NoSQL Databases: state-of-the-art and Security Challenges

Rabi Prasad Padhy[1], Deepti Panigrahy[2]
[1(]*Technical SME, IBM India Pvt. Ltd., Bangalore, Karnataka, India)*
[2(]*R & D Engineer, Redknee Technologies, Bangalore, Karnataka, India)*

***Abstract*:**
Traditional RDBMS's were facing challenges in meeting the performance and scale requirements of Big Data. NoSQL stores present themselves as alternatives that can handle huge volume of data, support for next-generation web applications and offer a significant change to how enterprise applications are built. These databases are commonly schema-less, easy replication support, simple API, eventually consistent / BASE (not ACID), So these databases are used more and more in companies and startups where there is a huge need to dig the 'big-data' treasures. This research paper describes the NoSQL database background, basic characteristics, data models & architecture. In addition, this paper classifies NoSQL databases according to the CAP theorem. Finally, the mainstream NoSQL databases are described in detail and extract some properties to help enterprises to choose NoSQL. Consequently we also identified research challenges, security measures and future prospects of NOSQL databases.

Keywords: *NoSQL Data Store, Big Data, Hbase, Cassandra, MangoDB, Cloud Database;*

I. INTRODUCTION

Digital world is growing very fast and become more complex in the volume (terabyte to petabyte), variety (structured and unstructured and hybrid), velocity (high speed in growth) in nature. To addressing ever increasing changes in data management needs require solutions that can achieve unlimited scalability, high availability and massive parallelism while ensuring high performance levels [1]. Big Data and NoSQL technologies are simultaneously marketing hypes and tools that could significantly change the database and application development landscape. This refers to as 'Big Data' that is a global phenomenon. The new breed of applications like business intelligence, enterprise analytics, Customer Relationship Management, document processing, Social Networks, Web 2.0 and Cloud Computing require horizontal scaling of thousands of nodes as demanded when handling huge collections of structured and unstructured data sets that traditional RDBMS fail to manage. This led to the development of horizontally scalable, distributed non-relational data stores, called Nosql databases. Currently there are about 150 different NoSQL databases available [2].

II. OVERVIEW OF NOSQL DATABASE SYSTEMS

In general, NoSQL databases have become the first alternative to relational databases, with scalability, availability, and fault tolerance being key deciding factors. They go well beyond the more widely understood legacy, relational databases (such as Oracle, SQL Server and DB2 databases) in satisfying the needs of today's modern business applications [1]. A very flexible and schema-less data model, horizontal scalability, distributed architectures, and the use of languages and interfaces that are "not only" SQL typically characterize this technology. From a business standpoint, considering a NoSQL or 'Big Data' environment has been shown to provide a clear competitive advantage in numerous industries. In the 'age of data', this is compelling information as a great saying about the importance of data is summed up with the following "if your data isn't growing then neither is your business". By design, NoSQL databases and management systems are relation-less (or schema-less). They are not based on a single model (e.g. relational model of RDBMSs) and each database, depending on their target-functionality, adopt a different one.

NoSQL CHARACTERISTICS:

A. Based on Distributed Computing:

Unlike traditional RDBMS, NoSQL database have been designed to favour distributed computing and a shared nothing architecture. This is essentially because scaling horizontally is believed to be the only cost effective way of handling large volumes of data. Additionally horizontally scaled databased is a simpler way to handle large workloads.

B. Commodity Hardware:

Most NoSQL database have been designed to run on cheap commodity hardware (in reality high end commodity hardware) instead to high end servers. This has mainly been done in order to enable scaling in a cost effective manner.

C. ACID, BASE and the CAP Theorem:

NoSQL database have traded one or more of the ACID (atomicity, consistency, isolation and durability) properties for BASE properties (Basic Availability, Soft-state, Eventual consistency). As all new NoSQL databases use distributed computing and due to the limitations placed by the CAP theorem NoSQL database have chosen BASE (Basic Availability, Soft-state, Eventual consistency) over ACID. While ACID is a pessimistic approach and forces consistency at the end of each transaction, BASE is an optimistic approach where by it accepts that data will be in a state of flux but will eventually sort itself out [4]. Choosing BASE over ACID enables systems to scale horizontally.

D. Provide a Flexible Schema:

In order to store the large growing amount of semi structured and unstructured data developers need a flexible solution that easily accommodates different types of data. Additionally due to the constant change in requirements a schema which is easily evolvable is also required. Thus most new NoSQL database generally provide a flexible schema which can be easily evolved as opposed to the rigid schemas required by RDBMS. This has made working with semi structured and unstructured data a lot easier.

III. NOSQL DATABASE ARCHITECTURE

The core of the NoSQL database is the hash function – a mathematical algorithm that takes a variable length input and produces a consistent, fixed-length output. The key of each key/value pair being fed to a NoSQL database is hashed and this hash value is used to direct the pair to a particular NoSQL database server, where the record is stored for later retrieval. When an application wishes to retrieve a key value pair, it provides the database with the key [5]. This key is then hashed again to determine the appropriate server where the data would be stored (if the key exists in the database) and then the database engine retrieves the key/value pair from that server.

Role of Data Architecture in NoSQL

A. Components: There are four components in its building block.

B. Modelling Language:It describes the structure of the database and also defines schema on which it is based. Data is stored in the form of rows and columns using XML formats. And each data (value) corresponding to it is assigned a key that is unique in nature. For faster access of data, the model is built in a suitable environment.

C. Database Structure:Each and every database while building uses its own data structures and stores data using permanent storage device.

D. Database Query language:All the operations are performed on the database that are creation, updation, read and delete.

E. Transactions:During any transaction in the data, there may be any type of faults or a failure; then, the machine will not stop working.

IV. NOSQL DATABASE DESIGN

NoSQL database design uses a set of rules called BASE (basically available, soft-state, eventually consistent) to guide their design [6]. Design strategies for NoSQL databases depend on the type of database and the negatives of different data model techniques. Where relational databases have a user-centered approach and NoSQL databases have an application-centered approach. This is a critical difference both in data structures as well as approaches to designing a database. The key design difference between NoSQL and relational databases is the structure of data in each database. Relational databases require data be organized ahead of time. NoSQL databases can have their structure modified on the fly with little impact because they use key-value pairs; updating a data structure in NoSQL can involve adding additional data to the value of one or more keys while leaving other key-value pairs in the database untouched.

Key-value pairs are the main feature of these databases. Keys are names or unique ID numbers and values range from simple data to documents to columns of data to structured lists (arrays) of key-value data. Each row in a NoSQL table includes the key and its value. The design of NoSQL databases depends on the type of database, called stores. Document Stores pair each key identifier with a document which can be a document, key-value pairs, or key-value arrays. Graph Stores are designed to hold data best represented by graphs, interconnected data with an unknown number of relations between the data, for example, social networks or road maps [7]. Key-Value Stores are the simplest type with every bit of data stored with a name (as key) and its data (value). Wide Column Stores are optimized for queries across large data sets.

V. STATE OF THE ART

There are three basic requirements for databases management systems, confidentiality, integrity and availability. The stored data must be available when it is needed (availability), but only to authorized entities (confidentiality), and only modified by authorized entities (integrity). Traditional relational database management systems (RDBMS), like Oracle, SQL and MySQL, have been well-developed to meet the three requirements [8]. In addition, enterprise RDBMS are further required to have ACID properties, Atomic, Consistency, Isolation, and Durability, that guarantee that database

transactions are processed reliably With such desirable properties, RDBMS have been widely used as the dominant data storage choice [9].

VI. NOSQL DATABASES DATA MODELS

The family of data stores belonging to the NoSQL category can be further sub-classified based on their data models. NoSQL data stores into four major categories: key-value stores, column-family stores, document stores, and graph databases. Figure shows representations of these models [10].

Key-value stores:
Key-value stores have a simple data model based on key-value pairs, which resembles an associative map or a dictionary. The key uniquely identifies the value and is used to store and retrieve the value into and out of the data store. The value is opaque to the data store and can be used to store any arbitrary data, including an integer, a string, an array, or an object, providing a schema-free data model. Along with being schema-free, key-value stores are very efficient in storing distributed data, but are not suitable for scenarios requiring relations or structures [11]. Any functionality requiring relations, structures, or both must be implemented in the client application interacting with the key-value store. Furthermore, because the values are opaque to them, these data stores cannot handle data-level querying and indexing and can perform queries only through keys. Key-value stores can be further classified as in-memory key-value stores which keep the data in memory, like Memcached and Redis, and persistent key-value stores which maintain the data on disk, such as BerkeleyDB, Voldemort,andRiak.

Column-family stores:
Most column-family stores are derived from Google Bigtable, in which the data are stored in a column-oriented way. In Bigtable, the dataset consists of several rows, each of which is addressed by a unique row key, also known as a primary key [12]. Each row is composed of a set of column families, and different rows can have different column families. Similarly to key-value stores, the row key resembles the key, and the set of column families resembles the value represented by the row key. However, each column family further acts as a key for the one or more columns that it holds, where each column consists of a name-value pair. Hadoop HBase directly implements the Google Bigtable concepts, whereas Amazon SimpleDB and DynamoDB have a different data model than Bigtable. SimpleDB and DymanoDB contain only a set of column name-value pairs in each row, without having column families. Cassandra, on the other hand, provides the additional functionality of super-columns, which are formed by grouping various columns together. Typically, the data belonging to a row is stored together on the same server node. However, Cassandra offers to store a single row across multiple server nodes by using composite partition keys. In column-family stores, the configuration of column families is typically performed during start-up. However, a prior definition of columns is not required, which offers huge flexibility in storing any data type. In general, column-family stores provide more powerful indexing and querying than key-value stores because they are based on columnfamilies and columns in addition to row keys [13]. Similarly to key-value stores, any logic requiringrelations must be implemented in the client application.

Document stores:
Document stores provide another derivative of the key-value store data model by using keys to locate documents inside the data store. Most document stores represent documents using JSON (JavaScript Object Notation) or some format derived from it. For example, CouchDB and the Couchbase server use the JSON format for data storage, whereas MongoDB stores data in BSON (Binary JSON). Document stores are suitable for applications in which the input data can be represented in a document format [14]. A document can contain complex data structures such as nested objects and does not require adherence to a fixed schema. MongoDB provides the additional functionality of grouping the documents together into collections. Therefore, inside each collection, a document should have a unique key. Unlike an RDBMS, where every row in a table follows the same schema, each document inside these document stores can have a different structure. Document stores provide the capability of indexing documents based on the primary key as well as on the contents of the documents. This indexing and querying capability based on document contents differentiates this data model from the key-value stores model, in which the values are opaque to the data store. On the other hand, document stores can store only data that can be represented as a document. Like key-value stores, they are inefficient in multiple-key transactions involving cross-document operations.

Graph databases:
Graph databases originated from graph theory and use graphs as their data model. A graph is a mathematical concept used to represent a set of objects, known as vertices or nodes, and the links (or edges) that interconnect these vertices. By using a completely different data model than key-value, column-family, and document stores, graph databases can efficiently store the relationships between different data nodes. In graph databases, the nodes and edges also have individual properties consisting of key-value pairs [15]. Graph databases are specialized in handling highly interconnected data and therefore are very efficient in traversing relationships between different entities. They are suitable in scenarios such as social networking

applications, pattern recognition, dependency analysis, recommendation systems and solving path finding problems raised in navigation systems. Some graph databases such as Neo4J are fully ACID-compliant. However, they are not as efficient as other NoSQL data stores in scenarios other than handling graphs and relationships [16]. Moreover, existing graph databases are not efficient at horizontal scaling because when related nodes are stored on different servers, traversing multiple servers is not performance-efficient.

VII. DIFFICULTIES MIGRATING FROM SQL TO NOSQL

The key requirements for our application:

Some of the requirements that match the need for NoSQL are

A. Rapid application development

- Changing market needs.
- Changing data needs.

B. Scalability

- Unknown user demand.
- Need for constantly growing throughput to access, add and update data.

C. Consistent performance

- Low response time for better user experience.
- High throughput to handle viral growth.

D. Operational reliability

- High-availability to handle failures gracefully with minimal impact to the application.
- Built-in monitoring APIs for easy ongoing maintenance.

VIII. KEY CONSIDERATIONS WHEN CHOOSING OUR NOSQL PLATFORM

A. Workload Diversity

Big Data comes in all shapes, colors and sizes. Rigid schemas have no place here; instead you need a more flexible design. We want our technology to fit our data, not the other way around. And we want to be able to do more with all of that data – perform transactions in real-time, run analytics just as fast and find anything we want in an instant from oceans of data, no matter what from that data may take.

B. Scalability

With big data we want to be able to scale very rapidly and elastically, whenever and wherever we want. This applies to all situations, whether scaling across multiple data centers and even to the cloud if needed.

C. Performance

In an online world where nanosecond delays can cost the business, Big Data must move at extremely high velocities no matter how much we scale or what workloads our database must perform. Performance of our environment, namely our applications, should be high on the list of requirements for deploying a NoSQL platform.

D. Continuous Availability

Building off of the performance consideration, when we rely on big data to feed your essential, revenue-generating 24/7 business applications, even high availability is not high enough. Our data can never go down, therefore there should be no single point of failure in our NoSQL environment, and thus ensuring applications are always available.

E. Manageability

Operational complexity of a NoSQL platform should be kept at a minimum. Make sure that the administration and development required to both maintain and maximize the benefits of moving to a NoSQL environment are achievable.

F. Cost

This is certainly a glaring reason for making the move to a NoSQL platform as meeting even one of the considerations presented here with relational database technology can cost become prohibitively expensive. Deploying NoSQL properly allows for all of the benefits above while also lowering operational costs.

G. Strong Community

This is perhaps one of the more important factors to keep in mind as we move to a NoSQL platform. We need to make sure there is a solid and capable community around the technology, as this will provide an invaluable resource for the individuals and teams that will be managing the environment. Involvement on the part of the vendor should not only include strong support and technical resource availability, but also consistent outreach to the user base. Good local user groups and meetups will provide many opportunities for communicating with other individuals and teams that will provide great insight into how to work best with the platform of choice.

IX. NOSQL PRODUCTS

A. Redis

Redis is a “NoSQL” key-value, networked, in-memory data store written in ANSI C. Its very popular key-value data store, languages that already have bindings for it include ActionScript, C, C++, C#, Clojure, Common Lisp, Dart, Erlang, Go, Haskell, Haxe, Io, Java, JavaScript (Node.js), Lua,

Objective-C, Perl, PHP, Pure Data, Python, R, Ruby, Scala, Smalltalk and Tcl. Key features include a dictionary data model key-mapped to values, persistence through storage of the entire dataset in memory, master-slave replication and better performance via in-memory storage. Redis also offers alpha stage clustering, ease-of-use in IaaS and PaaS platforms, and the ability to use Radis as a managed service without having to launch the VM instance of the database [17].

Fig. 1. NoSQL Products

B. Riak:

Riak uses a simple key/value model for object storage. Objects in Riak consist of a unique key and a value, stored in a flat namespace called a bucket. We can store anything we want in Riak: text, images, JSON/XML/HTML documents, user and session data, backups, log files, and more [20]. It has Amazon S3-API compatibility, per-tenant visibility (accessible over network I/O), metadata and large object support, multi-datacenter replication, and more. Data in Riak is private by default and Access Control Lists are available to further refine data visibility.

C. Apache Cassandra:

This database providing scalability, high availability and fault-tolerance on hardware, virtual systems or cloud infrastructure [23]. With column indexing, log-structured updates, denormalized and materialized views and built-in caching, many large-scale organizations have chosen to use Cassandra (including Constant Contact, CERN, Comcast, eBay, GitHub, GoDaddy, Hulu, Instagram, Intuit, Netflix, Reddit, The Weather Channel, and many others). Features include automatic replication to multiple nodes for fault-tolerance, avoiding single points of failure by keeping cluster nodes identical, synchronous or asynchronous replication during updates, and read/write throughput supported without downtime or interruption. Third party contract support services for Apache Cassandra are also available.

D. HBase:

Apache Hbase is distributed, scalable, secure and provides high availability. Modeled after Google's BigTable, HBase can handle massive data tables containing billions of rows, millions of columns, and utilizes storage, memory and CPU resources across multiple servers within a cluster so that the database scales horizontally [22]. Other features include Kerberos security across tables and columns, automatic sharding, full consistency, and a scale-out architecture allowing for the addition of servers for increased capacity. HBase also features compression, in-memory operation and Bloom filters on a per-column basis. MapReduce jobs run in Hadoop and can use HBAse tables for input and output.

E. Amazon DynamoDB:

DynamoDB is a cloud based NoSQL database offered by Amazon. It is one of the fastest growing AWS services. It 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. All data items are stored on Solid State Drives (SSDs), and are replicated across three Availability Zones for high availability and durability [25].

F. PROS:

- Scalable
- Simple
- Hosted by Amazon
- Good SDK
- Free account for small amount of reads/writes
- Pricing based on throughput

G. CONS:

- Poor documentation
- Limited data types
- Poor query comparison operators
- Unable to do complex queries
- 64KB limit on row size
- 1MB limit on querying

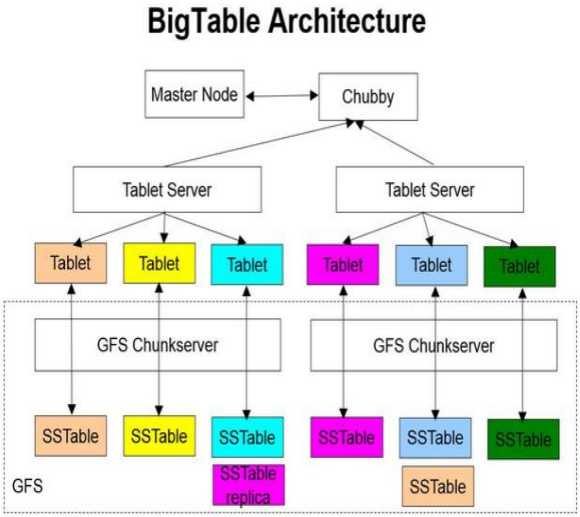
H. Google Bigtable:

Bigtable is a distributed storage system for structured data. Bigtable can handle data that scales to a very large size, even to petabytes, distributed across thousands of servers. Many Google projects such as Google Earth, Google Finance, and Orkut with varied latency requirements and real-time processing use Bigtable to store their data [21]. These applications have asynchronous processes updating the data simultaneously at a very high speed. A read/write of about a million operations per second is what is expected. Bigtable stores data as a distributed multidimensional sorted map with row, column and timestamp. It places frequently accessed columns together as column families. Storing the timestamp allows multiple versions of the contents to be stored

in the same cell and users can access the most recent version or base query on timestamp range. Rows are ordered lexicographically and groups of contiguous rows are stored on same machines as a single tablet for easy access.

Bigtable implementation involves one Master server and many tablet servers. The master assigns data tablets which are contiguous rows of data in a table to tablet servers, balances the load and collects garbage. The tablet servers service the clients. The data is stored as tables and each table is split into many tablets based on a range of rows. Table/ tablets are split automatically when the size of the tablet increases or the load becomes heavy on a tablet. There is no replication of tablets. Each tablet is serviced by only one tablet server. Access to data is in the form of a three level hierarchy. The first level is an address in the chubby to the root directory; the second level is the address of the tablet in the metadata table. The third level is the actual address of the user tablet that contains the data. The traffic on the root directory is regulated by caching the information of the metadata tablet on the client machines and also dedicating one tablet server to service just that metadata tablet. In case of this tablet server going down, the cached data on the clients are used until the metadata tablet is reassigned. If the need arises, it is also possible to replicate this metadata information. The chubby directory keeps track of the tablets assigned to various tablet servers. When the user needs any information, the three level hierarchical access takes the user straight to the tablet without worrying about the actual physical location. This completely abstracts the path access from the user.

Fig. 2. BigTable Architecture



I. CouchDB:

CouchDB is a database that completely embraces the web. Store our data with JSON documents. Access our documents and query our indexes with our web browser, via HTTP. Index, combine, and transform your documents with JavaScript. CouchDB works well with modern web and mobile apps. We can even serve web apps directly out of CouchDB. And we can distribute our data/apps, efficiently using CouchDB's incremental replication. CouchDB supports master-master setups with automatic conflict detection [19].

J. MongoDB:

This is a open source document database written in C++. Features include document-oriented storage (JSON-style documents, dynamic schemas), full index support (on any attribute), replication and high availability (across LANs and WANs for scale), auto-sharding (scale horizontally), querying, rapid in-place updates and map/reduce [18]. MongoDB also has flexible aggregation and data processing, GridFS, (store files of any size), MongoDB management service and professional support. One advantages of MongoDB is embedded documents and arrays, which reduce the need for expensive joins. Additionally, dynamic schema supports fluent polymorphism and documents correspond to native data types in many programming languages.

K. Neo4J:

Neo4J is a Java-based open source NoSQL graph database. With a graph database, which can search social network data, connections between data are explored [26]. Neo4j can solve problems that require repeated network probing (the database is filled with nodes, which are then linked), and the company stresses Neo4j's high performance. The importance of graph database technology as well as Neoo4j's potential in the mobile space. Eifrem also stressed his confidence in Java, despite recent security issues affecting the platform.

L. InfiniteGraph:

InfiniteGraph is a distributed graph database implemented in Java, and is from a class of NOSQL (or Not Only SQL) data technologies focused on graph data structures [27]. Graph data typically consist of objects or things (nodes) and various relationships (edges) that may connect two or more nodes. Developers may use Infinitegraph to build web and mobile applications and services that need to solve graph problems or answer.

X. COMPARISON BETWEEN RDBMS VS NOSQL

A. RDBMS

- Structured and organized data
- Structured query language (SQL)

- Data and its relationships are stored in separate tables.
- Data Manipulation Language, Data Definition Language
- Tight Consistency
- BASE Transaction

B. NoSQL

- Stands for Not Only SQL
- No declarative query language
- No predefined schema
- Key-Value pair storage, Column Store, Document Store, Graph databases
- Eventual consistency rather ACID property
- Unstructured and unpredictable data
- CAP Theorem
- Prioritizes high performance, high availability and scalability

XI. SECURITY CHALLENGES

Security in NoSQL databases is very weak, Authentication and Encryption is almost nonexistence or is very weak when implemented. In comparison with the relational databases, NoSQL databases provide a very thin layer of security. The NoSQL databases emerge with different security issues. The NoSQL databases are built to meet the requirements of analytical world of big data, and less emphasis on security is given during design stage. To overcome the security issues of NoSQL databases, developers must embed the security mechanism at the middleware along with strengthening the database itself in comparison with the relational databases without compromising the scalability and performance features.

The below are security issues associated with NoSQL databases:

- Administrative user or authentication is not enabled by default.
- It has a very weak password storage
- Client communicates with server via plaintext(MongoDB)
- Cannot use external encryption tools like LDAP, Kerberos etc
- Lack of encryption support for the data files
- Weak authentication both between client and the servers
- Vulnerability to SQL injection
- Denial of service attacks.
- Data at rest is Unencrypted.
- The Available encryption solution isn't production ready
- Encryption isn't available for client communication.

XII.CONCLUSION & FUTURE WORK

NoSQL databases taking the world by storm and promising future for the lightweight storage of data in a highly efficient manner. We have reviewed most popular NoSQL databases and outlines their main security features, problems and researchers to choose appropriate storage solutions, and identifying challenges and opportunities in the field. A comparison between RDBMS & NoSQL was performed on a number of dimensions, including data models, querying capabilities, scaling, and security attributes. We also reviewed the main functionality and security features of two of the most popular NoSQL databases.

REFERENCE

[1] Padhy, Rabi Prasad, ManasRanjan Patra, and Suresh Chandra Satapathy. "RDBMS to NoSQL: Reviewing some next-generation non-relational databases." International Journal of Advanced Engineering Science and Technologies 11.1 (2011): 15-30.

[2] Ramanathan, Shalini; Goel, Savita; Alagumalai, Subramanian; , "Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable," Recent Trends in Information Systems (ReTIS), 2011 International Conference on , vol., no., pp.165-168, 21-23 Dec. 2011 doi: 10.1109/ReTIS.201 1.614686.

[3] Floratou, A., Teletia, N., DeWitt, D. J., Patel, J.M., and Zhang, D. (2012) 'Can the elephants handle the NoSQL onslaught?', Proceedings of the VLDB Endowment, 5, 12, 1712-1723

[4] Clarence J M Tauro, Aravindh S, Shreeharsha A. B,"Comparative Study of the New Generation, Agile,Scalable, High Performance NOSQL Databases", International Journal of Computer Applications (0975–888) Volume 48–No.20, June 2012 doi:10.5120/7461-0336

[5] Jing Han; Haihong, E.; Guan Le; Jian Du; , "Survey on NoSQL database," Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on , vol., no., pp.363-366, 26-28 Oct. 2011 doi: 10.1109/ ICPCA.2011.6106531

[6] Rick Cattell, 'Scalable SQL and NoSQL data stores' , ACM SIGMOD Record archive, Volume 39 Issue 4, December 2010, Pages 12-27, ACM New York, NY, USA, doi:10.1145/1978915.1978919

[7] ingjie Shi, XiaofengMeng, Jing Zhao, Xiangmei Hu, Bingbing Liu, Haiping Wang, 'Benchmarking cloud-based data management systems', CloudDB '10 Proceedings of the second international workshop on Cloud data management,pages 47-54, ACM New York, NY,USA 2010, ISBN: 978-1-4503-0380-4, doi: 10.1145/1871929.1871938

[8] Bogdan Tudorica, 'Challenges for the NoSQL systems: Directions for Further Research and Development', The International Journal of Sustainable Economies Management (IJSEM), Volume 2: Issue 1 (2013), DOI:10.4018/IJSEM.2013010106, ISSN: 2160-9659, EISSN: 2160-9667.

[9] Bogdan Tudorica, Bucur Cristian, 'A comparison between several NoSQL databases with comments and notes', The proceedings of „2011 - Networking in Education and Research" IEEE International Conference, June 23, 2011 – June 25, 2011, AlexandruIoanCuza University from Iasi.

[10] Stonebraker, Michael; Madden, Samuel; Abadi, Daniel J.; Harizopoulos, Stavros, "The end of an architectural era: (it's time for a complete rewrite)," Proceedings of the 33rd international conference on Very large data bases, VLDB, p. 1150–1160, 2007.

[11] Sharma and M. Dave, "SQL and NoSQL Databases," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 8, pp. 20-27, 2012.

[12] [ishthaJatana, SahilPuri, Mehak Ahuja, IshitaKathuria, DishantGosain, "A Survey and Comparison of Relational and Non-Relational Database," International Journal of

Engineering Research & Technology (IJERT), vol. I, no. 6, 2012.

[13] Jayathilake, D.; Sooriaarachchi, C.; Gunawardena, T.; Kulasuriya, B.; Dayaratne, T., "A study into the capabilities of NoSQL databases in handling a highly heterogeneous tree," Information and Automation for Sustainability (ICIAfS), 2012 IEEE 6th International Conference on, vol., no., pp.106,111, 27-29 Sept. 2012. doi: 10.1109/ICIAFS.2012.6419890.

[14] Jing Han; Haihong, E.; Guan Le; Jian Du, "Survey on NoSQL database," Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on, vol., no., pp. 363, 366, 26-28 Oct. 2011. doi:10.1109/ICPCA.2011.6106531.

[15] Tudorica, B. G.; Bucur, C., "A comparison between several NoSQL databases with comments and notes,"Roedunet International Conference (RoEduNet), 2011 10th, vol., no., pp.1,5, 23-25 June 2011. doi:10.1109/RoEduNet.2011.5993686.

[16] Bhatewara, Ankita; Waghmare, Kalyani, "Improving network scalability using nosql database," International Journal of Advanced Computer Research, 2012, Vol. 2 Issue 6, pp. 488

[17] Redis : http://redis.io/documentation

[18] MongoDB: https://www.mongodb.org/

[19] CouchDB: http://couchdb.apache.org/

[20] Riak: http://docs.basho.com/riak/latest/

[21] Bigtable: https://cloud.google.com/bigtable/

[22] HBase: https://hbase.apache.org/

[23] Cassandra: http://cassandra.apache.org/

[24] GraphDB:http://ontotext.com/products/ontotext-graphdb/

[25] https://aws.amazon.com/dynamodb

[26] http://neo4j.com/docs/

[27] http://support.objectivity.com/docs/infinitegraph/