Comparison and Classification of NoSQL Databases for Big Data

Ahmed Oussous[1], Fatima-Zahra Benjelloun[1], Ayoub Ait Lahcen[1,2], Samir Belfkih[1]

[1]Laboratoire Génie des Systèmes, ENSA, Ibn Tofail University, Kenitra, Morocco
[2]LRIT, Unité associée au CNRST URAC 29, Mohammed V University, Rabat, Morocco
oussous.ah@gmail.com, fatimzara.benjelloun@gmail.com, {ayoub.aitlahcen, samir.belfkih}@univ-ibntofail.ac.ma

***Abstract*—NoSQL solutions have been created to respond to many issues encountered when dealing with some specific applications like those of Big Data (e.g., storage of very large data sets, the need of cheaper storage or the need of less management overhead in the Cloud). In fact, the traditional RDMS ensures data integrity and transaction consistency. But, this is at the cost of a rigid storage schema and a complex management. Certainly, data integrity and consistency are required in many cases like in financial applications but they are not always needed. The goal of this article is to establish a precise picture about NoSQLs evolution and mechanisms as well as the advantages and disadvantages of the main NoSQL data models and frameworks. For this purpose, first, a deep comparison between SQL and NoSQL databases is presented. Many criteria are examined such as: scalability, performance, consistency, security, analytical capabilities and fault-tolerance mechanisms. Second, the four major types of NoSQL databases are defined and compared: key-value stores, document databases, column-oriented databases and graph databases. Third, we compare for each NoSQL data model the main available technical solutions.**

***Index Terms*—NoSQL, Key-Value Databases, Document Databases, Column-Oriented Databases, Graph Databases, Big Data**

I. Introduction

NoSQL solutions were not created for the same purposes as SQL. While SQL is mainly dedicated for structured data and to handle transactions, NoSQL solutions were created to resolve the storage problems of massive unstructured datasets. In fact, there are advantages and disadvantages of both solutions.

[1] outlines that NoSQL have been developed long time ago before its fully adoption. [2] argues that the term NoSQL was first used in 1998 for relational databases that do not use SQL and used afterward in 2009 for a conference about non-relational databases in San Francisco. Indeed, the proliferation of clouds especially Databases as service (DBaas) and the urgent need for rapid, scalable and cheaper databases to handle Big Data encouraged the spread of NoSQL databases.

[2] adds others motives that encouraged NoSQL proliferation. That is the need to store data in a simpler structure to support object-oriented principles while avoiding expensive object-relational mapping. So NoSQL was an answer to fulfill the requirements of simple applications that are not complex. Another trend is the proliferation of web technologies and Cloud Computing that need low administration overhead and high scalability. There is also the movement in programming languages and development frameworks that tried to hide the complexity of SQL and the relational databases to offer more flexible and convenient technologies (e.g., Ruby, Java Persistant API, ADO.net, etc.).

The goal of this article is to establish a precise picture about NoSQLs evolution and mechanisms as well as the advantages and disadvantages of the main NoSQL data models and frameworks. For this purpose, first, a deep comparison between SQL and NoSQL databases is presented. Second, the four major types of NoSQL databases are defined and compared. Third, we present for each NoSQL data model the main available technical solutions.

II. Comparing SQL and No-SQL databases features

A. Scalability, performance and flexibility

There are three main issues encountered by RDMS when dealing with Big Data and some web applications. This includes 1- Scale out data, 2- Performance of single servers and 3- Rigid schema design.

SQL databases are vertically scalable [3]. Indeed, to handle increasing load, users have to increase the capacity and the performance inside a single server. This is achieved by increasing for example the capacity of the CPU, the RAM, or the SSD of the dedicated database server. However, sharding multiple tables across large clusters or grids is expensive and complex.

On the contrary, NoSQL databases are horizontally scalable. So, to handle large data volumes, users have just to add servers to the NoSQL database infrastructure. Consequently, system scalability is easier and cheaper to achieve using NoSQL.

On the other hand, relational databases require a predefined data model and structured data. They offer advanced functionality to manage, update and query data using SQL. They have various benefits such us preserving integrity, consistency and reliability of data and transactions. There is no doubt that relational databases ensure more reliability in comparison to NoSQL databases. SQL databases save the reliability and the integrity of data and transactions by respecting ACID properties (Atomicity, consistencey, Isolation, Durability).

However, ensuring ACID properties is hard to achieve in the case of huge growing data sets. That is why NoSQL databases rely instead on BASE principals (Basically Avaliable, Soft

state, Eventually consistent). Thus, they offer a flexible architecture to handle not only structured data but also unstructured and semi-structured data. Users can easily perform frequent code pushes and quick iterations.

It is worth mentioning that both ACID and BASE properties are derived from CAP theorem (Consistency, Availability, Partition tolerance). BASE principles are more flexible than ACID principles whereas ACID properties ensure more consistency and transaction reliability. However, those two qualities are achieved as the cost of performance and important investments. Thus, depending on the use case and business needs, users have to analyze their needs in term of flexibility and performance. They can choose either relational database to guarantee consistency through ACID properties or NoSQL databases when flexibility and performance are privileged to handle large data sets and to manage multiple servers in a cluster, even if flexibility means less integrity [4].

In addition to all that, relational databases lack efficiency when dealing with Big Data. In fact, the performance of relational databases tends to decrease as data volume increases especially when dealing with semi-structured data in large warehouses. In addition, they require important investment when having to increase scalability (e.g., adding servers to store and process large data sets requires purchasing additional licenses). Furthermore, the increasing need for real-time analysis of large evolving heterogeneous data volumes adds another level of difficulty [1]. In addition row storage model of RDBMS are less rapid than column stores (e.g., statistical processing is slow in RDMS). Some researches propose upgrades to RDMS to deal with this issues. [5] argue that RDMS should incorporate array storage and be extended to include matrices and mathematical libraries. Others, experts support the promise of NoSQL databases and schema free databases (e.g., graphs or object-oriented databases).

Unlike RDMS, NoSQL databases were adapted and enhanced to provide scalability, performance, flexibility needed for Big Data use cases. For instance, [2] reported that billion data can be injected per day in the column-store Hypertable of Zvent, while google is able to process 20 petabytes data stored in BigTable via MapReduce. In addition, they are based on more affordable hardware and technologies compared to Relational databases. NoSQL are even privileged for some simple applications where data storage and processing do not require the advanced features of RDMS nor to ensure data integrity as for banking transactions. Thus, NoSQL enable to avoid the unnecessary complexity of relational databases [2]. For example, social media sites and big web applications do not necessarily need trustful transaction and ACID properties (e.g., update Facebook status or Tweets comments). Zero data loss, zero service interruption are not crucial in those cases. Furthermore, implementing ACID properties of RDMS can be expensive compared to the utility of social media [2].

For all the mentioned reasons, relational databases are not suitable for Cloud environment. In fact, RDMS have a limited scalability and rely on ACID properties. Thus, they cannot support semi-structured and unstructured very large data sets. On the contrary, NoSQL databases provide a better availability, scalability, performance and flexibility. They can handle all types of data (structured, semi-structured and unstructured data).

Furthermore, Change management is very difficult to handle in relational databases. Users have to define the database schema before data injection. In addition, any change in database schema or tables should be studied carefully. Otherwise, such changes can cause service failure, reduce performance or may require maintenance and additional investment to adapt application modules.

On the contrary, NoSQL Databases enable an easy change management. In fact, there is no need to specify in advance rigid database schema. This gives users flexibility to store data without a predefined schema. Furthermore, it is possible to change any time data model without affecting system or application performance. Thus users have to choose the appropriate data model and database depending on their use case.

B. Querying and analytics

Users of relational databases launch queries using the known Structured Query Language (SQL) standard. However, there is no standard to query NoSQL databases. Indeed, each NoSQL database has its unique way to manage, extract and query data. Consequently, data scientists face the challenge to understand the query language of each NoSQL database.

On the other hand, SQL databases are powerful to handle complex queries through a standardized interface. However, NoSQL databases lack performance when dealing with complex queries. Joints are difficult to achieve in NoSQL databases. Instead, NoSQL are more adequate to handle parallel computations and mathematical equations on distributed large data sets.

C. Security

While NoSQL databases offer better scalability and flexibility to handle Big Data, they have many security issues that providers and researchers are trying to solve. In fact, most NoSQL databases do not secure client
server communications and do not provide authentication nor auditing mechanisms. Couchdb is an example that offers auditing but it stores users name and password in logs files. This compromises data security. To ensure authentication, users have usually to add external components to NoSQL infrastructure.

Furthermore, while encryption of structured data is easier in relational databases, the encryption of very large unstructured data sources is difficult to achieve. Thus most of such sources are stored in clear format in NoSQL.

Therefore, we can conclude that NoSQL rise many security issues because they are not mature. For some use cases, security is important to protect valuable, confidential or sensitive large sources (e.g. health, government, system security and so on).

D. Sharding

Sharding means to partition large volumes of data across servers and virtual data nodes. NoSQL databases embraces sharding to balance the load [6] and to ensure parallel storage and processing. They offer the valuable option to add or remove servers from data layer without affecting application performance.

On the contrary, RDMS were not originally created with this purpose. Instead, sharding feature was added to RDMS. Tables are partitioned over multiple servers. Sharding is based on mapping between shards (data partitions) and data nodes that contain those shards. The mapping can either be dynamic or static. One downside of Sharding is that it does not allow joins between shards.

E. Data replication

Traditional means of data redundancy focus on data mirroring. They replicate data over target arrays at the data center or over a distant site. This method consumes a lot of storage space especially in the case of large data sets that exceeds petabytes. In fact, it is an overhead and expensive for organization to store large streams of data (data in motion) as well as big data archives using traditional means. To prevent data loss, most NoSQl databases provide automatic data replication for fault-tolerance. They replicate data across cluster servers and even across data centers. Thus, using Big Data technologies and NoSQL databases, developers do not have to worry about the complexity of the heterogeneous storage environment nor the mechanisms of parallel processing. IBM InfoSphere Data Replication provides a good example of a real-time data replication. [7] confirms that real-time data replication allow to ensure continuous high data availability in both heterogeneous and homogenous environment. Real-time data replication is crucial to perform reporting, interactive analysis and to ensure synchronized transactions. It helps to extract accurate insight, supports rapid decision-making and optimizes resources.

[8] outlines the utility of another option based on an error-correcting algorithm called Erasure Coding that is paired to object-based storage technique. Such solution is an alternative to data replication in a distributed environment. For instance, a data object (e.g., document with its metadata) is splited into segments. Each segment is encoded and cut into slices that are stored on different servers. Thus, if some slices are no more accessible due to a disk failure, organization can still reconstruct the original data. This solution reduces cost, consumes less storage and guarantees fault-tolerance repositories. However, it is not yet mature.

III. Types of NoSQL Databases

NoSQL databases can be classied using different approaches and various criteria. Some experts classify them according to their data model and most of them outline four major types of NoSQL databases, including: key-value databases (e.g., Redis, Voldermort, Riak), document databases (e.g., CouchDB, MongoDB, Simple DB), column-oriented databases (e.g., BigTables, HBase, Cassandra) and graph databases. In the following sections, we present these four major types.

A. Key-value databases

This model is implemented using a Hash Table where there is a unique key and a pointer to a particular item of data creating a key-value pair. Hash Table are suitable to lookups for simple or complex values in extremely large data sets.

Key-value databases [9] can handle very large number of records. They can support high volumes of state changes per second with millions of simultaneous users through distributed processing and storage. Key value databases rely on their in redundancy to face the loss of storage nodes and to protect applications.

They are very useful for both storing the results of analytical algorithms (such as phrase counts among massive numbers of documents) and for producing those results via reports. However, Key-value databases inherit one drawback of NoSQL databases. They do not provide any kind of traditional database capabilities. Thus, to ensure transactions atomicity or the consistency of multiple parallel transactions, users should instead rely on the application itself [10].

Another drawback is that users cannot access data by value. Indeed, it is impossible to query a keyvalue data store in order to extract all records that contain a particular set of values. As confirmed by [11], the only way to query a keyvalue database is by specifying a request by key or by a range of keys.

We compare hereafter three examples of Key-value databases: Redis, Riak and Voldemort. All of those three solutions ensure scalability and fault-tolerance and a near-liear increase in performance.

Riak is designed for highly distributed environments such as the cloud. It ensures high fault-tolerance but with less performance than Redis. So Redis is more suitable for time-critical applications because it relies on in-memory dataset for fast responses. Redis is appropriate to handle rapidly changing data such as real-time data collection from sensors and real-time communications. On the other hand, Voldermort is suitable for very large data sets such as geological data and meta-data of maps. Indeed, it can handle the storage of huge volumes without a great impact on performance [12].

Experiments showed that Redis scales when increasing data sets volumes but it does not scale with the increasing number of nodes. On the contrary, Voldemort scales when the number of nodes increases but it does not scale with the increasing size of datasets. Redis ensures a better data availability compared to Voldermort. However, both of them showed a reduced availability when having to deal with very large data sets. It has been also shown that adding nodes to Voldermore system helps to enhance its availability [12]. Table I summarizes the important features of Redis, Riak and Voldemort.

B. Document databases

Document databases [13] were designed to handle the storage and the management of large scale documents. This type of database assigns a key value to each document. Documents

TABLE I Overview of Riak, Redis and Voldermort features

K-V stores / Properties	Riak	Redis	Voldemort
Language	Erlang	C, C++	Java
Fault tolerance	Replication	Replication	Data partition, Replication, RAID repair
Data model	Buckets, Keys-Values	Data structures	Structured blob/text
Community	Apache	BSD	Apache
Protocol	Http/REST or custom binary	Telnet-like, binary safe	Http
Data storage	Bitcask, LevelDB, Volatile memory, file system	Volatile memory, File system	TSconfig, LevelDB, HDFS, GridGain
Query language	Bhttp, Javascript, REST, Erlang	API calls	API calls
Map Reduce	YES	NO	NO
Replication mode	Multi master replication	Master Slave replication	Symmetric Replication
Operating system	Cross platform	Linux Mac OS Windows	Linux, Windows, MAC OS
Best for	High availability, Partition tolerance, Persistence	For rapidly changing data, Frequently written, rarely read statistical data	Application with large requirement on data capacity

may contain multiple key-value pairs, or key-array pairs, or even nested documents. Documents are encoded in a standard data exchange format such as XML, JSON (Javascript Option Notation) or BSON (Binary JSON). Document databases are recognized as a powerful, flexible and agile tool to store Big Data.

Document databases are different from key-value stores. In fact, while key-value stores enable to search for data only by key value, document databases allow users to search for data based on the content of documents. They can query either by keys, values or examples. In fact, the encoded documents contain metadata objects, so it is possible to query data by example [10]. This gives document databases a great flexibility required by some use cases. To launch queries, users can either rely on a programming API or a query language.

On the contrary of the simple key-value stores, the value column in document databases contains semi-structured data and specifically the attribute name/value pairs. Furthermore, document databases support a flexible schema. Indeed, they allow storing hundreds of attributes in a single column of a document scheme. So rows can receive various amount and types of attributes.

As examples, CouchDB and MongoDB are open source document databases. For both, data is stored in documents with self-contained records and no intrinsic relationships. CouchDB stores data to disk by append-only files while MongoDB stores data the Memory-Mapped Storage Engine. It uses memory mapped files for all disk I/O. As an interchange format, CouchDB offers an HTTP API for both data access and administration. MongoDB provides instead a socket-based wire protocol with BSON.

For fault-tolerance, CouchDB supports both master/master and master/slave replication. Replication can be finely tuned via replication filters. On the contrary, Mongo manages replication using a form of asynchronous master/slave replication called Replica Sets.

To conclude, both have many common features such us replication for fault-tolerance and volatile memory file system for data storage. Both rely on MapReduce paradigm for data processing and have a good community support as well. However, CouchDB is not adapted to extremely changing data. In fact, while CouchDB requires to set up pre-defined queries, MongoDB is suitable for dynamic queries and ensures a better performance on Big databases. Table II summarizes the important features of CouchDB and MongoDB.

TABLE II Overview of MongoDB and CouchDB features.

DB / Properties	MongoDB	CouchDB
Language	C++	Erlang
Fault tolerance	Replication	Replication
Data model	Document oriented (BSON)	Document oriented (JSON)
Community	AGPL and others	Apache
Protocol	TCP/IP	HTTP/REST
Data storage	Volatile memory, file system	Volatile memory, file system
MapReduce	YES	YES
Replication mode	Maste-Slave replication	Multi-master replication
Best for	Dynamic queries, Defining indexes, Good performance on a big DB.	Accumulating, Occasionally changing data, Pre-defined queries to be run, Web use cases and mobile applications.

C. Wide-column databases

Wide-column databases (also called extensible record stores) represent an extension of the key-value architecture with columns. They were designed to process distributed data over a pool of infrastructure. Wide Column databases [13] are based on a hybrid approach that rely on relational databases declarative characteristics and various key-value stores schema.

Hbase, Cassandra and Accumulo are some examples of column family databases. HBase [14] is a NoSQL database management system designed to run on top of the HDFS. It is an open source project that is suitable for handling various large data sets. It is based on column-oriented key/value data model. HBase is designed to support high table-update rates and to scale out horizontally in distributed clusters. HBase provides flexible structured hosting for very large tables in a BigTable-like format. HBase provides many features such us real-time queries, natural language search, consistent access

TABLE III Overview of Hbase, Cassandra and Accumulo features.

Column stores / Properties	Hbase	Cassandra	Accumulo
Language	Java	Java	Java
Fault tolerance	Replication, Partitioning	Replication, Partitioning	Replication
Data model	BigTable	BigTable and Dynamo	BigTable
Community	Apache	Facebook	Apache
Protocol	Custom API, Thrift, Reset	Thrift	Thrift
Data storage	HDFS	Inspired by Amazon's Dynamo for storing data	HDFS
Query language	Apl calls, Reset XML, Thrift API	Apl calls, Thrift API	Java API, Thrift API, REST calls
Map Reduce	YES	YES	YES
Replication mode	Maser-Slave Replication	Master-Slave replication	Multi-master replication
Best for	Real-time access, Do bulk operation (indexing, ...)	When you write more than you read (logging), Must use Java	Access on the cell level

to Big Data sources, linear and modular scalability, automatic and configurable sharding of tables [15]. It is a popular non-relational database which is included in many Big Data solutions and data driven websites.

Cassandra [**?**] is also a popular NoSQL database for very large data sets. It is a key-value database that uses column-oriented storage, sharding by key ranges, and redundant storage. It provides scalability, read/write performances, as well as resiliency against "hot" nodes and node failures. Cassandra allows configuring settings to adjust tradeoffs preferences between consistency and availability.

Apache Accumulo [16] is distributed column store solution that provides scalability and high performance. Apache Accumulo is based on Google's BigTable design and it is built on the top of Hadoop, Zookeeper, and Thrift. It allows an access control at cell level on the BigTable. It also allows to modify key/value pairs at various points in the data management process. This is ensured through a server-side programming mechanism.

Table II summarizes the important features of Hbase, Cassandra and Accumulo.

D. Graph Databases

Both relational databases and NoSQL databases like key-value stores are not efficient when dealing with highly connected data. They lack performance and flexibility needed to process and query multiple relationships inside large data sets [17].

Even though that MapReduce paradigm paired to Hadoop framework provide scalability, fault tolerance and an easy-to-program tools for large data sets, it has been proven that some NoSQL databases like key-value stores are not always suitable for connected data and very large graphs [18].

On the contrary, graph databases are suitable to store not only information about objects but also all relationships that exist among them. They rely on schema-free graph model in order to easily model and represent connected data. Such model includes vertices (e.g., objects or items represented by nodes) and edges to represent connections between data.

To illustrate this, a graph can refer to a professional network like that in Viadeo. In this case, the vertices represent professionals while the directed edges represent links and relationships between those professionals. Each vertex is also initialized with a value. It is worth mentioning that even if graph databases save relationships, they have nothing to do with relational databases.

They are useful to store, access and analyze the strength and the nature of relationships between two or more items (e.g., How close is the relationship between two people? How far away is a taxi driver from another one or from a touristic site). Answering such questions enable to formulate valuable recommendations in many industries.

Graph databases offer for many use cases enhanced performance (they ensure lower latency in comparison to batch processing of aggregates), flexible data model (easy way to express relationships and to enrich the graph as data and business requirements get more precised) and agility (ability to evolve applications in a controlled manner aligned with Agil and Test Driven software development practices) [17]. Unlike most classes of NoSQL data store, Graph databases are not the best solutions for updating sets of data or for very large volumes of data [17].

Neo4J, ArangoDB and OrientDB are three examples of Graph databases.

Neo4J is an open-source database that stores highly connected data in a graph format rather than in tables (that are more suitable for aggregated data). In fact, data are stored in nodes connected to each other by defined relationship. Both nodes and relationships have their properties. It is written entirely in Java and supported by Neo technology. It is an embedded, disk-based, fully transactional Java persistence engine with few small jars. It provides high scalability allowing to add up to several billion of data, high availability even if data is distributed across many machines, fast queries and rapid path identification through its traversal framework. In addition, data analysts can use its human readable query language adapted for graph models. Neo4j offers also a convenient and simple access (by Rest interface or an object oriented Java API). It also provides, like relational databases, full ACID properties for reliable transactions [19]. Neo4j Spatial is a library of utilities for Neo4j that permits to add spatial indexes to already located data. It is designed to facilitate spatial operations on data (i.e., to search for specific data by region or within a defined perimeter).

ArangoDB [20] is an open source distributed and multi-purpose NoSQL database. It supports multiple data models including documents, graphs, and key-values. It is suitable for

applications that need space efficiency, high performance with convenient querying tools. Indeed, it allows to use an SQL-like query language as well as JavaScript and Ruby extensions. In fact, ArangoDB Query Language (AQL) is designed to support complex queries especially on ArangoDB data models. The storage and retrieval of data is based on collections. AQL is considered as a declarative language focusing on the results instead of how the results should be produced. Furthermore, AQL is independent of the programming language of the clients. So all the clients use the same language and syntax. In addition, it offers REST option for querying documents and it permits to query by example. It permits a vertical and horizontal sharding (i.e., to add more computation power and to shard data to many servers). It runs on different platforms such us Linux, Windows, OSX and even Raspberry Pi. It is available under Apache 2 license.

OrientDB [21] is an open source NoSQL database management system that is released under the Apache 2 License. It is written in Java so it can run on Linux, Windows and any system that supports Java. It provides the flexibility of documents as well as a good performance to handle distributed graphs. In fact, OrientDB is a document-based database that consists of ODocuments (ossibility to dynamically add and remove properties). At the same time, it allows to manage relationships as in graph databases with direct connections between records. Therefore, it emulates the property of indexfree adjacency of documents. It supports multiple modes including schema-less, schema-full and schema-mixed.Furthermore, OrientDB supports SQL as a query language with the possibility to manage graphs of connected documents. It can handle relationships without SQL joins. For fast insertions and queries, OrientDB is based on a new indexing algorithm called MVRB-Tree. OrientDB supports other features such us ACID for reliable transactions and security profiling through roles and users,

Both Neo4J and OrientDB are dedicated to support the storage of large data sets based on graphs model. This ensures scalability and performance when inserting or querying data. The main difference between those two NoSQL databases lies in the core storage. Indeed, while OrientDB is based essentially on documents as the main storage (in addition to a graph layer that supports graphs), Neo4J is based on graphs as its core storage [22]. According to some experiments [23], both Neo4J and OrientDB demonstrate comparable performance when dealing with small graphs. However, Neo4J appears to be more efficient than OrientDB when handling the storage and queries on big graphs.

IV. Conclusion

NoSQL databases provides many advantages to deal with Big Data storage, processing and querying. NoSQL are suitable for high volumes of state changes per second with millions of simultaneous distributed users. Unlike traditional stores, NoSQL databases ensure better sharding and real-time data replication at a lower cost while optimizing system resources. However, relational databases are better for structured data, complex queries, trustful transactions and high integrity. Since NoSQL solutions are not mature and are progressing at different speeds, database administrators have to choose carefully between NoSQL and relational databases according to their specific needs in terms of consistency, performances, security, scalability, costs and other non-functional criteria.

References

[1] C. Ordonez, I.-Y. Song, and C. Garcia-Alvarado, "Relational versus non-relational database systems for data warehousing," in *Proceedings of the ACM 13th international workshop on Data warehousing and OLAP*. ACM, 2010, pp. 67–68.

[2] C. Strauch, U.-L. S. Sites, and W. Kriha, "Nosql databases," *Lecture Notes, Stuttgart Media University*, 2011.

[3] M. A. Mohamed, O. G. Altrafi, and M. O. Ismail, "Relational vs. nosql databases: A survey," *International Journal of Computer and Information Technology*, vol. 03, no. 03, pp. 598–601, May 2014.

[4] P. Zikopoulos, C. Eaton *et al.*, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[5] C. Ordonez, "Can we analyze big data inside a dbms?" in *Proceedings of the sixteenth international workshop on Data warehousing and OLAP*. ACM, 2013, pp. 85–92.

[6] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[7] *Derive actionable real-time insight from your big data with data replication*. IBM Software Thought, 2014. [Online]. Available: http://public.dhe.ibm.com/common/ssi/ecm/im/en/imw14758usen/IMW14758USEN.PDF

[8] J. Moore, "How cloud storage could catch up with big data," Apr 17, 2012. [Online]. Available: http://fcw.com/Articles/2012/04/30/FEAT-BizTech-cloud-storage.aspx?Page=1

[9] A. Reeve, "Big data architectures nosql use cases for key value databases," Nov. 2013. [Online]. Available: http://infocus.emc.com/april_reeve/

[10] D. Loshin, *Big data analytics: from strategic planning to enterprise integration with tools, techniques, NoSQL, and graph*. Elsevier, 2013.

[11] M. Manoochehri, *Data Just Right: Introduction to Large-scale Data & Analytics*. Addison-Wesley, 2013.

[12] H. Feng, "Benchmarking the suitability of key-value stores for distributed scientific data," *MSc thesis in High Performance Computing, The University of Edinburgh*, 2012.

[13] A. B. M. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics - classification, characteristics and comparison." *International Journal of Database Theory and Application*, vol. 06, no. 04, 2013.

[14] D. Carstoiu, E. Lepadatu, and M. Gaspar, "Hbase-non sql database, performances evaluation." *Int. J. Adv. Comp. Techn.*, vol. 2, no. 5, pp. 42–52, 2010.

[15] Apache HBase, https://hbase.apache.org/.

[16] G. J. Halldorsson, *Apache Accumulo for Developers*. Packt Publishing Ltd, 2013.

[17] I. Robinson, J. Webber, and E. Eifrem, *Graph databases*. O'Reilly Media, Inc, 2013.

[18] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[19] K. Kaur and R. Rani, "Modeling and querying data in NoSQL databases," in *IEEE International Conference on Big Data*, Oct 2013, pp. 1–7.

[20] L. Dohmen, P. D. R. Klamma, and F. Celler, "Algorithms for large networks in the nosql database arangodb," *Bachelors thesis, RWTH Aachen, Aachen*, 2012.

[21] Y. Abubakar, T. S. Adeyi, and I. G. Auta, "Performance evaluation of nosql systems using ycsb in a resource austere environment," *Performance Evaluation*, vol. 7, no. 8, 2014.

[22] K. Barmpis and D. S. Kolovos, "Comparative analysis of data persistence technologies for large-scale models," in *Proceedings of the 2012 Extreme Modeling Workshop*. ACM, 2012, pp. 33–38.

[23] S. Beis, S. Papadopoulos, and Y. Kompatsiaris, "Benchmarking graph databases on the problem of community detection," in *New Trends in Database and Information Systems II*. Springer, 2015, pp. 3–14.