

NoSQL Key-Value DBs Riak and Redis

Cristian Andrei BARON
University of Economic Studies, Bucharest, Romania
andrei.baron30@gmail.com

In the context of today's business needs we must focus on the NoSQL databases because they are the only alternative to the RDBMS that can resolve the modern problems related to storing different data structures, processing continue flows of data and fault tolerance. The object of the paper is to explain the NoSQL databases, the needs behind their appearance, the different types of NoSQL databases that current exist and to focus on two key-value databases, Riak and Redis.

Keywords: NoSQL Databases, Key-Value, Riak, Redis, RDBMS

1 Introduction

Nowadays, we talk more and more about NoSQL (Not Only SQL) databases because we are dealing with very large volumes of information from different sources that have to be stored and analyzed in real time. For the last decades, the relational model has been the first and maybe for some times the only viable solution for both small and big companies. In the last years some of the biggest Internet companies such as Google, Facebook and Amazon have invested a lot of money in developing alternative solutions for the RDBMS to fulfill their needs. We don't consider that the RDBMS will disappear in the future years because there is a strong community based on this type of databases, and big companies that are using enterprise applications will continue to use RDBMS mostly because of the support that the RDBMS vendors are offering. But a big part of the rest of companies and individuals will search to discover alternatives options, like schema-less, high availability, MapReduce, alternative data structures and horizontal scaling that are supported by the various types of NoSQL databases. When a new application that will need a storing and retrieval data mechanism, begins to be developed, we should first analyze deeply the business needs of the enterprise, the structure, amount and speed of the data that will be managed

and decide if we will proceed with the standard RDBMS or we will choose a type or a combination of NoSQL data store types. The NoSQL databases are divided in a variety of genres, many of them are part of one of these major categories: key-value, wide column, graph, and document-based. It is important to learn for what kinds of problems they are best suited, what aspects they resolve that the RDBMS cannot, if they are focused on flexible schemas and querying mechanism or they are focused on storing large amounts of data across several machines. When choosing the correct NoSQL database there are several questions that we must answer.

First one refers to the way that you can talk to the database, this means that you must check the variety of connections, if it has or not a command-line interface, in what programming language is written (C, Erlang, JavaScript) and what protocols it supports (REST, Thrift).

A second question can refer to the aspects that make the NoSQL database unique, some can allow querying on arbitrary fields; others can provide indexing for rapid lookup or support ad hoc queries.

The last questions can refer to performance and scalability; this aspects are in strong relationship with the unique qualities of NoSQL databases because sometimes we may be constrained to give up from performance or scalability in order to enjoy some unique functionality. Aspects related to performance may include supporting

sharding, replication or ability for tuning reading, writing or some other operations, where scalability refers mainly to the supported type: horizontal scaling (Riak, MongoDB or HBase), traditional vertical scaling (Redis, Postgres or Neo4J) or a combination of this two types [1] .

2 Classification of NoSQL Databases

The NoSQL databases can be divided based on the optimization strategy and on the different kinds of tasks that they resolve in four groups: key-value, document-based, wide column and graph, pictured in **Fig. 1**.

Type	Examples
Key-Value Store	 
Wide Column Store	 
Document Store	 
Graph Store	 

Fig. 1. NoSQL classification [5]

The “Key-Value” model presented in **Fig. 2**. stores data as simple identifiers (keys) and the associated values in standalone tables, called often “hash tables” where data retrieval is usually performed using the associated keys. The values can be of different types, from simple text strings to complex lists or sets, but the queries can be run only against keys, and are limited to exact matches and because of their simplicity, they are ideal to be used for highly scalable retrieval of values [2].

Example of key-value databases: Redis, Riak, Dynamo (Amazon), Voldemort(LinkedIn)

Key: 100	Badge : 112355	Name : John	Age : 45	Nickname: J
Key: 101	Username: alice.walker		Password: A12345	
Key: 102	Google Account: james@gmail.com		Id: 456	Location: Bucharest

Fig. 2. Key-Value Model

The “Document-based” model main concept is the idea of a “document”, depicted in **Fig. 3**.

There are many document-oriented database implementations but all of them encapsulate and encode information in a standard format or encoding like XML, JSON (Javascript Option Notation), YAML or a binary format BSON. This model consists basically of version documents that are collections of other key-value collections. This model represents the next iteration of key-value, allowing nested values associated with each key and supporting a more efficiently query mechanism [4]. Unlike the simpler version of key-value stores, in this type of store, the value column contains semi-structured data, mainly pairs of attribute names and values. The value of a column can reach hundreds of attribute pairs, where the type and the number of attributes can vary from one row to other, but in the same time both keys and values remain fully searchable [2] .

Example of Document databases: CouchDB (JSON), MongoDB(BSON)

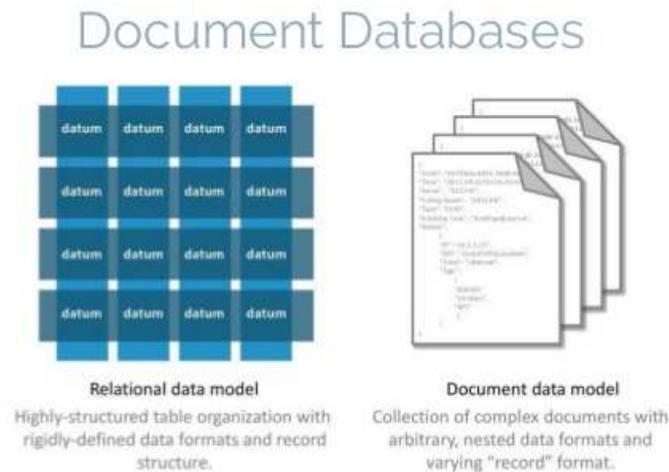


Fig. 3. Document-based model [6]

“Wide Column” model pictured in **Fig. 4.**, also known as “Column-Family” or “Big Table implementation” has a database structure that is similar to the standard RDBMS because all the data is stored under sets of columns and rows. One important functionality is the grouping of the often used columns in

column family [3]. This model is best to be used for distributed data storage, large-scale, batch-oriented data processing like sorting, parsing, conversions between hexadecimal, binary and decimal code and predictive analytics [2].

Example of wide column databases: Cassandra, HBase, BigTable(Google)



Fig. 4. Wide Column model [7]

The “Graph” model is designed to store data and the relations between them that

can be easily represented as a graph consistent of interconnected elements with a

limited number of relations between them. This model pictured in **Fig. 5.** is very useful for social networking, road maps or transport routes, generating recommendations (suggestions) or pattern detection where you are more concentrated on the relationships between data than in the data itself. Graph databases have a different terminology compared to the other NoSQL databases that were presented earlier, mainly because they are designed based on the graph architecture. We can identify “edges” that are kind of joins between different rows of a table and “nodes” that can have properties and values and are similar to the table rows [2].

Example of graph databases: Neo4j, InfoGrid, Sones GraphDB, AllegroGraph, InfiniteGraph.

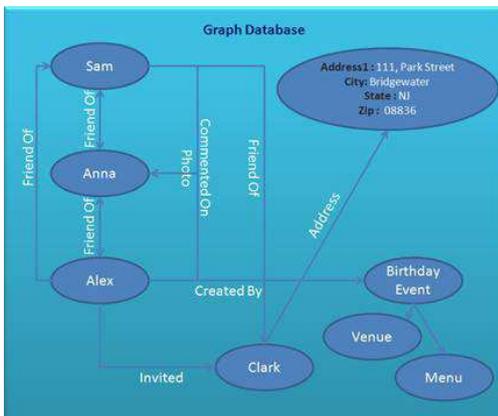


Fig. 5. Graph model [8]

3 Riak Database

Riak represents a distributed key-value database that can store any type of values, from plain text, JSON, XML to images or video clips that can be accessible by a simple HTTP interface. Riak supplies an HTTP REST interface where you are able to query via URLs, headers, and verbs and receive standard HTTP response codes. REST comes from Representation State Transfer and it is used to map resources to URLs and interact with them using the CRUD set of verbs: Create (POST), Read (GET), Update (PUT) and Delete (DELETE).

Because Riak is a key-value store, it has implemented a mechanism to avoid key collisions. This mechanism comes in the form of “buckets”, where it is possible to have the same key multiple times, but only one time in each “bucket”. It is not mandatory to explicitly create a bucket if you don’t have a bucket created; putting the first value into a bucket name will create it. The Riak HTTP REST interface follows this pattern:

```
http://SERVER:PORT/riak/BUCKET/KEY
```

There are two ways of populating a Riak bucket and the first one is to know your key in advance and add the key/value pair through a PUT request and you will receive a HTTP 204 No Content response code. The second one is without specifying the key through a POST request, where Riak will generate a key for the newly added resource and will return a HTTP 201 Created response together with the generated key as part of the header under location flag. [1]

PUT

```
http://localhost:8091/riak/cars/
bmw Header: "Content-Type:
application/json"
Body: '{"model" : "520", "year"
: "2014"}'
```

Response:

```
HTTP/1.1 200 OK
```

POST

```
http://localhost:8091/riak/cars
Header: "Content-Type:
application/json"
Body: '{"model" : "A4", "year" :
"2010"}'
```

Response:

```
HTTP/1.1 201 Created
Location: /riak/cars/
9cBK3o9z1Xq7B45kJrm1S0Ma3PO
```

To retrieve the value of a resource you can simply send a GET request to the specific location. Example:

```
GET
http://localhost:8091/riak/cars/
bmw
```

To remove a key/value pair, simply send a DELETE request to the specific location and you will receive a HTTP 204 response code in case of success or a HTTP 404 in the case of error.

Example:DELETE

```
http://localhost:8091/riak/cars/bmw
```

Riak databases introduce the “Links” concept in order to support relations between keys. A “Link” is a metadata that associates multiple keys and it consist of two parts: the key where the value links to, and a string tag describing how the link relates to this value. (Link: </riak/bucket/key>; riaktag="contains")

PUT

```
http://localhost:8091/riak/dealer/bmwShop
```

Header:

```
"Content-Type:
application/json"
"Link: </riak/cars/bmw>;
riaktag="contains")"
```

```
Body: '{"cars" : "20",
"address" : "Bucharest,
Street Paris, Number 10"}'
```

MapReduce is an algorithm and a programming model to process and generate large data sets. The associated implementation breaks the problem into two parts. The first part is to specify a map() function that can process a key/value pair to generate a set of intermediate key/value pairs and the second part is a reduce() function that converts the second list of intermediate key/value pairs into one or more scalar values [9]. Following this pattern, Riak allows a system to divide tasks into smaller components and run them across a massive cluster of servers in parallel.

When it comes to availability and scalability, Riak exceeds some of the RDBMS such as MySQL and document databases like CouchDB, maintaining

replication of data on a number of its nodes, controlled by a value called the N-Value. By default for Riak the value of N is 3 for all the nodes, which means that Riak will replicate all the information for three times, but with Riak this value can be overridden on each bucket. Riak databases are designed to be used as distributed systems and by adding nodes to the cluster, the data read and write, as well as the execution of the map/reduce queries will be faster [10].

4 Redis database

Redis database is part of the Key-Value NoSQL database group that supports data structures more advanced than the Riak database, but less than a document-oriented database and it supports a set-based query operations.

It is one of the fastest NoSQL databases trading durability over speed. Redis can be considered more a toolkit of useful data structure algorithms than an ordinary member of a database group because it contains a list of processes and functionalities like a blocking queue or stack, a publisher-subscriber system and a list of configurable features as expiry policies, durability levels and replication options [1].

In Redis, the operations to create and update data are made using the SET and MSET keywords. The syntax follows this patterns: SET <key> <value> or MSET <key1> <value1> <key2> <value2>. MSET keyword is used to specify a multiple set operation provided by Redis for reducing the traffic. In the case of successful adding or updating the data, the Redis server will respond with an “OK” message. Example: “SET bmw 320i” vs “MSET bmw 320i audi A4”

For data retrieval we have the counterpart keywords GET and MGET using the following syntax: “GET <key>” vs “MGET <key1> <key2>”

Redis can store not only string text values but also numeric ones and will recognize integers and will provide some simple operations for them, like INCR/INCRBY

(increment / increment by) and DECR/DECRBY (decrement /decrement by).

In comparison to Riak, the previous key-value database type presented, Redis add the transaction concept using the MULTI block atomic commands that offer the possibility to execute multiple operations like SET or INCR in a single block that will complete either successfully or not at all. Different from the traditional transaction concept from RDBMS, in Redis when it is decided to stop a transaction with the DISCARD command there will be no rollback triggered and no reverts in the database because the commands will not have been executed. The effects are the same, even though there is a different mechanism (transaction rollback vs operation cancellation).

Redis popularity does not rise from running operations with simple types like text strings or integers, but from processing operations with complex data types as lists, hashes, sets and sorted sets over a huge number of values up to 2^{32} elements per each key. Hashes can take

any number of key/value pairs and help to avoid storing data with artificial key prefixes. In the case of hashes, all the commands are prefixed by the H character. To create a hash that contains key/value pairs, run the HMSET command as it follows: HMSET <hash> <key1> <value1> <key2> <value2>...<keyN> <valueN>. To retrieve all values from a hash, run: HVALS <hash> and to check all the keys, run HKEYS <hash>. To get a single value, run: HGET <hash> <key>. Lists contain multiple ordered values that can be stored and retrieved like FIFO (First in, First out) in the case of queues or like LIFO (Last in, First out) in the case of stack. It also has specific insert operations, for example insert on the right (end) of a list (RPUSH), insert on the left (begin) of a list or insert in the middle of a list. Sets are represented by unordered collections that do not have duplicated values and are an excellent choice for running complex operations between more key values, as unions or intersections [1].

5 Comparison between Riak and Redis databases

Table 1. Summary of Riak and Redis characteristics [11] [12] [13]

Characteristic	Riak	Redis
Official Product Name	Riak	Redis
Company/Maintainer/Builder	Basho Technologies (http://docs.basho.com/)	Salvatore Sanfilippo (http://redis.io/)
License	Apache	BSD
Protocols	HTTP RESTful and custom binary	Telnet-like Proprietary
Replication/Clustering	Masterless	Master / Slave Replication
Language/Frameworks	Erlang / C	C/C++
Key Feature	Fault tolerant	Very fast
Category	Database	Database In-Memory Data Management
Database model	Key-Value Schema-less	Key-Value Schema-less Publish/Subscriber
Query language	HTTP JavaScript	API calls

	REST Erlang	
Data types	Binary/Data structures/JSON	Data structures
MapReduce	Yes	No
ACID properties	CID (Consistency, Isolation and Durability)	ACID(Atomicity, Consistency, Isolation and Durability)
Transactions	No	Yes
Server operating systems	Linux, OS X,	BSD, Linux, OS X, Windows
Has hashes, sets and lists	No	Yes
Buckets	Yes	No
Best for	High availability, Partition tolerance, Persistence	For rapidly changing data, Frequently written, rarely read statistical data

Table 1 describes the comparisons between the main characteristics of two key-value database types: Riak vs Redis. It can be observed the programming language they are written in, Erlang/C for Riak and C/C++ for Redis, the main protocols used to communicate; Riak is using a more friendly HTTP RESTful interface. The main features of both databases are presented, Riak excels at high availability, partition tolerance and on the other hand Redis is well known as very fast database used in scenarios where we have rapidly changing data.

5 Conclusions

Even if NoSQL databases have been present in our business from some years they are still in a continuing development process, mainly because of the problem's that the real world is having managing real-time data flows that are evolving every day. We are living in some interesting days where many new products, devices are invented each day and devices produce new data with some new structures that have to be stored in some database. This structures will probably be stored in a well-known NoSQL database or a new model will be developed to fulfill the needs of the modern life.

The key-value model of NoSQL databases can resolve most of the actual common problems to store data, provide functionalities like schema-less, MapReduce and high availability. Redis and Riak databases are two of the most popular key-value NoSQL databases. Redis is the type that you choose when you need to move data with fast speed but it is not the ideal solution when you store data. On the other hand, Riak is not so fast but is fault tolerant, maintains the integrity of data and supports tuning for writes and reads [11]. When choosing the right type of key-value database you have to analyze first the business requirements that you need to fulfill and then properly analyze the unique functionalities of each type of database.

References

- [1] Eric Redmond, Jim R. Wilson. "Seven Databases in Seven Weeks, A Guide to Modern Databases and the NoSQL Movement". Dallas, Texas, North Carolina: The Pragmatic Bookshelf, May 2012. pp. 1-7, 51-99,261-307. ISBN-13:978-1-93435-692-0.
- [2] A.B.M. Moniruzzaman, Syed Akther Hossain, *NoSQL Database:*

- New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison*, International Journal of Database Theory and Application Vol. 6, No. 4. 2013.
- [3] Veronika Abramova , Jorge Bernardino, Pedro Furtado, *Experimental Evaluation Of NoSQL Databases*, International Journal of Database Management Systems (IJDMMS) Vol.6, No.3, June 2014
- [4] A SHORT HISTORY OF DATABASES: FROM RDBMS TO NOSQL & BEYOND. www.3pillarglobal.com. [Read: 16 April 2016.] <http://www.3pillarglobal.com/insights/short-history-databases-rdbms-nosql-beyond>.
- [5] NOSQL DATABASES. www.sqrrl.com. [Read: 15 April 2016.] <https://sqrrl.com/product/nosql/>
- [6] Mahdi Atawneh, "01 NoSql and multi model database", <http://www.slideshare.net/MahdiAtawneh>. [Read: 15 April 2016.] <http://www.slideshare.net/MahdiAtawneh/01-nosql-and-multi-model-database>
- [7] Sandip Shinde, "What is Wide Column Stores?", SQL Server Business Intelligence and Big Data, [Read: 16 April 2016.], <https://bi-bigdata.com/2013/01/13/what-is-wide-column-stores/>
- [8] Sandip Shinde, "What is Graph Databases?", SQL Server Business Intelligence and Big Data, [Read: 16 April 2016.], <https://bi-bigdata.com/2013/01/14/what-is-graph-databases/>
- [9] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Google Research Publications*, <http://research.google.com/archive/mapreduce.html>
- [10] Yousaf Muhammad, "Evaluation and Implementation of Distributed NoSQL Database for MMO Gaming Environment", October 2011, Department of Information Technology, <http://www.diva-portal.org/smash/get/diva2:447210/FULLTEXT01.pdf>
- [11] "Not So Versus, Riak Versus Redis", <https://compositecode.com/>, [Read: 16 April 2016.], <https://compositecode.com/2013/02/10/riak-redis/>
- [12] Riak vs Redis, <http://vschart.com/> [Read: 16 April 2016.], <http://vschart.com/compare/riak/vs/redis-database>
- [13] Ahmed Oussous , Fatima-Zahra Benjelloun , Ayoub Ait Lahcen , Samir Belfkih, *Comparison and Classification of NoSQL Databases for Big Data.*, [Read: 16 April 2016.], https://www.researchgate.net/profile/Ayoub_Ait_Lahcen/publication/278963532_Comparison_and_Classification_of_NoSQL_Databases_for_Big_Data/links/55880d3408ae65ae5a4dfa26.pdf



Cristian Andrei Baron has graduated the Faculty of Economic Cybernetics, Statistic and Informatics of the Bucharest University of Economic Studies in 2011. In 2013, he graduated the master program "Economic Informatics" at Faculty of Economic Cybernetics, Statistic and Informatics of the Bucharest University of Economic Studies. At present he is studying for the doctor's degree at the Academy of Economic Studies from Bucharest.