ISSN:2249-5789
Pankaj Sareen et al, International Journal of Computer Science & Communication Networks,Vol 5(5),293-298

NOSQL DATABASE AND ITS COMPARISON WITH SQL DATABASE

Pankaj Sareen[1]

Assistant Professor,
Computer Science Department, SPN College Mukerian
pankaj.sareen.mca@gmail.com

Parveen Kumar[2]

Assistant Professor,
Computer Science Department, SPN College Mukerian
Parveenspn20@gmail.com

Abstract- NOSQL databases is an emerging alternative to the most widely used relational databases. As the name suggests, it does not completely replace SQL but compliments it in such a way that they can co-exist. In this paper we will be discussing the NOSQL database, types of NOSQL database type, advantages and disadvantages of NOSQL.

Keywords-NOSQL, Relational Databases, Data Stores

I. INTRODUCTION

A **NoSQL** (originally referring to "non SQL" or "non relational" [1] database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early twenty-first century, triggered by the needs of Web 2.0 companies such as Face book, Google and Amazon.com.

Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines, which is a problem for relational databases, and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, graph, or document) differ slightly from those used by default in relational databases, making some operations faster in NoSQL and others faster in relational databases. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.

NoSQL databases are increasingly [4]used in big data and real-time web applications. NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages.

Many NoSQL stores compromise consistency (in the sense of the CAP theorem) in favor of availability, partition tolerance, and speed. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages (instead of SQL, for instance the lack of ability to perform ad-hoc JOIN's across tables), lack of standardized interfaces, and huge previous investments in existing relational databases.

Most NoSQL stores lack true ACID transactions, although a few recent systems, such as FairComc-treeACE, Google Spanner (though technically a NewSQL database), FoundationDB, SymasLMDB and OrientDB have made them central to their designs. Instead they offer a concept of "eventual consistency" in which database changes are propagated to all nodes "eventually", so queries for data might not return updated data immediately.

Not all NoSQL systems live up to the promised "eventual consistency" and partition tolerance, but in experiments with network partitioning often exhibited lost writes and other forms of data loss.Fortunately, some NoSQL systems provide concepts such as "Write Ahead Logging" to avoid data loss. Current relational databases also "do not allow referential integrity constraints to span databases" as well.

II. NOSQL DATABASE TYPES

A.Document databases pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

B. Graph stores are used to store information about networks, such as social connections. Graph stores include Neo4J and HyperGraphDB.

C.Key-value stores are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or "key"), together with its value. Examples of key-value stores are Riak and Voldemort. Some key-value stores, such as Redis, allow each value to have a type, such as "integer", which adds functionality.

D.Wide-column stores such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

III. ARCHITECTURE

In MongoDB[2], data or records are called documents and the documents are stored in a binary JSON (BSON) format. Records—documents—are further organized in collections. However, in the same collection, the schema of one document can be different from the other. Hence, flexible schemas are supported. Because developers are familiar with JSON formats, using BSON is simpler.

Nested data structures and array formats are also supported with MongoDB. To implement the same in a relational DB would mean multiple tables with foreign key relationship. It also supports typed data, such as integer, string, date time, double, and so forth.

It supports indexing and querying. A query can vary from key-value, range, or geospatial to aggregation framework queries. Indexes enhance performance when querying the data. Indexes can be declared on unique/single and multiple fields. An index also can be on fields from a nested structure. As with SQL, it also supports verifying the execution plan to optimize query performance.

MongoDB provides a replica set: a failover mechanism. There is only one Primary database that allows a write operation and multiple secondary servers only for read operations. A minimum of three servers is required for a replica set: Primary, Secondary and, Arbiter. Arbiter doesn't store any data; it's only used during failover to decide which server will be the next primary server.

IV. ADVANTAGES OF NOSQL

The relational database (RDBMS) has been[8] the dominant model for database management. But, today, non-relational, "cloud," or "NoSQL" databases are gaining mindshare as an alternative model for database management. The various advantages are:

A. Elastic scaling

For years, database administrators have relied on *scale up* — buying bigger servers as database load increases — rather than *scale out* — distributing the database across multiple hosts as load increases. However, as transaction rates and availability requirements increase, and as databases move into the cloud or onto virtualized environments, the economic advantages of scaling out on commodity hardware become irresistible.

RDBMS might not scale out easily on commodity clusters, but the new breed of NoSQL databases are designed to expand transparently to take advantage of new nodes, and they're usually designed with low-cost commodity hardware in mind.

B. Big data

Just as transaction rates have grown out of recognition over the last decade, the volumes of data that are being stored also have increased massively. O'Reilly has cleverly called this the "industrial revolution of data." RDBMS capacity has been growing to match these increases, but as with transaction rates, the constraints of data volumes that can be practically managed by a single RDBMS are becoming intolerable for some enterprises. Today, the volumes of "big data" that can be handled by NoSQL systems, such as Hadoop, outstrip what can be handled by the biggest RDBMS.

C. Goodbye DBAs

Despite the many manageability improvements claimed by RDBMS vendors over the years, high-end RDBMS systems can be maintained only with the assistance of expensive, highly trained DBAs. DBAs are intimately involved in the design, installation, and ongoing tuning of high-end RDBMS systems.

NoSQL databases are generally designed from the ground up to require less management: automatic repair, data distribution, and simpler data models lead to lower administration and tuning requirements — in theory. In practice, it's likely that rumors of the DBA's death have been slightly exaggerated. Someone will always be accountable for the performance and availability of any mission-critical data store.

D. Economics

NoSQL databases typically use clusters of cheap commodity servers to manage the exploding data and transaction volumes, while RDBMS tends to rely on expensive proprietary servers and storage systems. The result is that the cost per gigabyte or transaction/second for NoSQL can be many times less than the cost for RDBMS, allowing you to store and process more data at a much lower price point.

E. Flexible data models

Change management is a big headache for large production RDBMS. Even minor changes to the data model of an RDBMS have to be carefully managed and may necessitate downtime or reduced service levels.

NoSQL databases have far more relaxed or even nonexistent data model restrictions. NoSQL Key Value stores and document databases allow the application to store virtually any structure it wants in a data element. Even the more rigidly defined BigTable-based NoSQL databases (Cassandra, HBase) typically allow new columns to be created without too much fuss.

The result is that application changes and database schema changes do not have to be managed as one complicated change unit. In theory, this will allow applications to iterate faster, though, clearly, there can be undesirable side effects if the application fails to manage data integrity.

V. EXAMPLES OF NOSQL DATABASES

There have been various approaches to classify NoSQL databases, each with different categories and subcategories, some of which overlap. A basic classification based on data model, with examples:

A. *Column:*Accumulo, Cassandra, Druid, HBase, Vertica
B. *Document:*Clusterpoint, Apache CouchDB, Couchbase, DocumentDB, HyperDex, Lotus Notes, MarkLogic, MongoDB, OrientDB, Qizx
C. *Key-value:* CouchDB, Oracle NoSQL Database, Dynamo, FoundationDB, HyperDex, MemcacheDB, Redis, Riak, FairComc-treeACE, Aerospike, OrientDB, MUMPS
D. *Graph:*Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog
E. *Multi-model:*OrientDB, FoundationDB, ArangoDB, Alchemy Database, CortexDB

VI. DIFFERENCE BETWEEN NOSQL VS. SQL [3]

Difference between NOSQL and SQL is shown in the table below:

	SQL Databases	**NOSQL Databases**
Types	One type (SQL database) with minor variations	Many different types including key-value stores, document databases, wide-column stores, and graph databases
Development History	Developed in 1970s to deal with first wave of data storage applications	Developed in 2000s to deal with limitations of SQL databases, particularly concerning scale, replication and unstructured data storage
Examples	MySQL, Postgres, Oracle Database	MongoDB, Cassandra, HBase, Neo4j
Data Storage Model	Individual records (e.g., "employees") are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., "manager," "date hired," etc.), much like a spreadsheet. Separate data types are stored in separate tables, and then joined together when more complex queries are executed. For example, "offices" might be stored in one table, and "employees" in another. When a user wants to find the work address of an employee, the database engine joins the "employee" and "office" tables together to get all the information necessary.	Varies based on database type. For example, key-value stores function similarly to SQL databases, but has only two columns ("key" and "value"), with more complex information sometimes stored within the "value" columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single "document" in JSON, XML, or another format, which can nest values hierarchically.
Schemas	Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline.	Typically dynamic. Records can add new information on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically.
Scaling	Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required.	Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary.

Development Model	Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database)	Open-source
Supports Transactions	Yes, updates can be configured to complete entirely or not at all	In certain circumstances and at certain levels (e.g., document level vs. database level)
Data Manipulation	Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE…	Through object-oriented APIs
Consistency	Can be configured for strong consistency	Depends on product. Some provide strong consistency (e.g., MongoDB) whereas others offer eventual consistency (e.g., Cassandra)

Table 1. Difference between SQL Databases and NoSQL Databases

VII. STUDY OF DIFFERENCES BETWEEN SQL SERVER AND MONGODB

The main reason[5] why you can ask yourself why you need to move to NoSQL databases are necessity in huge data storage (in other words Big Data), scalability and performance reasons.

Relational databases were developed in 1970s. It was very courteous way to store data and satisfied those day needs. But today, relational databases are not able to solve the needs of the current scope for storing gigantic data. NoSQL technique for storing data is a substitute for solving the nowadays needs. One of the most well-known and leading NoSQL database is MongoDB.

	MS SQL	MongoDB
Data Storage Model	Relational DBMS	Document-oriented
JOINs	Yes	No
Transaction	ACID	No
Support agile practices	No	Yes
Data schema	Fixed	Dynamic
Scalability	Vertical	Horizontal
Replication	Yes (depending on software edition)	Primary-Secondary
MapReduce	No	Yes
Query Language	SQL query language	JSON query language
Secondary Indexes	Yes	Yes
Triggers	Yes	No
Foreign keys	Yes	No
Concurrency	Yes	Yes
Official Website	http://www.microsoft.com/en-us/sqlserver/default.aspx	http://www.mongodb.org
Company	Microsoft	MongoDB, Inc
Licence	Commercial	Open Source
Implementation language	C++	C++
OS support	Windows	Windows, Linux, OS X, Solaris

Table 2. Difference between MS SQL and MongoDB

VIII. WHICH NOSQL DATABASE SHOULD YOU USE?

To this point[6] we have been able to detail many reasons for a business to adopt a NoSQL, or non-relational database platform, and also covered different types of NoSQL database options. Determining which NoSQL database to adopt is an exercise that requires a business to a long hard look at itself and its applications – which is always good. Understand clearly what your business goals are, what your application(s) needs – and challenges – are today, and what those needs may be on the horizon. Also, make sure that both the business and the technology goals are aligned so that the platform decision made delivers for all key stakeholders. Then work back to the NoSQL platform that will satisfy these needs and provide the most solid foundation for you to build on.

Key considerations when choosing your NoSQL platform include:

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. You want your technology to fit your data, not the other way around. And you want to be able to do more with all of that data- perform transactions in real-time, runanalytics just as fast and find anything you want in an instant from oceans of data, no matter what from that data may take.

B. Scalability

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

C. Performance

As has already been discussed, in an online world where nanosecond delays can cost you sales, Big Data must move at extremely high velocities no matter how much you scale or what workloads your database must perform. Performance of your environment, namely your applications, should be high on the list of requirements for deploying a NoSQL platform.

D. Continuous Availability

Building off of the performance consideration, when you rely on big data to feed your essential, revenue-generating 24/7 business applications, even high availability is not high enough. Your data can never go down, therefore there should be no single point of failure in your NoSQL environment, 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 you move to a NoSQL platform. 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. CHALLENGES OF NOSQL

The promise of the NoSQL [7]database has generated a lot of enthusiasm, but there are many obstacles to overcome before they can appeal to mainstream enterprises. Here are a few of the top challenges.

A.Maturity

RDBMS systems have been around for a long time. NoSQL advocates will argue that their advancing age is a sign of their obsolescence, but for most CIOs, the maturity of the RDBMS is reassuring. For the most part, RDBMS systems are stable and richly functional. In comparison, most NoSQL alternatives are in pre-production versions with many key features yet to be implemented.

Living on the technological leading edge is an exciting prospect for many developers, but enterprises should approach it with extreme caution.

B. Support

Enterprises want the reassurance that if a key system fails, they will be able to get timely and competent support. All RDBMS vendors go to great lengths to provide a high level of enterprise support.

In contrast, most NoSQL systems are open source projects, and although there are usually one or more firms offering support for each NoSQL database, these companies often are small start-ups without the global reach, support resources, or credibility of an Oracle, Microsoft, or IBM.

C.Analytics and business intelligence

NoSQL databases have evolved to meet the scaling demands of modern Web 2.0 applications. Consequently, most of their feature set is oriented toward the demands of these applications. However, data in an application has value to the business that goes beyond the insert-read-update-delete cycle of a typical Web application. Businesses mine information in corporate databases to improve their efficiency and competitiveness, and business intelligence (BI) is a key IT issue for all medium to large companies.

NoSQL databases offer few facilities for ad-hoc query and analysis. Even a simple query requires significant programming

expertise, and commonly used BI tools do not provide connectivity to NoSQL.

Some relief is provided by the emergence of solutions such as HIVE or PIG, which can provide easier access to data held in Hadoop clusters and perhaps eventually, other NoSQL databases. Quest Software has developed a product — Toad for Cloud Databases — that can provide ad-hoc query capabilities to a variety of NoSQL databases.

D.Administration

The design goals for NoSQL may be to provide a zero-admin solution, but the current reality falls well short of that goal. NoSQL today requires a lot of skill to install and a lot of effort to maintain.

E. Expertise

There are literally millions of developers throughout the world, and in every business segment, who are familiar with RDBMS concepts and programming. In contrast, almost every NoSQL developer is in a learning mode. This situation will address naturally over time, but for now, it's far easier to find experienced RDBMS programmers or administrators than a NoSQL expert.

X.CONCLUSION

NoSQL databases are becoming an increasingly important part of the database landscape, and when used appropriately, can offer real benefits. However, enterprises should proceed with caution with full awareness of the legitimate limitations and issues that are associated with these databases

REFERENCES

[1] Leavitt, N.,"Will NoSQL Databases Live Up to Their Promise?" Computer, vol.43, no.2, pp.12-14, Feb. 2010doi: 10.1109/MC.2010.58
[2] MongoDB, http://www.mongodb.org/about/introduction/
[3] 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
[4] http://en.wikipedia.org/wiki/NoSQL
[5] db4o, http://www.db4o.com/about/
[6] Apache Cassandra, http://wiki.apache.org/cassandra/
[7] http://www.gmv.com/blog_gmv/nosql-database-challenge
[8] http://www.dummies.com/how-to/content/10-advantages-of-nosql-over-rdbms.html
[9] 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
[10] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber,” Bigtable: A Distributed Storage System for Structured Data”, Google Inc
[11] UnQL , http://www.unqlspec.org/display/UnQL/Home
[12] Apache CouchDB , http://wiki.apache.org/couchdb/
[13] Amazon DynamoDB, http://aws.amazon.com/dynamodb/
[14] Neo4j , http://www.neo4j.org/learn