NoSQL provides a less expensive solution to manage huge data.

When dealing with huge data and some web applications, RDMS runs into three major challenges. This includes the following:

1. Scale out data
2. Performance of single servers
3. Rigid schema design

Scalability, performance, flexibility, cloud usage, data model, queries and analytics, security, data replication, standardization, and maturity are all factors to consider when comparing relational vs NoSQL databases.

3.1 Scalability

In relational databases, big data storage, retrieval, and processing are complicated concerns. SQL databases are, in fact, vertically scalable. Users must enhance the capacity and performance of a single server to manage increased load. This is accomplished by boosting the dedicated database server's CPU, RAM, or SSD capacity, for example. Sharding numerous tables across huge clusters or grids, on the other hand, is expensive and difficult in such an architecture.

NoSQL databases, on the other hand, are horizontally scalable. Users only need to add servers to the NoSQL database infrastructure to accommodate high data volumes. As a result, the workload is divided among a large number of servers. As a result, achieving system scalability using NoSQL is easier and less expensive.

3.2 Performance

Relational databases, on the other hand, necessitate a specified data model and organized data. They provide advanced SQL functionality for managing, updating, and querying data. They have a number of advantages, including maintaining the integrity, consistency, and reliability of data and transactions. In comparison to NoSQL databases, relational databases are unquestionably more reliable. By adhering to ACID principles, SQL databases ensure the consistency and integrity of data and transactions.

In the event of large, expanding datasets, however, ensuring ACID characteristics is difficult. As a result, NoSQL databases rely on BASE concepts (essentially available, softstate, eventually consistent). As a result, they provide a versatile architecture that can handle organized, unstructured, and semi-structured data. It is simple for users to do frequent code pushes and iterations.

It's worth noting that the consistency, availability, partition tolerance (CAP) theorem is the source of both ACID and BASE features. BASE principles are more flexible than ACID principles, while ACID features provide transaction consistency and reliability. Those two attributes, however, come at the expense of performance and significant investments. Users must evaluate their needs in terms of flexibility and performance based on the use case and business needs. They can pick between relational databases, which guarantee consistency through ACID features, and NoSQL databases, which prioritize flexibility and performance when handling huge datasets and managing several servers in a cluster, even if flexibility comes at the expense of integrity.

Furthermore, relational databases are inefficient when dealing with large amounts of data. In truth, the performance of relational databases degrades as data volume grows, especially in big warehouses dealing with semi-structured data. Furthermore, when it comes to scalability, they necessitate a significant expenditure (e.g., adding servers to store and processing large datasets require purchasing additional licenses).

Furthermore, the growing demand for real-time analysis of enormous, developing heterogeneous data volumes adds a new layer of complexity. Furthermore, RDBMS row storage models are slower than column stores (e.g., statistical processing is slow in RDMS). Some studies suggest that RDMS be upgraded in order to deal with this problem. It is proposed that RDMS be expanded to support matrices and mathematical libraries, as well as array storage. Other experts agree that NoSQL databases and schema-free databases hold promise (e.g., graphs or object-oriented databases).

Unlike RDMS, NoSQL databases have been modified and improved to give the scalability, performance, and flexibility required for big data applications. For example, it has been stated that billions of data can be injected per day into Zvent's column-store Hypertable, while Google can use MapReduce to handle 20 petabytes of data stored in BigTable. Furthermore, they are built on less expensive hardware and technology than relational databases. NoSQL is also preferred for some simple applications that do not require the extensive capabilities of RDMS or the assurance of data integrity, such as banking transactions. As a result, NoSQL eliminates the need for relational databases' excessive complexity. Social media sites and large online applications, for example, may not always require secure transaction and ACID features (e.g., updating Facebook status or Tweets comments). In those circumstances, zero data loss and zero service interruption aren't critical. Furthermore, as compared to the utility of social media, adopting ACID features of RDMS can be costly.

3.3 Cloud usage

Relational databases are not appropriate for use in a cloud setting. In fact, RDMS are based on ACID principles and have restricted scalability. As a result, they are unable to handle very big semi-structured and unstructured datasets.

NoSQL databases, on the other hand, are the greatest choice for cloud applications. This is due to the fact that they offer increased availability, scalability, performance, and flexibility. They are capable of dealing with a wide range of data (structured, semi-structured and unstructured data).

3.4 Data models

On the one hand, change management in relational databases is difficult to manage. Before data injection, users must define the database schema. Furthermore, any changes to the database schema or tables should be thoroughly investigated. Otherwise, such modifications may result in service failure, reduced performance, or the need for further maintenance and investment to modify application modules.

NoSQL databases, on the other hand, provide for simple change management. In reality, there is no need to establish a rigorous database schema ahead of time. This gives you the freedom to store data without having to adhere to a strict schema. Furthermore, the data model can be changed at any moment without compromising the performance of the system or application. As a result, users have the option of selecting the best data model and database for their needs.

3.5 Queries and analytics

A query language is a programming language that allows programmers to manage data in a database.

The common structured query language (SQL) standard is used by users of relational databases to launch queries. There is no standard for querying NoSQL databases, though. Each NoSQL database has its own style of managing, extracting, and querying data. As a result, data scientists confront the task of learning each NoSQL database's query language.

SQL databases, on the other hand, are capable of handling sophisticated queries using a standardized interface. When dealing with complicated queries, however, NoSQL databases fall short. In NoSQL databases, joints are difficult to achieve. NoSQL, on the other hand, is better suited to handling concurrent computations and mathematical equations on distributed big and changing datasets.

NoSQL systems are less suitable for business intelligence use cases than RDMS. This is because, as previously said, NoSQL is typically difficult to employ for advanced analytics sophisticated queries and joins.

3.6 Security

Secure techniques are built into relational databases in general. They are, however, still vulnerable to SQL injection, cross-site scripting, root kits, and insecure communication protocols.

NoSQL databases, without a doubt, provide greater scalability and flexibility. However, most NoSQL databases lack built-in security features. As a result, users must cope with a variety of security risks. On top of NoSQL databases, some security tools and modules have been implemented. In comparison to relational databases, they have a very thin security layer.

Most NoSQL databases, in reality, do not encrypt client-server communications or provide authentication or auditing capabilities.

CouchDB, for example, provides auditing but saves user names and passwords in log files. Data security is jeopardized as a result of this. Users are frequently required to add extra components to NoSQL infrastructure to ensure authentication.

Furthermore, while relational databases make encryption of organized data easier, very large unstructured data sources are challenging to encrypt. As a result, most of these sources are saved in NoSQL in a straightforward way.

As a result, we may conclude that NoSQL has a lot of security risks because it isn't mature. However, in other circumstances, such as protecting valuable, confidential, or sensitive large sources, security is critical (e.g., health, government, system security and so on).

3.7 Sharding

Sharding is a common approach in which many servers are used. It refers to the process of distributing enormous amounts of data from a single database over several servers and virtual data nodes. Because each server handles various data portions, sharding improves performance.

It is, however, advised that replication be used instead of sharding. This is due to the fact that replication ensures both performance and dependability.

Sharding is used in NoSQL databases to balance the load and ensure parallel storage and processing. They provide the valuable capability of adding or removing data layer servers without compromising application performance.

RDMS, on the other hand, were not designed with this in mind. Instead, RDMS now has a sharding feature. Multiple servers are used to split tables. Sharding is built on the relationship between shards (data divisions) and the data nodes that contain them. A dynamic or static mapping can be used. One disadvantage of sharding is that it prevents shards from being joined together.

3.8 Data replication

The concept of data replication is the distribution of data throughout a system. A non-interactive and trustworthy approach is superior for achieving such a concept.

In the case of relational databases, replication is difficult to achieve. This is due to the fact that they were not designed to handle horizontal scaling. Replication and backup of relational databases is done using a semi-manual procedure.

In the case of big data, however, automatic live recovery of huge and geographically distributed datasets is frequently necessary.

Data mirroring is a common method of data redundancy. They duplicate data to target arrays in the data center or to a remote location. This method uses a lot of storage space, especially when dealing with massive datasets in the petabyte range. In fact, storing massive streams of data (data in motion) as well as big data archives using standard methods is an overhead and costly for businesses.

Because NoSQL databases are horizontally scalable, they make data replication easy to maintain and prevent data loss. In fact, most NoSQL databases offer fault-tolerance through automatic data replication. Data is replicated between cluster computers and even data centers. They make it simple for users and administrators to create and optimize replication settings by describing where and how data should be copied across remote systems.

Developers don't have to worry about the complexity of the heterogeneous storage environment or parallel processing processes when they use big data technologies and NoSQL databases.

A good example of real-time data replication is IBM InfoSphere data replication. Real-time data replication ensures sustained high data availability in both heterogeneous and homogeneous situations, according to an IBM White Paper (2014). To provide reporting, interactive analysis, and ensure synchronized transactions, real-time data replication is required. It aids in the extraction of reliable information, speedy decision-making, and resource optimization.

Another option is described, which is based on an error-correcting algorithm called erasure coding and is used in conjunction with an object-based storage technique. In a distributed setting, such a technique is an alternative to data replication. A data object (for example, a document with metadata) is divided into segments. Each segment is encoded and divided into slices, which are then kept on separate servers. As a result, even if some slices are no longer accessible due to disk loss, the organization may reconstitute the original data. This method saves money, saves space, and ensures fault-tolerance repositories. It is, however, still in its infancy.

3.9 Standardisation and open source

NoSQL databases are free and open source. This could help them gain more traction among big data users. However, because it does not support standardization practices, it could be a disadvantage. Each NoSQL Database, in reality, is distinct from the others. This is likewise true for the queries they support. Indeed, there is no reliable standard for NoSQL databases as far as we know. Each one has its own set of queries. As a result, developers and administrators must learn and be trained on each NoSQL option accessible. Relational databases, on the other hand, are standardised and use a common SQL language.

3.10 Maturity

Relational databases are widely employed in businesses around the world due to their popularity. They've been around for quite some time. As a result, they give a common query language, a wealth of functionality, and a higher level of acceptance. There are also a variety of professionals and consultants who can assist businesses with exploitation, management, and administration of their traditional databases. This fact encourages their growth and maturity by allowing a large number of professionals to engage in reporting flaws and improving the use of relational databases.

Instead, despite the fact that NoSQL databases have been around for over ten years and are developer-friendly, they still have a long way to go in terms of general adoption. In reality, as compared to relational databases, NoSQL databases are still in their infancy. Furthermore, there is a scarcity of database developers and administrators who are familiar with these types of databases. Their maturation is slowed as a result of this.

4 Types and main characteristics of NOSQL databases

Experts categorise NoSQL databases based on a variety of factors, including the data model. Such databases are well-suited to dealing with the complexities of big data and its 3Vs (volume, velocity and variety). However, each form of NoSQL Database provides a distinct level of flexibility and data model to address different big data scenarios.

In fact, customers can choose from a variety of NoSQL databases based on their data format and storage and retrieval requirements, including document, key-value, column family, and graph databases.

Various databases are available in the NoSQL environment. HBase, Cassandra, DynamoDB, MongoDB, Riak, Redis, Accumulo, and Couchbase are some of the most well-known.

4.1 Key-value databases

Schemas aren't required for key-value storage. This concept does, in fact, offer interesting storage flexibility as well as a straightforward structure based on a hash table. A key-value pair is formed by a unique key and a pointer to a specific piece of data in such a database. As a result, key-value stores are well suited to operations based on key properties. Indeed, hash tables are handy for searching exceedingly big databases for simple or complicated values. Users who require a specific structure for their data can rely on key-pair collections.

In terms of flexibility, key-value stores allow for the addition of any sort of new value at runtime while maintaining system availability. This can be done without jeopardizing previously saved data, which may have a different structure.

A vast number of records can be stored in a key-value store. They can handle millions of simultaneous users and enormous volumes of state changes per second thanks to distributed processing and storage. The datasets of most key-value databases are stored in memory. As a result, they're ideal for caching intense SQL queries. Furthermore, by computing portions of a webpage in advance, those stores make it possible to speed up the display of online pages. The result can be rapidly fetched and presented when user-IDs make a request.

They're great for both storing and generating reports based on the results of analytical algorithms (such as phrase counts over large amounts of documents).

Key-value databases, on the other hand, inherit one of NoSQL databases' flaws. They don't offer any form of standard database functionality. Users should instead rely on the application to maintain transaction atomicity or the consistency of numerous simultaneous transactions.

Another disadvantage is that consumers are unable to sort data by value. Indeed, querying a key value data store to extract all records containing a specific set of values is impossible. A key-value database may only be queried by expressing a request either by key or by a range of keys.

4.1.1 Key-value databases examples and comparison

Following are comparisons of three key-value databases: Redis, Riak, and Voldemort. All three systems offer scalability, fault tolerance, and a near-linear performance gain.

Riak is built on a basic and symmetric architecture and is intended for use in highly distributed contexts like the cloud. Riak, like Voldemort, uses a consistent hashing system. It uses a map-reduce programming model to divide the work into several jobs that are distributed over multiple cluster nodes.

Any node in Riak has the ability to respond to a client request. Riak does not rely on a single node to track system status. Instead, it uses a node-to-node gossip protocol to keep track of node state (nodes that are alive, nodes that hold data). Riak has a higher fault tolerance than Redis, although it performs worse.

Because it relies on in-memory datasets for fast replies, Redis is better suited for time-critical applications. Redis, like other key-value stores, allows for simple actions like inserting, deleting, and looking up data. Redis, like Voldemort, allows users to associate lists and sets with a key as well as a blob (huge data object) or a text. It also supports operations like list and set.

Redis is well suited to dealing with quickly changing data, such as real-time sensor data collecting and real-time communications. Voldemort, on the other hand, is best suited to very huge datasets such as geological data and map metadata. Indeed, it is capable of storing large amounts of data without sacrificing performance.

Experiments have shown that Redis scales with growing dataset quantities but not with rising node counts. Voldemort, on the other hand, scales as the number of nodes increases, but not as the quantity of datasets grows. In comparison to Voldemort, Redis provides superior data availability. However, when dealing with very large datasets, both of them demonstrate a reduction in availability. It has also been demonstrated that adding nodes to Voldemort's system improves its availability.

4.2 Document databases

Document databases were created with the purpose of storing and managing vast amounts of data. Each document in this database is assigned a key value. Multiple key-value pairs, key-array pairings, and even nested documents can be found in documents. Documents are encoded in XML, JavaScript option notation (JSON), or binary JSON, which are all standard data transmission formats (BSON). Document databases are widely recognized as a powerful, adaptable, and agile solution for storing large amounts of data. Indeed, because data is saved in a JSON format that can be interpreted, such stores may accommodate a wide range of data types and are convenient for developers.

The document stores, unlike key-value stores, allow you to query collections based on multiple attribute value constraints. In fact, document databases allow users to search for data based on the content of documents, whereas key-value stores only allow users to search for data by key value. They have the option of searching by keys, values, or examples. In fact, because the encoded documents contain metadata objects, data can be queried by example.

Complex data structures, such as nested objects, can be handled more readily in document databases. As a result, document stores can be more expressive than column family data models. They also allow you to use secondary indexes, query nested documents, and perform operations like as 'and,' 'or,' and 'between.' Users can utilize a query language or a comprehensive programming API to launch queries. These options give document databases with the flexibility that multiple use cases demand. They're commonly used for real-time analytics, logging, and the storage layer of small, flexible websites such as blogs. This is due to the fact that they are simple to maintain.

In contrast to basic key-value stores, document databases' value column contains semi-structured data, notably attribute name/value pairs. Document databases also allow for a customizable schema. They do not have any schema restrictions, and they let you to store documents with hundreds of attributes in a single document scheme column. As a result, different amounts and types of characteristics can be assigned to different rows. Attributes can also be added during runtime.

4.2.1 Document databases examples and comparisons

In this section, we compare CouchDB and MongoDB, two prominent document databases. Both are open source document databases that are built to easily scale across several nodes. Data is kept in documents with self-contained records and no inherent links in both cases. MongoDB, on the other hand, ensures consistency, ensuring that each client sees the same data. CouchDB, on the other hand, ensures availability. As a result, all clients can read and write at any time.

MongoDB users can utilize a simple and intuitive query-like language to display the query's results. They take the form of a JSON-like format. CouchDB, on the other hand, uses a 'map-reduce' technique and a view notion. Queries are executed using CouchDB's 'views.' To specify field constraints, they are defined using Javascript.

CouchDB uses append-only files to save data to disk, whereas MongoDB uses a memory-mapped storage engine. For all disk I/O, memory mapped files are used. CouchDB provides an HTTP API for data access and administration as an interchange standard. Instead, MongoDB uses BSON, a socket-based wire protocol.

CouchDB supports both master/master and master/slave replication for fault tolerance. Replication filters can be used to fine-tune replication. MongoDB, on the other hand, uses replica sets, which are a type of asynchronous master/slave replication.

To summarize, both CouchDB and MongoDB share many common features, such as replication for fault tolerance and data storage in a volatile memory file system. Both use the MapReduce paradigm to process data and have a strong community behind them.

CouchDB, on the other hand, isn't designed to deal with rapidly changing data. In contrast, whereas CouchDB needs the creation of pre-defined queries, MongoDB is better suited to dynamic queries and has higher performance on large databases.

4.3 Wide-column databases

Wide columnar stores, orientated stores, and extensible record stores are all terms used to describe wide-column databases. They are a column-based expansion of the key-value architecture. Wide-column stores are built to handle distributed data across a shared infrastructure pool. Their adaptable architecture allows them to handle a high number of columns and schema modifications on a regular basis.

Wide column databases are based on hybrid techniques that combine the declarative qualities of relational databases with the structure of diverse key-value stores. The graphical representation of column family stores is comparable to that of relational databases. Wide column databases, on the other hand, only store a key value pair in one row if a dataset requires it, whereas relational databases save a null value in each column when a dataset has no value for it. This is a method of dealing with null values and sparse data that is more efficient (data with various numbers of attributes).

Wide column databases, in fact, are well-suited to applications that require massive amounts of data to be stored on very large clusters due to their data model's ability to be efficiently partitioned.