International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 4 Issue 4, April-2015, pp: (118-124), Impact Factor: 1.252, Available online at: www.erpublications.com

Comparative Study of Column Oriented NoSQL Databases on Characteristics

Alireza Jomeiri[1], Mahboubeh Shamsi[2], Elham Kazemi[3]

[1]Departmentof Engineering, Marand branch, Islamic Azad University, Marand, IRAN
[2]Department of Engineering, Qom University of Technology, Qom, IRAN
[3]Department of Computer Engineering, Abadan branch, Islamic Azad University, Abadan, IRAN

Abstract: NoSQL database, also called Not Only SQL, is an approach to data management and database design that's useful for very large sets of distributed data. The growing popularity of big data will compel many companies to use NoSQL databases instead of traditional database. Generally, there are three main types of NoSQL databases: key-value stores, column oriented databases and document based stores. Because the column oriented databases have specific characteristics and advantages compared to others, we decided to introduce the various popular column based NoSQL databases and explain their properties and functionalities. In this paper, we evaluate and compare six popular column oriented NoSQL databases. These databases are compared by CAP theorem, their persistence, concurrency controls and replication opportunities. In addition, this paper classifies NoSQL databases according to the CAP theorem. Finally, the column based NoSQL databases are separately described in detail, and extract some properties in a table to help enterprises to choose NoSQL.

Keywords: Big data, distributed database, NOSQL, column oriented, cap theorem.

Introduction

Database technology has gone through more than three decades of development. Various database systems have been proposed for different scales of datasets and diverse applications. Traditional relational database systems obviously cannot address the variety and scale challenges required by big data. Due to certain essential characteristics, including being schema free, supporting easy replication, possessing a simple API, eventual consistency and supporting a huge amount of data, the NoSQL database is becoming the standard to cope with big data problems. Generally, there are three primary types of NoSQL databases that are organized by the data model, i.e., key-value stores, column-oriented databases, and document databases [1].

Key-value stores have a simple data model in which data are stored as a key-value pair. Each of the keys is unique, and the clients put on or request values for each key. Key-value databases that have emerged in recent years have been heavily influenced by Amazon's Dynamo [2].

Column-oriented databases store and process data by column instead of by row. Both rows and columns will be split over multiple nodes to achieve scalability. The main inspiration for column-oriented databases is Google's Bigtable, which will be discussed next, followed by several derivatives [3].

Document stores support more complex data structures than key-value stores. There is no strict schema to which documents must conform, which eliminates the need of schema migration efforts [4].

NoSQL databases by definition break the paradigm constraints of traditional relational databases. From a data storage perspective, many NoSQL databases are not relational databases, but are hash databases that have key-value data format. Because of the abandonment of the powerful SQL query language, transactional consistency, and normal form constraints of relational databases, NoSQL databases can solve challenges faced by traditional relational databases to a great extent. In terms of design, they are concerned with high concurrent reading and writing of data and massive amounts of data storage. Compared with relational databases, they have a great advantage in scalability, concurrency, and fault tolerance.

Leavitt argues that non-relational models have been available for more than 50 years in forms such as object-oriented, hierarchical, and graph databases, but recently this paradigm started to attract more attention with models such as key-store, column oriented, and document-based stores. The causes for such raise in interest, according to Levitt, are better performance, capacity of handling unstructured data, and suitability for distributed environments [5].

Han et al. presented a survey of NoSQL databases with emphasis on their advantages and limitations for Cloud computing. The survey classifies NoSQL systems according to their capacity in addressing different pairs of CAP (consistency, availability, partitioning). The survey also explores the data model that the studied NoSQL systems support [6]. Hecht and Jablonski compared different NoSQL systems in regard to supported data models, types of query supported, and support for concurrency, consistency, replication, and partitioning. Hecht and Jablonski concluded that there are big differences among the features of different technologies, and there is no single system that would be the most suitable for every need. Therefore, it is important for adopters to understand the requirements of their applications and the capabilities of different systems so that the system whose features better match their needs is selected [7].

Even if NoSQL databases have already been introduced and compared in the past, no column oriented survey is available. When the complexity and size of data warehouse is continuously increasing, only Row Oriented approach is not enough to fulfill the current business needs. This is because there are more and various requirements for reporting and analytics from all business areas, there is need of increased time period of retention of data. Data warehouse stores many observations and for each of the observation there are many attributes which have to maintain. This was manageable till the large data was in the structured format. But now day's organizations are more concentrating on unstructured data storage for analysis (Like images, void/ video recording).

Unlike the traditional defines schemas of relational databases, column-based NoSQL solutions do not require a pre-structured table to work with the data. Each record comes with one or more columns containing the information and each column of each record can be different. That is why a new approach has been invented some years ago in which data is stored column wise.

The rest of this paper is organized as follows: the next section compares column base systems against row based systems. The CAP theorem and classification of NoSQL databases according to it is described in section 3. Column oriented databases are presented in section 4 and afterward the popular column based databases are summarized. In section 5, we focus on the features of the column oriented databases and highlight the advantages and disadvantages of each. In this section, the main characteristics of the databases listed in a table. The section 6 concludes the paper.

Column Based vs Row Based

Unlike the row-based systems (Key Value and Document Oriented DBs) these are as the name implies oriented to storing data in columns. Where Document Oriented DBs excel at OLTP, Column Oriented DBs excel at OLAP. Data are stored in cells grouped in columns as opposed to rows. Columns are grouped in Column Families and each can contain an essentially unlimited number of columns. Each storage block contains data from only one column. Data can be sparse, that is not all cells need to be filled and the cell-to-column organization allows for greater compression of data on the disks. Compression reduces query time since fewer read actions are required.

Moreover CODBs would be selected where queries are likely to look at similar data items on many different records, for example "find all the people with the last name Smith" can be retrieved in a single operation. Other operations like counting the number of matching records or performing math over a set of data (e.g. find the average salary of all employees at level X) can be much faster. Since these data elements will reside in single columns they can be retrieved very quickly. The CODB might be able to retrieve a single data item from all records in a single operation, contrasted to row based systems where each row would need to be read and the data items extracted. Compared to REBMS this speed increase can be in the range of 5X to 100X. Consequently CODBs are the go-to solution for OLAP applications. Some advantages of CODBs are [8]:

- Good horizontal scaling. High availability.
- Supports semi-structured data (as do Document DBs).
- Compression allows efficient access to data stored on hard disks reducing seek time and latency. Only fully in-memory databases offer faster access.
- Efficient when an aggregate value needs to be computed over many rows but where the number of rows queried is significantly smaller than the whole.
- Particularly efficient when updating all the values in a column at once as the new column can be written efficiently without touching other data columns.
- Strong for OLAP applications which call for a smaller number of highly complex queries over large quantities of data (terabytes or greater).
- Can handle logging continuous streams of data that may not require consistency guarantees (see lesson 2) as they can accommodate high volumes of writes over the distributed architecture.
- Compared to RDBMS, 5X to 100X faster query performance and 5X to 10X less disk space due to compression.
- Can span multiple data centers.

CAP Theorem

When designing a distributed system it is very important to understand the concept of CAP Theorem and NoSQL databases are no exceptions when it comes to these issues. The CAP Theorem was introduced by Dr. Brewer in a keynote addressing the trade-offs in distributed systems and was later formalized by Gilbert and Lynch. It states that in a distributed data storage system only two features out of availability, consistency and partition tolerance can be provided.

Availability means in this case that clients can always read and write data in a specific period of time. A partition tolerant distributed database is failure tolerant against temporal connection problems and allows partitions of nodes to be separated [9].

A system that is partition tolerant can only provide strong consistency with cutbacks in its availability, because it has to ensure that each write operation only finishes if the data is replicated to all necessary nodes, which may not always be possible in a distributed environment due to connection errors and other temporal hardware failures.

Therefore NoSQL databases can be classified using the CAP Theorem. The entire current NoSQL database follow the different combinations of the C, A, P from the CAP theorem (see fig. 1). Here is the brief description of three combinations CA, CP, AP:

A. Consistency with Availability

These types of data bases are mainly concerned with consistency and availability. Systems concern the CA are: the traditional relational databases, Vertica (Column oriented), Aster Data (Relational), Greenplum (Relational) and so on.

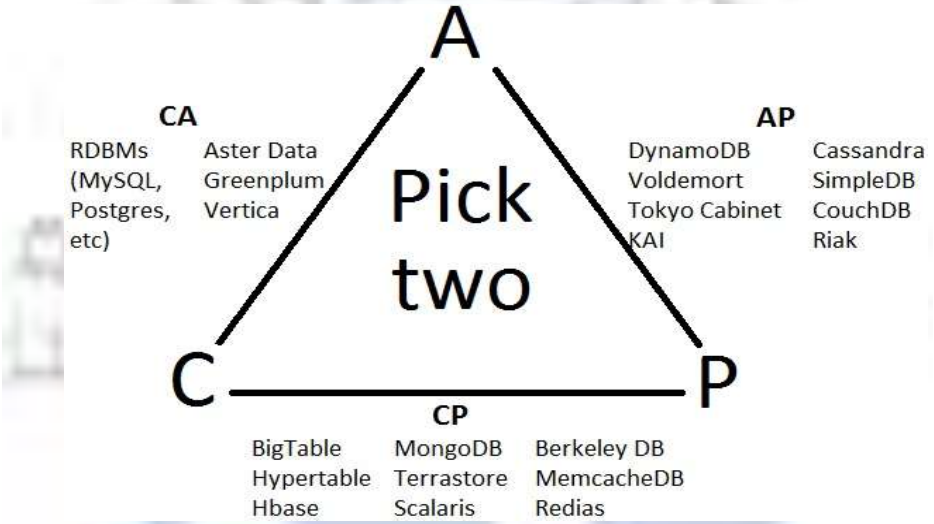

Figure 1. Representation of CAP theorem.

B. Consistency with Partition Tolerance

Such a database system stores data in the distributed nodes, but also ensures the consistency of the data, but it does not have good support for availability. Example of databases which employ the CP system are: BigTable (Column oriented), Hypertable (Column-oriented), HBase (Columnoriented), MongoDB (Document), Terrastore (Document),Redis (Key-value),Scalaris (Key-value),MemcacheDB (Key value), Berkeley DB (Key-value).

C. Availability with Partition Tolerance

Such systems ensure availability and partition tolerance primarily by achieving consistency, AP's system: DynamoDB (Column oriented), Voldemort (Key-value), Tokyo Cabinet (Keyvalue), KAI (Key-value), CouchDB (Document oriented), SimpleDB (Column oriented), Riak (Document-oriented).

Column Oriented Databases

Column-oriented or Wide-table data stores are designed to address following three areas- huge number of columns, sparse nature of data and frequent changes in schema. Unlike relational databases where rows are stored contiguously, in column-oriented databases column values are stored contiguously. This change in storage design results in better performance for some operations like aggregations, support for ad-hoc and dynamic query etc. In row-oriented databases, all columns of those rows which satisfies the where clause of the query are retrieved, which causes unnecessary disk input/output if only few columns out of all returned columns were required. These databases are also best suited for analytical purposes as they deal with only few specific columns and for compression also as data-type and range of values are fixed for each column. For each column, row-oriented storage design deals with multiple data types and limitless range of values, thus making compression less efficient overall.

Most of the columnar databases are also compatible with MapReduce framework, which speeds up processing of large amount of data by distributing the problem on large number of systems. Popular open source column-oriented databases are Hypertable, HBase and Cassandra. Hypertable and HBase are derivatives of BigTable whereas Cassandra takes its features from both BigTable and Dynamo.

1)BigTable

Bigtable is described as "a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers". It is used by over sixty projects at Google as of 2006, including web indexing, Google Earth, Google Analytics, Orkut, and Google Docs. These projects have very different data size, infrastructure and latency requirements: "from throughput-oriented batch-processing jobs to latency sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data". According to Chang et al. experience at Google shows that "Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability".

In contrast to RDBMSs, data can be indexed by Bigtable in more than one dimension—not only row- but also column-wise. A further distinguishing proposition is that Bigtable allows data to be delivered out of memory or from disk—which can be specified via configuration [10].

2)Hypertable

Hypertable is modelled after Google's Bigtable. The project's goal is "to set the open source standard for highly available, petabyte scale, database systems". Hypertable is almost completely written in C++ and relies on a distributed file system such as Apache Hadoop's HDFS as well as a distributed lock-manager. Regarding its data model it supports all abstractions available in Bigtable; in contrast to Hbase column-families with an arbitrary numbers of distinct columns are available in Hypertable. Tables are partitioned by ranges of row keys (like in Bigtable) and the resulting partitions get replicated between servers. The data representation and processing at runtime is also borrowed from Bigtable: "[updates] are done in memory and later flushed to disk". Hypertable has its own query language called HQL and exposes a native C++ as well as a Thrift API. Originally developed by Zvents Inc., it has been open-sourced under the GPL in 2007 and is sponsored by Baidu, the leading chinese search engine since 2009 [11].

3)HBase

The HBase data store is a Bigtable-clone developed in Java as a part of Apache's MapReduce-framework Hadoop, providing a "a fault-tolerant way of storing large quantities of sparse data". Like Hypertable, HBase depends on a distributed file system (HDFS) which takes the same role as GFS in the context of Bigtable. Concepts also borrowed from Bigtable are the memory and disk usage pattern with the need for compactations of immutable or append-only files, compression of data as well as bloom filters for the reduction of disk access. HBase databases can be a source of as well as a destination for MapReduce jobs executed via Hadoop. HBase exposes a native API in Java and can also be accessed via Thrift or REST. A notable usage of HBase is the real-time messaging system of Facebook built upon HBase since 2010 [12].

4)Cassandra

As a fourth data store in this paper Apache Cassandra which adopts ideas and concepts of both, Amazon's Dynamo as well as Google's Bigtable, shall be discussed. It was originally developed by Facebook and open-sourced in 2008. Lakshman describes Cassandra as a "distributed storage system for managing structured data that is designed to scale to a very large size". It "shares many design and implementation strategies with databases" but "does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format". Besides Facebook, other companies have also adopted Cassandra such as Twitter, Digg and Rackspace [13].

5)SimpleDB

Amazon SimpleDB is a highly available and flexible non-relational data store that offloads the work of database administration. Developers simply store and query data items via web services requests and SimpleDB does the rest. Unbound by the strict requirements of a relational database, Amazon SimpleDB is optimized to provide high availability and flexibility, with little or no administrative burden. Behind the scenes, SimpleDB creates and manages multiple geographically distributed replicas of your data automatically to enable high availability and data durability. The service charges you only for the resources actually consumed in storing your data and serving your requests. You can change your data model on the fly, and data is automatically indexed for you. With SimpleDB, you can focus on application development without worrying about infrastructure provisioning, high availability, software maintenance, schema and index management, or performance tuning [14].

6)DynamoDB

Amazon DynamoDB helps manage the state of services that have high reliability requirements and require tight control over the tradeoffs between availability, consistency, cost-effectiveness, and performance. DynamoDB provides a simple primary key only interface to accommodate application requirements. It uses a combination of successfully implemented techniques to accomplish scalability and availability: data is abstracted and consistently replicated through object versioning. Replica consistency is maintained during updates using a synchronization protocol that applies a quorum like technique and decentralized replica. DynamoDB provides a decentralized arrangement, keeping administration charges to

a minimum. Storage nodes are automatically added and removed without requiring manual administration or redistribution [15].

DynamoDB is a fast, fully managed NoSQL database service that makes it simple and cost-effective to store and retrieve any amount of data, and serve any level of request traffic. Its reliable throughput and single-digit millisecond latency make it a great fit for gaming, ad tech, mobile and many other applications.

Discussion

Even with so many kinds of databases, no one can be best for all workloads and scenarios, different databases make distinctive tradeoffs to optimize specific performance. Cooper et al. discussed the tradeoffs faced in cloud based data management systems, including read performance versus write performance, latency versus durability, synchronous versus asynchronous replication, and data partitioning. Some other design metrics have also been argued in papers. In this section, we provide evaluation some of salient features of surveyed NoSQL databases. See table 1.

***CAP Option*:** CAP theorem reveals that a shared data system can only choose at most two of three properties: consistency, availability, and tolerance to partitions. To deal with partial failures, cloud based databases also replicate data over a wide area, this essentially leaves just consistency and availability to choose. Thus, there is a tradeoff between consistency and availability. Various forms of weak consistency models have been implemented to yield reasonable system availability. Strict consistency cannot be achieved together with availability and partition tolerance according to CAP theorem. Two types of weak consistency, eventually consistency and timeline consistency, are commonly adopted. Eventual consistency means all updates can be expected to propagate through the system and the replicas will be consistent under the given long time period. Timeline consistency refers to all replicas of a given record apply all updates to the record in the same order.

Cassandra is a distributed columnar key value database that uses the eventual consistency model. The Cassandra database offers good scalability and high availability without compromising performance. Its demonstrated fault-tolerance on commodity hardware (cloud infrastructures) and linear scalability make it the ideal platform for mission-critical data. It does partitioning. HBase provides linear and modular scalability (ability of a database to handle a growing amount of data), consistent reads and writes, automatic and configurable sharding (a horizontal partition in a database) of tables, and automatic failover support between Region Servers. Applications which need to respond too many parallel read and write requests and which only have to provide a certain level of consistency should use BigTable, Hypertable or Hbase. On CAP theorem Hbase focuses on Consistency and partition tolerance, offering strict consistency model for optimized reads. Cassandra, SimpleDB and DynamoDB offer optimistic replication, wherefore they can be used in any context. Since BigTable, HBase and Hypertable do not use replication for load balancing, these stores offer full consistency.

DynamoDB helps manage the state of services that have high reliability requirements and require tight control over the tradeoffs between availability, consistency, cost-effectiveness, and performance. It uses a combination of successfully implemented techniques to accomplish scalability and availability. Dynamo pioneered the idea of eventual consistency as a way to achieve higher availability and scalability: data fetched are not guaranteed to be up-to-date, but updates are guaranteed to be propagated to all nodes eventually. Like most of the systems we discuss, SimpleDB supports eventual consistency, not transactional consistency. Cassandra datasets are partitioned horizontally by consistent hashing, whereas the BigTable clones HBase and Hypertable use range based partitioning. Since column family data models can be partitioned more efficiently, these databases are more suitable for huge datasets than other data models.

***Persistence*:** Persistency is ensured in column oriented databases. All CODBs are persistent and strictly consistent fault tolerant distributed databases.

Concurrency Control: There are three concurrency control mechanisms in the surveyed systems, locks, MVCC, and none. Locks mechanism allows only one user at a time to read or modify an entity (an object, document, or row). MVCC mechanism guarantees a read-consistent view of the database, but resulting in multiple conflicting versions of an entity if multiple users modify it at the same time. Some systems do not provide atomicity, allowing different users to modify different parts of the same object in parallel, and giving no guarantee as to which version of data you will get when you read.

Cassandra focuses on “weak” concurrency (via MVCC) and HBase and HyperTable on “strong” consistency (via locks and logging). In Hbase row operations are atomic, with row-level locking and transactions. There is optional support for transactions with wider scope. These use optimistic concurrency control, aborting if there is a conflict with other updates. BigTable and HBase are the only column based NoSQL systems, which support a limited variant of classic locks and transactions, since these techniques can only be applied on single rows. Hypertable is very similar to HBase and BigTable. It uses column families that can have any number of column “qualifiers”. It uses timestamps on data with MVCC.

Replication: can insure that mirror copies are always in sync. Alternatively, the mirror copy may be updated asynchronously in the background. Asynchronous replication allows faster operation, particular for remote replicas, but some updates may be lost on a crash.

Replication mode in Bigtable is asynchronous. In HBase, replication is done asynchronously, that is, the clusters can be geographically distant and the links connecting them can be offline for some time, and rows inserted on the master server may not be available at the same time on the slave servers, hence providing only eventual consistency.

Cassandra support asynchronous replication—that is, wide-area replication—without adding significant overhead to the update call itself. In this model, writes are allowed anywhere, and conflicting writes to the same object are resolved afterward. With replication and peer to peer model Cassandra is fault tolerant and provides no single point of failure. However, Cassandra has a weaker concurrency model than some other systems: there is no locking mechanism, and replicas are updated asynchronously. In DynamoDB replica consistency is maintained during updates using a synchronization protocol that applies a quorum like technique and decentralized replica. Like most of the other systems, SimpleDB does asynchronous replication.

In general, it is hard to maintain ACID guarantees in big data applications. The choice of data management tools depends on many factors including the aforementioned metrics. For instance, data model associates with the data sources; data storage devices affect the access rate. Big data storage system should find the right balance between cost, consistency and availability.

Table 1: Comparison of column oriented databases

	BigTable	Hypertable	Hbase	Cassandra	SimpleDB	DynamoDB
Implementation Language	C C++ Java	C++	Java	Java	Erlang	Erlang
Consistency	yes	yes	yes	no	no	no
High Availability	no	no	no	yes	yes	yes
Partition Tolerance	yes	yes	yes	yes	yes	yes
Persistence	yes	yes	yes	yes	yes	yes
Concurrency Control	Locks	MVCC	Locks	MVCC	None	MVCC
Replication	Async	Sync	Async	Async	Async	Sync
Best Use	designed to scale across hundreds or thousands of machines	Same as HBase, since it's basically a replacement	Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement	Web analytics, to count hits by hour, by browser, by IP, etc. Transaction logging. Data collection from huge sensor arrays.	It provides core database functions of information indexing and querying in the cloud. It provides a simple API for storage and access.	Fast and flexible database service for all applications that need consistent, single-digit millisecond latency at any scale.

Conclusion

Basically, column-based NoSQL databases are two dimensional arrays whereby each key (i.e. row / record) has one or more key / value pairs attached to it and these management systems allow very large and un-structured data to be kept and used (e.g. a record with tons of information).
These databases are commonly used when simple key / value pairs are not enough, and storing very large numbers of records with very large numbers of information is a must. DBMS implementing column-based, schema-less models can scale extremely well.
We reviewed the concepts of the column oriented NoSQL databases, motivation behind NoSQL databases and why many of big companies using them. NoSQL databases different in many aspects from traditional databases like structured schema, transaction methodology, complexity, crash recovery and dealing with storing big data which the feature lead to use NoSQL in cloud computing and may be data warehouses.
In this paper, we discussed about popular column oriented NoSQL databases and strengths and weaknesses of various column-based NoSQL database approaches to supporting applications that process huge volumes of data.

A comparison among the most prominent databases was performed on a number of dimensions, including CAP theorem, persistent, concurrency control and replication modes. The discussion of the main characteristics, together with the comparison of data stores, will assist practitioners in choosing the best storage solution for their needs. To the best of our knowledge, there is no publication that explained comparing and evaluating characteristics of column oriented NoSQL databases.

Finally NoSQL has well experience big evolution in the near future because most of current applications and software are tend to depending on web also size of data need to store is in continues increasing rapidly, that convince us to believe that NoSQL databases well face huge growth and improvement and well solve its problems soon or later.

Acknowledgment

I'd like to thank Dr. Mahboubeh Shamsi for her support during this research paper. Any errors are my own, however!

References

[1]. Moniruzzaman, A. B. M., and Syed Akhter Hossain. "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison." INTERNATIONAL JOURNAL OF DATABASE THEORY AND APPLICATION, preprint arXiv: 1307.0191 (2013).

[2]. Mohamed, Mohamed A., Obay G. Altrafi, and Mohammed O. Ismail. "Relational vs. NoSQL Databases: A Survey." International Journal of Computer and Information Technology (ISSN: 2279–0764) Volume 03 – Issue 03, May 2014

[3]. Tauro, Clarence JM, S. Aravindh, and A. B. Shreeharsha. "Comparative study of the new generation, agile, scalable, high performance NOSQL databases." International Journal of Computer Applications, Volume 48– No.20, June 2012.

[4]. Deka, Ganesh Chandra. "A Survey of Cloud Database Systems." IT Professional 2 (2014): 50-57.

[5]. Leavitt, Neal. "Will NoSQL databases live up to their promise?" Computer 43.2 (2010): 12-14.

[6]. Jing Han; Haihong, E.; Guan Le; Jian Du, "Survey on NoSQL database", IEEE 6th International Conference on Pervasive Computing and Applications (ICPCA)" , vol., no., pp.363-366, 26-28 Oct. 2011.

[7]. Hecht, Robin, and Stefan Jablonski. "Nosql evaluation." IEEE International Conference on Cloud and Service Computing. 2011.

[8]. BĂZĂR, Cristina, and Cosmin Sebastian IOSIF. "The Transition from RDBMS to NoSQL. A Comparative Analysis of Three Popular Non-Relational Solutions: Cassandra, MongoDB and Couchbase." Database Systems Journal vol. V, no. 2/2014.

[9]. Cattell, Rick. "Scalable SQL and NoSQL data stores." ACM SIGMOD Record 39.4 (2011): 12-27.

[10]. Tauro, Clarence JM, Baswanth Rao Patil, and K. R. Prashanth. "A Comparative Analysis of Different NoSQL Databases on Data Model, Query Model and Replication Model." Proceedings of the International Conference on "Emerging Research in Computing, Information, Communication and Applications" ERCICA 2013, ISBN. Vol. 1120603436. 2013.

[11]. Zaki, Asadulla Khan. "NoSQL DATABASES: NEW MILLENNIUM DATABASE FOR BIG DATA, BIG USERS, CLOUD COMPUTING AND ITS SECURITY CHALLENGES." International Journal of Research in Engineering and Technology Volume: 03 Special Issue: 03, May-2014, eISSN: 2319-1163.

[12]. Zvarevashe, Kudakwashe, and Tatenda Trust Gotora. "A Random Walk through the Dark Side of NoSQL Databases in Big Data Analytics." International Journal of Science and Research 3 (2014): 506-09.

[13]. Manoj, V. "Comparative Study of NoSQL Document, Column Store Databases and Evaluation of Cassandra." International Journal of Database Management Systems (IJDMS) Vol.6, No.4, August 2014.

[14]. Kumar, Rakesh, Sanketh S. Salian, and Suraj Nayak. "Comparative Analysis of Various Techniques Used In Managing Big Data." International Journal of Innovative Research in Computer and Communication Engineering, Vol.2, Special Issue 5, October 2014.

[15]. Grolinger, Katarina, et al. "Data management in cloud environments: NoSQL and NewSQL data stores." Journal of Cloud Computing: Advances, Systems and Applications 2.1 (2013): 22.