Analysis and comparison of NoSQL databases with an introduction to consistent references in Big Data storage systems

Adam Dziedzic and Jan Mulawka

Warsaw University of Technology, Nowowiejska 15/19, Warsaw, Poland

ABSTRACT

NoSQL is a new approach to data storage and manipulation. The aim of this paper is to gain more insight into NoSQL databases, as we are still in the early stages of understanding when to use them and how to use them in an appropriate way. In this submission descriptions of selected NoSQL databases are presented. Each of the databases is analysed with primary focus on its data model, data access, architecture and practical usage in real applications. Furthemore, the NoSQL databases are compared in fields of data references. The relational databases offer foreign keys, whereas NoSQL databases provide us with limited references. An intermediate model between graph theory and relational algebra which can address the problem should be created. Finally, the proposal of a new approach to the problem of inconsistent references in Big Data storage systems is introduced.

Keywords: NoSQL, Big Data, CouchDB, MongoDB, Riak, Neo4j, referential integrity, databases

1. INTRODUCTION

In recent years there has been considerable interest in new emerging databases called NoSQL or NewSQL. These new databases were created for the purpose of offering high performance, both in terms of speed and size, and high availability at the expense of loosing atomicity, consistency, isolation and durability. NoSQL can be an abbreviation for **N**ot **o**nly **SQL**. These new types of databases started offering SQL like query language, because many programmers adjusted to the simplicity and expressiveness of SQL. NoSQL term is used as an umbrella for all databases, which do not follow popular relational principles and often relate to large data sets accessed and manipulated on a web scale.[1] NoSQL is not a single product or a single technology, rather it represents a class of products and a collection of diverse, and sometimes related, concepts about data storage and manipulation.

The main difference between relational databases and NoSQL databases is that in the former, there are static data and dynamic queries, while in the latter dynamic data and static or dynamic queries can be found.[2] RDBMS requires schema of data to be precisely defined and as a result users are able to run different, dynamic SQL queries on data. Moreover, execution of SQL queries can be accelerated because knowing the data structure beforehand enables query optimization. As far as NoSQL databases are concerned, it is assumed that the schema of the data is not known in advance and data with different characteristics are stored. On the other hand, it is assumed that users can predict future queries and it is not necessary, for example, to execute ad-hoc queries.

Indubitably, relational databases remain the best solution for data that has many relations. However, partitioning is one of the main weak points of relational databases. Relational databases scale vertically, but not horizontally. One of the NoSQL datastores might be a better choice in the following cases:

- data requirements are too flexible to easily fit into the rigid schema requirements of a relational database;
- a very high-volume of reads and writes as key-value pairs is required;
- there is a need to store many or only large blobs of data;

Adam Dziedzic: E-mail: a.dziedzic@stud.elka.pw.edu.pl
Jan Mulawka: E-mail: j.mulawka@elka.pw.edu.pl, Telephone: +48 22 234 5319

Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2014, edited by Ryszard S. Romaniuk, Proc. of SPIE Vol. 9290, 92902V
 · doi: 10.1117/12.2075095

- one does not need the overhead of a relational database.

The databases selected for this study are Riak, MongoDB, CouchDB, Neo4j. They are popular representatives of the current NoSQL world, horizontally scalable, without fixed table schemas and able to provide high performance on very big data sets. All of the databases are in production use by many organizations. For example, MongoDB and CouchDB are in production use at CERN. Furthermore, each of them represents different theoretical approaches, which leads to interesting comparisons.

2. NOSQL CHARACTERISTICS

NoSQL databases have a number of distinctive characteristics. The most important and essential of them are presented in the following points:

- **Relaxing ACID properties.** Distributed NoSQL databases often do not attempt to provide atomicity, consistency, isolation and durability guarantees for all operations and transactions which are key attributes of classic relational database systems. Many operations in NoSQL databases are atomic with limited scope, for example, only at a row level (columnar databases) or a document level (document oriented databases).
- **No single point of failure.** There is no part of a system, which will stop the entire system from working if it fails. The internal architecture is fault-tolerant. Failures occur in a controlled environment and are dealt with gracefully. Single problems do not cascade through an entire server system, but stay isolated in single requests.
- **Horizontal scalability.** In contrast to relational database management systems, most NoSQL databases are designed to scale horizontally and not rely on highly available hardware. The main goal is massive scaling on demand (elasticity). Machines can be added and removed without causing the same operational efforts to perform sharding in RDBMS cluster-solutions. Some NoSQL datastores, such as MongoDB, provide automatic sharding whereas Riak even provides automatic resharding. Moreover, horizontal scaling is transparent on the application layer.
- **Flexible schema.** The schema of data in NoSQL solutions can be changed easily and gradually. In columnar databases, a new column can be added without any problems and there can be different columns for different rows in the same table. In document oriented databases, a different set of fields can be defined in documents. One can benefit from NoSQL flexible schema even on a small scale. The main aim is simplified application development and deployment.

3. ON THE ORIGINS OF NOSQL

There are at least two main sources of NoSQL development. One of them is big data and the other is key-value cache.

The main objective of NoSQL databases is to address a problem of big data. There is a good deal of web pages, a billion Facebook users, hundreds of millions of Twitter accounts, hundreds of millions of tweets per day, and billions of Google queries per day. The Large Hadron Collider (LHC) produces roughly 20 petabytes of data annually.

The question arises how much data qualifies as big data. Currently, a data set over a few terabytes is classified as big data.[1] This is typically the size where the data set is large enough to start spanning multiple storage units. Many NoSQL databases were created to face the following challenges posed by big data:

- efficiently storing and accessing large amounts of data (as well as making backups of them);
- while running immensely parallel processes to manipulate large data sets, it must be guaranteed that a system can recover gracefully from any failures during such a run;
- managing the continuously evolving schema and metadata for semi-structured and un-structured data generated by diverse sources.

Recently, programmers started using a memcached system to store data in RAM. The data access to this cache was written using only a key-value interface. It occurred that some types of data required only this kind of access and programmers started creating databases, which used a key-value interface for persistent storage. This was a special case when an abstraction layer of SQL language was removed, because a key-value type of access was sufficient. The key-value databases required more work from programmers on the application layer, but offered more flexibility and control over them in comparison with relational databases.

4. CLASSIFICATION OF NOSQL DATABASES

There are many NoSQL databases which have been created to fit specific requirements regarding scalability performance, maintainance and feature-set. Some of these databases have taken up ideas from either Amazons Dynamo or Googles Bigtable or a combination of both. Creators of CouchDB database used existing ideas (mainly from Lotus Notes) towards modern web technologies. Others have pursued totally different approaches like Neo4j (graph database).

The non-relational databases establish a variety of approaches regarding the non-functional requirements and the feature-set. NoSQL databases are classified in this paper by their datastructure, following the taxonomies of Yen, North and Cattell, which is also shared by most other sources specified in the bibliography.[3] The classes of NoSQL databases used in this paper are: key-value stores, document databases, columnar databases and graph databases.

4.1 Key-value stores

The key-value store has the simplest model. As the name implies, the database pairs keys to values in the same way that a map (or hash-table) does in programming languages. Some key-value store implementations permit complex value types such as hash tables or lists, but this is not required. Other implementations provide a means of iterating through the keys, but this is an added-value as well. Because the key-value stores demand so little, databases of this kind can be highly efficient in a number of scenarios but can be inefficient when one is interested in requesting or updating part of the value associated with a certain key. Furthermore, it is difficult to build complex data structures on top of key-value stores, serve complex queries to this simple interface or aggregate data. Many open source options of key-value stores are available. Some of the most popular are: Riak, Redis, MemcacheDB, Scalaris.

4.2 Document databases

The document, or document-oriented, databases are so named because documents are used to store data instead of rows. A document is a set of key-value pairs, with a unique ID field and values that may be any of a variety of types, including maps, tables, numbers and strings. Documents can contain nested structures. They exhibit a high degree of flexibility, allowing for variable domains. Document databases are considered the next step from key-value stores as they allow nested values. They permit the querying of data structures more efficiently than key-value stores as they do not necessarily return the whole values when requesting a key.

Different document databases take different approaches with respect to indexing, ad-hoc querying, replication, consistency and other design decisions. Choosing wisely between them requires an understanding of these differences and how they impact a particular use case. The two major open source players in the document database market are MongoDB and CouchDB.

4.3 Columnar databases

The columnar, or column-oriented, databases so named because the important aspect of their design is that data from a given group of columns (in the two-dimensional table sense) is stored together. By contrast, a row-oriented database (like an RDBM) keeps information about a row together. The difference may seem inconsequential, but the impact of this design decision runs deep. In column-oriented databases, adding columns is quite inexpensive and is done on a row-by-row basis. Each row can have a different set of columns, or none at all, allowing tables to remain sparse without incurring a storage cost for null values. With respect to structure, columnar databases are in between relational databases and key-value stores. The three most important columnar databases are: Bigtable, HBase and Cassandra.

4.4 Graph databases

The Graph databases are a new solution and worth considering in special cases like modelling connections between devices. For example, the Neo4j graph database has a unique data model as objects and their relationships are modelled and persisted as nodes and edges of a graph. Queries fitting this model might be faster than corresponding queries in the other aforementioned stores.

4.5 Other non-relational databases

Some classifications and taxonomies of NoSQL databases include object-oriented and hierarchical databases. Object-oriented and hierarchical databases were proposed many years ago. The main problem with object-oriented databases is their poor performance. Hierarchical databases are still used, for example, for tree-structured data like location information.

5. RIAK

The Riak database was inspired by the Dynamo database*. Riak is primarily focused on scalability and high availability. The Basho Technologies company[4] which created the Riak database, can provide users with additional support. The core Riak database is an open source database, but advanced features like masterless multi-site replication, SNMP monitoring (database remote monitoring), JMX integration, consulting and services for organisations must be paid for as part of an enterprise version.

Riak is classified generally as a key-value datastore (rarely as a document datastore). Values can be of any type, including string, text in JSON, XML format and blob. Data can be accessed via an HTTP interface. The Riak database is fault tolerant and without a single point of failure. The database installed in a cluster continues to operate even when a server is added, removed or crashes. Riak is highly available, especially for write operations. It is a web oriented database. The main weakness of Riak is a lack of ad-hoc queries.

Fundamentally, Riak is a key-value store. Groups of keys are called buckets, whose meaning is similar to namespaces. Buckets are created implicitly when a new value is inserted into a bucket. Riak allows per-bucket configuration for things like replication factor and pre/post-commit actions, so called hooks. The pre commit actions are used to validate or transform incoming data. The post commit actions can be executed after a commit is successful and are used to log data or send an email. The pre commit actions can be written in JavaScript or Erlang language, while post commit actions can be written only in Erlang. The basic data access type to a Riak database is via a REST API. Riak links HTTP URLs to resources (values), uses HTTP methods (POST, READ, UPDATE, DELETE) for CRUD operations, and applies HTTP headers and error codes.

Data stored in Riak can be linked explicitly using links. An example of a link is as follows:
`Link: </riak/bucket/key>; riaktag=\"name_of_the_link\"` A link points to a key (`</riak/bucket/key>`) and names the relation (`names_of_the_link`). Links are metadata in a Riak database which establish one-way relationships between objects. They can be used to loosely model graph-like relationships between objects. It is crucial to indicate that this is only a declaration and not any form of a foreign key. A full exemplar of link usage is as follows:

```
curl v X PUT \
http://localhost:8091/riak/racks/1 \
> H "ContentType : application/json " \
> H " Link : </riak/pdus/1>; \
>      riaktag =\"contains \" " \
> d  { "name" : " rack1 ", \
         "power_consumption" : " 5 " }
```

A new RACK device[5] is added which contains a PDU. Both the PDU and the RACK device's ID value is equal to 1. The PDU device is in PDU-s bucket. Links allow clients to only go in one direction. If there is a link in

*Dynamo database was created by Amazon and was one of the first NoSQL databases.

an object A to an object B, then the client can go from A to B, but not the other way around. Links can be set on both sides of a relationship.

Riak does not control the consistency of links and data. For example, it is possible to add a RACK with a link to a non-existing PDU. Then, a PDU can be added and removed. The link does not check the existence of the linked value. A link is only a declarative reference.

Link walking is a special approach to accessing connected data. For example, it can be declared that a RACK device contains a PDU device and that a SERVER is connected to the PDU. Link walking can be used to create a graph-like structure in Riak database.

The guidelines on suitable and unsuitable use cases for key-value stores can be found in.[6] There are many real applications which use Riak database as their backend. Two of them are described in the following subsections.

5.1 Voxer

Voxer is a Walkie Talkie application for smartphones. Voxer enables the sending of instant audio, text and photo messages. Voxer is available for iOS and Android devices. The challenge in the case of the Voxer application was to find a database solution that could provide extreme scalability and continuous write availability. To solve this problem some relational databases with ORM mapping were tried, but a scalable and highly available solution was required. CouchDB was also applied, but it was too slow. MongoDB was much faster than CouchDB, but scalability and availability were not sufficiently supported (adding or removing a node from a cluster can be cumbersome). Cassandra was too complicated either. In the end, the Riak database was the best fit to the requirements because it provides a high available storage system and linear scalability.

It should be mentioned that Riak data storage in Voxer is divided into three parts: user, media and timelines. User is a simple part with a unique ID for each user and specific information about the user (profile and preferences). Media storage part is distributed to more than thirty nodes and this is where Riak fulfils the scalability requirements. Timelines is a complicated part, also supported by other NoSQL databases, like Redis.[7]

5.2 Bump

Bump is an interesting application for smartphones (for iPhone or Android) to share contact information, photos, videos, applications and files by gently bumping two phones together. Almost all user data (profile and preferences) are stored in Riak. A program written in Haskell operates as an interface between the application nodes and the database, which exclusively deals with conflict resolution and sibling mergers. Every Riak interaction is done through this tool, which guarantees that the application nodes always see consistent data.

6. MONGODB

There are two common reasons for using MongoDB. The first is its schema-less collections of documents and second is its inherently good performance and scalability. MongoDBs strength lies in high performance, ease of use, easy data access, ad-hoc queries and an ability to handle data sets both large and small. However, the name Mongo comes from huMONGOus and it means that the core goal of this database is to store not a small but a huge amount of data. MongoDB bridges the gap between the powerful queryability of a relational database and the distributed nature of many NoSQL databases.

MongoDB is a JSON document database. Each document has a unique identifier ID, which can be generated automatically (ObjectId type) or created by a user. The ObjectId type of ID field is similar to a SEQUENCE object for primary keys or auto-increment columns in an Oracle relational database. The size of an ObjectId type is always 12 bytes. The ObjectId is composed of a timestamp (4 bytes), client machine ID (3 bytes), client process ID (2 bytes), and an incremented counter (3 bytes). This design of an ObjectId identifier is the result of MongoDB's distributed nature, because each process on every machine can handle its own ID generation without colliding with other mongod instances.

MongoDB introduces a notion of collections, where documents of similar attributes or meaning are stored together. This approach mimics reality, where different documents are stored in different folders. Collections are also analogous to tables in relational databases, but are schema-less, which allows for the storage of even

disparate documents types within the same collection. Thus, a modification of a collection in MongoDB is easy, whereas a schema change (for example a new column in a table) requires complicated data migration in relational databases. Creating a collection in MongoDB requires the addition of a document to the collection. Interestingly, a database is not created until a new document is added to a collection which is an example of lazy initialization.

Because of MongoDB's distributed nature, joins are rather inefficient. Nevertheless, there are instances when documents have to reference each other. MongoDB recommends using manual references, but offers an alternative in the form of DBRefs. These are very simple references, which support references from documents in one collection to documents in another collection. We can easily delete a referenced document without a warning being displayed. Thus, there exist so-called dangling references. We can use fancy SQL-like queries in MongoDB, but this functionality works only for one node and is not supported for a cluster setup. DBRef consists of ref, which is a name of a collection, id is an identification of a document and db is a name of a database. For example, we can store information about user sessions and maintain a reference to user documents that are stored in an HR database and the user's collection as illustrated by the following code:

```
{ // user's session
  "_id" : ObjectId("5126bbf64aed4daf9e2ab771"),
  "user" : {
              "$ref" : "users",
              "$id"  : ObjectId("51002b67bcc325feed0be4c0"),
              "$db"  : "HR"
            }
}
```

MongoDB does not define transactional integrity or isolation levels during concurrent operations, for example, there can be a read of phantoms. Write operations are atomic on the level of a single document. No single write operation can atomically affect more than one document. Although there is an isolation operator, which isolates a write operation that affects multiple documents from other write operations, it does not work in sharded clusters.

By default, in MongoDB the data inserts and updates are recorded on a disk at regular intervals. Any failure between two synchronizations can lead to inconsistency and data loss. The frequency of a flush to a disk can be increased, but it comes at the expense of performance.

There are some huge production MongoDB deployments (often called Mongo) at Foursquare, bit.ly, and CERN. Nevertheless, MongoDB is in wide use.

6.1 CERN

NoSQL databases are used on a few projects in CMS experiment at CERN. MonogDB is a foundation for DAS (Data Aggregation System). The system is used for data discovery and was built due to the significant growth of meta-data in different databases. The DAS system can be described also as a dynamic cache in front of CMS data services which fetches and aggregates data on demand upon user queries and hides complexity of the data access. It provides a search interface across distributed relational and non-relational data-services and aggregates information from them.[8]

MongoDB is suitable for an ever-growing class of web projects, especially start-ups with large-scale data storage requirements, but very little budget to buy high efficient servers and scale vertically. Thanks to the lack of structured schema, MongoDB can evolve with a data model and scale servers horizontally. There are libraries in many languages for MongoDB that resembles the use of ORM mappers, especially for Python language and the Django framework. However, the latest news from start-ups shows that they migrate from MongoDB to other NoSQL database, such as Riak 5.2.

7. COUCHDB

CouchDB is a document database written in Erlang language. Couch is an acronym for Cluster of unreliable commodity hardware.[9] CouchDB was originally written in C++, but the project moved to the Erlang OTP platform for its emphasis on fault tolerance and availability. CouchDB can be defined as a local web platform which is designed around a use case where an application runs close to a user (on a person's device, on a local network or in a browser plugin). CouchDB was designed for offline replication. This is similar to the Lotus Notes philosophy, in which data should migrate with users. CouchDB introduces many innovative features, such as incremental MapReduce, changes to API and master-master replication.

CouchDB is a document and web oriented database. It stores a flat collection of documents in JSON format and uses incremental MapReduce functions written in JavaScript to access data. There are no built-in tools to group documents and the ad-hoc queries should not be used in production mode.

The access to data in CouchDB can be divided into two parts. REST-ful API constitutes the first part, which allows users to access all of the documents by their _id-s and query the defined views. The second part comprises views. A user can create views in special design documents.[10] The views allow the selection of many documents, the choosing of specific fields and the aggregating of some values. The REST API is built in CouchDB database, so users do not have to create their own API. However, in this case the CouchDB database has to be put on a private network due to the simple security measurements it provides.[8]

There are no references in CouchDB. Thus, a user has to implement the references in the application. CouchDB stores documents internally in B+ tree and append-only files, which means that data is never over-written and a database file is in a consistent state at each time. Because of the append-only file, writes do not block reads and vice versa. There is always an up to date snapshot of the data. There can be many reads at the same time, but only one write operation. CouchDB stores many versions of the same data, but this is only an artefact, which is used internally by CouchDB for replication and update conflict resolution purposes (MVCC - Multi Version Concurrency Control). Therefore it should not be used by a user. CouchDB does not remove obsolete versions of documents automatically, so a compaction of the database has to be done on a regular basis.

7.1 CouchDB at CERN for CMS experiment

CERN uses CouchDB for CMS experiment, because of its replication features, horizontal scaling possibilities and map-reduce operations. It eliminates issues with schema evolution. CouchDB is used as a storage engine for data management and workflow system. The system is a job submission and execution engine for CMS production machinery, which dispatches and manages jobs across CMS tier centres.[8] There are three tiers (a set of computer centres) in a computing grid, to which a source of information is provided by CERN (mainly data from LHC - Large Hadron Collider).

The initial implementation was built on top of MySQL relational database. The problem was with distributed MySQL servers and the lack of a horizontal scalability feature. Then, different parts of the system were migrated to a CouchDB database.

Currently, the replication rate estimates a few documents per second between different WMAgents. After six months of running, the busiest CouchDB node contains more than six million documents (about 300 GB in size) and is constantly growing.

8. NEO4J

Neo4j is one of the most popular graph databases. This database concentrates on relationships between values and is best suited for highly variable data. The nodes as well as edges can hold data as a set of key-value pairs. There are several ways to access the Neo4j database practically, for example, Java, REST, Cypher, Ruby and Gremlin. Gremlin is a graph traversal and declarative domain-specific language which was written in Groovy. Gremlin operations are executed as a series of pipes. This is a collection-oriented traversal method. On the other hand, the Cypher language is based on pattern matching and a SQL-like syntax.[11] Neo4j offers indexing as a separate service, so queries based on indexes are not executed in a standard way. A full-text search inverted index can be created as can hash like index. Interestingly, there are transactions in Neo4j, but the type of the

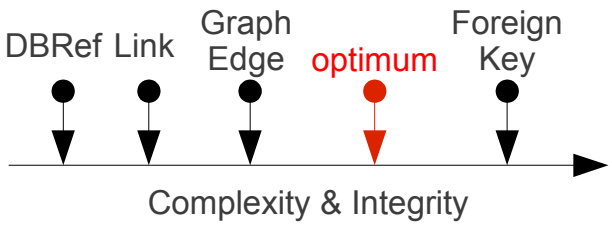

Figure 1. Review of references in NoSQL databases with a proposal of an optimal reference

transaction is rather confined. The operations that should be included in a transaction are stored in a transaction state object and only when a `Transaction.finish()` method is invoked can the transaction be either committed or rolled back.[12]

The references in Neo4j are based on edges. The main aspect of Neo4j is that there are no dangling references, because when we remove a node, all its edges are removed as well. For example, we can add a RACK device as a node and next a PDU device as another node. The PDU device should be connected to the RACK, thus an edge is created between those two nodes. Finally, the RACK node is removed, which triggers a removal of its edges, including the connection between the RACK and the PDU device. This process is presented in the following example:

```
gremlin> rack = g.addVertex([name: 'Rack']) ==>v[2]
gremlin> pdu = g.addVertex([name: 'Pdu']) ==>v[5]
gremlin> g.addEdge(pdu, rack, 'connected_to')
gremlin> g.E ==>e[1][5-connected_to->2]
gremlin> g.removeVertex(rack)
gremlin> g.E ==>
```

Main uses of Neo4j include social, recommendations, bioinformatics, fraud detection, network management, authorization and access control, content management, and parcel routing. Graph techniques are used for operating and analyzing electronic networks.

Network management applications built by Neo4j customers include network failure and degradation analysis, quality of service mapping, OSS network inventory mapping, and network asset management. Neo4j is a natural tool for social-media websites. However, companies like Facebook or LinkedIn devised their own graph databases since in these cases their businesses are based on the tool and even greater availability is required, besides which they need to store huge amounts of data.

9. CONCLUSIONS

Having analyzed different references in NoSQL databases it follows that a new model for references has to be built. Our idea stems from an example of an application that has to be created to store information on devices in a computer centre. Let us consider a UPS and a RACK device. A UPS provides power to RACK-s in case of a power cut. Thus, we have to know when there is no connection between a RACK and a UPS. We should at least be informed that a RACK was disconnected from a UPS, for instance, when a UPS is removed. One way to solve the problem of inconsistent references without much burden is to implement a mechanism similar to CouchDB's mechanism for monitoring a database for changes and receiving updates instantly (notification method). This probably would be sufficient for our requirements as the monitoring of devices in a data centre is ongoing. The monitoring for specific actions of connecting and disconnecting devices could be handled gracefully using notifications. A problem with more sophisticated solutions for this issue, which could be closer to a typical foreign key, would be the generation of a huge number of messages. The optimal solution is somewhere between

an edge in a graph and a foreign key, which is depicted in Figure 1. Our proposal is to investigate the notification mechanism more deeply for references in Big Data storage systems as well as other ways to mitigate the problems of inconsistency in NoSQL.

REFERENCES

[1] Tiwari, S., [*Professional NoSQL*], John Wiley & Sons, Inc., Indianapolis (2011).

[2] Plugge, E., Hawkins, T., and Membrey, P., [*The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*], Apress (2010).

[3] Christof Strauch, [*NoSQL databases. Selected Topics on Software-Technology Ultra-Large Scale Sites*], Stuttgart Media University (2011).

[4] Basho Technologies Inc., "Riak NoSQL database: http://www.basho.com," (2013).

[5] Jayaswal, K., [*Administering Data Centers: Servers, Storage, and Voice over IP*], ch. Power Distribution in a Data Center, 71–78, Wiley (2006).

[6] Pramod J. Sadalage, M. F., [*NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*], Addison-Wesley Professional (2012).

[7] Voxer, "Voxer about databases: http://vimeo.com/44498491," (2012).

[8] V. Kuznetsov, D. Evans, S. Metson, [*Life in extra dimensions of database world or penetration of NoSQL in HEP community.*], CERN (2012).

[9] Noah Slater J. Chris Anderson, J. L., [*CouchDB: The Definitive Guide*], O′Reilly Media (2010).

[10] Bradley Holt, [*Writing and Querying MapReduce Views in CouchDB.*], O′Reilly Media (2011).

[11] Eric Redmond, Jim R. Wilson, [*Seven Databases in Seven Weeks. A guide to modern databases and the NoSQL movement.*], The pragmatic programmers (2012).

[12] Tobias Lindaaker, "An overview of Neo4j Internals," (2012).