International Journal of Research in Advent Technology, Vol.6, No.2, February 2018
E-ISSN: 2321-9637
Available online at www.ijrat.org

A Survey Paper on NoSQL Databases: Key-Value Data Stores and Document Stores

Nikhil Dasharath Karande[1]
Department of CSE, Sanjay Ghodawat Institutes, Atigre
nikhilkarande18@gmail.com

Abstract- This paper focuses on next generation databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable i.e NoSQL. NoSQL databases have become more popularity in the recent years and have been successful in many production systems. The aim of this paper is to understand the different types of NoSQL databases and benefits of NoSQL databases. This paper includes survey information related to key-value data stores and document stores. At the end paper concludes with important aspects for both key-value data stores and document stores.

Index Terms- NoSQL, non-relational database, key-value data stores, document stores, RDBMS

1. INTRODUCTION

NoSQL is a non-relational database management system, different from traditional relational database management systems in some significant ways. It is designed for distributed data stores where very large scale of data storing needs. These types of data storing may not require fixed schema, avoid join operations and typically scale horizontally [1].

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS. Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL can mean "not SQL" or "not only SQL." As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises [3]. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images [9].

NoSQL databases are different than relational databases like MQSql. In relational database you need to create the table, define schema, set the data types of fields etc before you can actually insert the data. In NoSQL you don't have to worry about that, you can insert, update data on the fly. One of the advantages of NoSQL database is that they are really easy to scale and they are much faster in most types of operations that we perform on database [8].

2. BENEFITS OF NOSQL

NoSQL databases offer enterprises important advantages over traditional RDBMS, including [4]:

2.1. *Scalability*

NoSQL databases use a horizontal scale-out methodology that makes it easy to add or reduce capacity quickly and non-disruptively with commodity hardware. This eliminates the tremendous cost and complexity of manual partitioning that is necessary when attempting to scale RDBMS.

2.2. *Performance*

By simply adding commodity resources, enterprises can increase performance with NoSQL databases. This enables organizations to continue to deliver reliably fast user experiences with a predictable return on investment for adding resources—again, without the overhead associated with manual partitioning.

2.3. *High Availability*

NoSQL databases are generally designed to ensure high availability and avoid the complexity that comes with a typical RDBMS architecture that relies on primary and secondary nodes. Some "distributed" NoSQL databases

use a master less architecture that automatically distributes data equally among multiple resources so that the application remains available for both read and write operations even when one node fails.

2.4. *Global Availability*

By automatically replicating data across multiple servers, data centers, or cloud resources, distributed NoSQL databases can minimize latency and ensure a consistent application experience wherever users are located. An added benefit is a significantly reduced database management burden from manual RDBMS configuration, freeing operations teams to focus on other business priorities.

2.5. *Flexible Data Modeling*

NoSQL offers the ability to implement flexible and fluid data models. Application developers can leverage the data types and query options that are the most natural fit to the specific application use case rather than those that fit the database schema. The result is a simpler interaction between the application and the database and faster, more agile development.

3. TYPES OF NOSQL DATABASES

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories [5]:

3.1. *Key-value data stores*

Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned *and replicated* across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.

3.2. *Document stores*

Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.

3.3. *Wide-column stores*

Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.

3.4. *Graph stores*

A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

4. KEY-VALUE DATA STORES

As the name suggests, in a key-value store, data is represented as a collection of key–value pairs. It is also known as associative arrays, organized into rows. These databases store the data as a hash table with a unique key and a pointer to a particular item of data. Similar to traditional hash tables, it allows data storage and retrieval through keys. The key-value stores are used whenever the data would be queried by precise parameters and needs to be retrieved really fast [4].

4.1. *How does a key-value stores work?*

The key value stores do not impose a specific schema. Traditional RDBs pre-define the data structure in the database as a series of tables containing fields with well defined data types. Exposing the data types to the database program allows it to apply a number of optimizations. In contrast, key-value systems treat the data as a single opaque collection which may have different fields for every record. In each key-value pair the key is represented by an arbitrary string such as a filename, URI or hash. The value can be any kind of data like an image, user preference file or document. The value is stored as a blob requiring no upfront data modeling or schema definition.

This offers considerable flexibility and more closely follows modern concepts like object-oriented programming. Because optional values are not represented by placeholders as in most RDBs, key-value stores often use far less memory to store the same database, which can lead to large performance gains in certain workloads.

The storage of the value as a blob removes the need to index the data to improve performance. However, you cannot filter or control what's returned from a request based on the value because the value is opaque. In general, key-value stores have no query language. They provide a way to store, retrieve and update data using simple *get, put* and *delete* commands; the path to retrieve data is a direct request to the object in memory or on disk. The simplicity of this model makes a key-value store fast, easy to use, scalable, portable and flexible.

Now let us evaluate key-value stores in terms of different DBMs parameters [5]. **Concurrency:** In Key/Value Store, concurrency is only applicable on a single key, and it is usually offered as either optimistic writes or as eventually consistent. In highly scalable systems, optimistic writes are often not possible, because of the cost of verifying that the value haven't changed (assuming the value may have replicated to other machines), there for, we usually see either a key master (one machine own a key) or the eventual consistency model. **Queries:** As mentioned above, there really isn't any way to perform a query in a key value store, except by the key. Even range queries on the key are usually not possible. However, in many web application use-cases, the key-based access is required, and the need for the DBMS to actually "understand" the data is minimal. In use-cases like user profiles, user sessions, shopping carts etc, the DBMS can actually be oblivious to the data attributes and store this information as blob passing it to the application layer directly and relying on it to process it. Thus using key-value store in such cases makes it cheap to handle (one request to read, one request to write) when you run into concurrency conflict (you only need to resolve a single key). **Transactions:** While it is possible to offer transaction guarantees in a key value store, those are usually only offer in the context of a single key put. It is possible to offer those on multiple keys, but that really doesn't work when you start thinking about a distributed key value store, where different keys may reside on different machines. Some data stores offer no transaction guarantees. **Schema:** Key-value stores do not have a pre-defined schema – they have just two fields – a key and the value. They rely on the application using the data for parsing it. **Scaling up:** Key-value stores scale out by implementing partitioning (storing data on more than one node), replication and auto recovery. They can scale up by maintaining the database in RAM and minimize the effects of ACID guarantees (a guarantee that committed transactions persist somewhere) by avoiding locks, latches and low-overhead server calls [7].

The simplest way for key-value stores to scale up is to shard the entire key space that means that keys starting in A go to one server, while keys starting with B go to another server. In this system, a key is only stored on a single server. That drastically simplifies things like transactions guarantees, but it expose the system for data loss if a single server goes down. At this point, we introduce replication. **Replication:** In key value stores, the replication can be done by the store itself or by the client (writing to multiple servers). Replication also introduces the problem of divergent versions. In other words, two servers in the same cluster think that the values of key 'ABC' are two different things. Resolving that is a complex issue, the common approaches are to decide that it can't happen (Scalaris) and reject updates where we can't ensure no conflict or to accept all updates and ask the client to resolve them for us at a later date (Amazon Dynamo, Rhino DHT). **Portability and lower operational costs:** Key-value stores are portable because they do not require a complex query language. You can move an application from one system to another without rewriting code or constructing new architecture. Companies can expand their product offerings on new operating systems, without affecting their core technology [5].

4.2. *When to use key-value stores?*

Key-value stores handle size well and are good at processing a constant stream of read/write operations with low latency making them perfect for: Session management at high scale, User preference and profile stores, Product recommendations; latest items viewed on a retailer website drive future customer product recommendations, Ad servicing; customer shopping habits result in customized ads, coupons, etc. for each customer in real-time, Can effectively work as a cache for heavily accessed but rarely updated data.

Key-value stores differ in their implementation where some support ordering of keys like Berkeley DB, FoundationDB and MemcacheDB, some maintain data in memory (RAM) like Redis, some, like Aerospike, are built natively to support both RAM and solid state drives (SSDs). Others, like Couchbase Server, store data in RAM but also support rotating disks. Some popular key-value stores are: Aerospike, Apache Cassandra,

Berkeley DB, Couchbase Server, Redis, Riak. Common use cases for Key-Value Store: Storing data for customer preferences and using cache to accelerate application responses.

5. DOCUMENT STORES

A document database, also called a document store or document-oriented database is a subset of a type of NoSQL database. Some document stores may also be key-value databases. A document database is used for storing, retrieving, and managing semi-structured data. Unlike traditional relational databases, the data model in a document database is not structured in a table format of rows and columns. The schema can vary, providing far more flexibility for data modeling than relational databases [4].

5.1. *How does a document database work?*

A document database uses documents as the structure for storage and queries. In this case, the term "document" may refer to a Microsoft Word or PDF document but is commonly a block of XML or JSON. Instead of columns with names and data types that are used in a relational database, a document contains a description of the data type and the value for that description. Each document can have the same or different structure. To add additional types of data to a document database, there is no need to modify the entire database schema as there is with a relational database. Data can simply be added by adding objects to the database.

Documents are grouped into "collections," which serve a similar purpose to a relational table. A document database provides a query mechanism to search collections for documents with particular attributes.

5.2. *Benefits of a document database*

Document stores offer important advantages when specific characteristics are required, including [4]:

5.2.1. *Flexible data modeling*

As web, mobile, social, and IoT-based applications change the nature of application data models; document databases eliminate the need to force-fit relational data models to support new types of application data models.

5.2.2. *Fast write performance*

Unlike traditional relational databases, some document databases prioritize write availability over strict data consistency. This ensures that writes will always be fast even if a failure in one portion of the hardware or network results in a small delay in data replication and consistency across the environment.

5.2.3. *Fast query performance*

Many document databases have powerful query engines and indexing features that provide fast and efficient querying capabilities.

5.3. *When to use Document stores?*

Document stores are used when we require Flexible Schema: Document oriented databases are schema less which means two documents can have very different schema and data values, unlike relational model where each row in a table will have same columns. Fast Writes: Many document stores supports multi-version-concurrency-control there by making the writes to documents really fast. Whereas writes to RDMBS could be slow for various reasons like locks, transactional support, index updates, and so on. Partitioning: Document stores are effectively key value stores with document id being the key and the document itself being the value. Under such a setting, one cans simply shard/partition the document store by simply partitioning the key space. This process is rather complicated in a RDBMS setting where there are multiple tables and the query workload contains joins.

6. CONCLUSION

The aim of this paper was to give a thorough overview and introduction to the NoSQL database. NoSQL is evaluating new databases to support changing application and business requirements. NoSQL databases are becoming an increasingly important part of the database landscape, and when used appropriately, can offer real benefits.

Key-value store is the simplest data model. Technically it is just a distributed persistent associative array. The key is a unique identifier for a value, which can be any data application needs stored. This model is also the fastest way to get data by known key, but without the flexibility of more advanced querying. We conclude that it may be used for data sharing between application instances like distributed cache or to store user session data.

Document store is a data model for storing semi-structured document object data and metadata.

Documents can be queried by their properties in a similar manner to relational databases but aren't required to adhere to the strict structure of a database table. Additionally, only parts of the object may be requested or updated. We conclude that, document stores are used for aggregate objects that have no shared complex data between them and to quickly search or filter by some object properties.

REFERENCES

[1] Jing Han Meina Song and Junde Song. "A Novel Solution of Distributed Memory NoSQL Database for Cloud Computing" In ICIS 2011 10th IEEE/ACIS International Conference on Computer and Information Science 2011.
[2] Stonebraker, Michael; Madden, Samuel; Abadi, Daniel J.; Harizopoulos, Stavros, "The end of an architectural era: (it's time for a complete rewrite)," Proceedings of the 33rd international conference on Very large data bases, VLDB, p. 1150 – 1160, 2007.
[3] S Tonebraker, M.: SQL databases vs. NoSQL databases. Communications of the ACM, 53 (4): 10 - 11, 2010
[4] http://basho.com/resources/document-databases/
[5] http://basho.com/resources/nosql-databases/
[6] Nathan Hurst "Visual Guide to NoSQL Systems." http://blog.nahurst.comlvisual-guide-to-NoSQL-systems/
[7] "NoSQL databases," [Online]. Available: nosql - database.org.
[8] http://iranarze.ir/wp-content/uploads/2014/12/Kamputer-A-Survey-on-NoSQL-Databases.pdf
[9] NoSQL - http://nosql - database.org/