Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 3159-0040
Vol. 2 Issue 4, April - 2015

Relational Database vs NoSQL

Tejal Patel
Department of Computer Science
University of Bridgeport
Bridgeport, USA
E-mail Id: tejalpatel521@yahoo.in

Tarik Eltaieb
Department of Computer Science
University of Bridgeport
Bridgeport, USA
E-mail Id: teltaeib@bridgeport.edu

***Abstract*—"Big Data" problem has challenged most traditional relational database management system (RDBMS), major Web 2.0 companies have developed or adopted different NoSQL databases for their growing data and infrastructural needs Google (BigTable), LinkedIn (Voldemort), Facebook (Cassandra), Amazon(Dynamo) etc. Application developers have been frustrated with the impedance mismatch between the relational data structures and the in-memory data structures of the application. Using NoSQL databases allows developers to develop without having to convert in-memory structures to relational structures. From their inception, NoSQL databases have been designed for solving the Big Data issue by utilizing distributed, collaborating hosts to achieve satisfactory performance in data storage and retrieval**

Other important database requirements, such as data security and consistency, have not been fully addressed. However, the potential data inconsistency among replications may impede a wide acceptance of NoSQL by much less tolerable financial applications. NoSQL database is still evolving and is not very secure. NoSQL databases are becoming the targets of attackers who seek valuable information. They become even more susceptible to exploits as the attackers overcome the learning curve and are able to identify the hidden security and software weakness or loop holes.

Keywords—Relational database management system, NoSQL, Big Data, ACID properties, BASE transaction, Object relational mapper (ORM), aggregates, e_ventual consistency, location independence

I. INTRODUCTION

There are three essential prerequisites 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 these requirements. In addition to this enterprise RDBMS 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.

RDBMS are facing major performance problems in processing exponential growth of unstructured data, such as documents, e-mails, multi-media or social media. Thus a new breed of non-relational cloud-based, distributed databases, called NoSQL, has emerged to satisfy the unprecedented needs for scalability, performance and storage. NoSQL databases are sometimes referred to as cloud databases, non-relational databases, Big Data databases and a myriad of other terms and were developed in response to the sheer volume of data being generated, stored and analyzed by modern users (user-generated data) and their applications(machine generated data). They are designed to achieve the desired scalability and performance by sharing a BASE transaction concept (Basically Available, Soft state and eventually consistent). Under this concept, committed transactions are not written to database immediately to achieve data consistency as in RDBMS. Instead, the database just needs to reach a consistent state eventually among the clustering hosts.

II. What is NoSQL?

The word "NoSQL" originated from a hashtag (#NoSQL) about a meeting where people can talk about ideas and the new types of emerging databases. NoSQL means "Not Only SQL". NoSQL was never meant to knockout SQL or supplant it.

Defining NoSQL

NoSQL means Not Only SQL, implying that when designing a software solution or product, there is more than one storage mechanism that could be used based on the needs.

NoSQL systems store and retrieve data in different formats. There are four categories of NoSQL databases. They are:

1. Key value store: A Key-value stores are the simplest NoSQL data stores to use from an API perspective. The client can get the value for the key, put a value for a key, or delete a key from the data store. The value is a blob that the data store just stores, without caring or knowing what's inside, it's the responsibility of the application to understand what is stored. Since key-value stores use primary-key access, they generally have great performance and can be easily scaled.

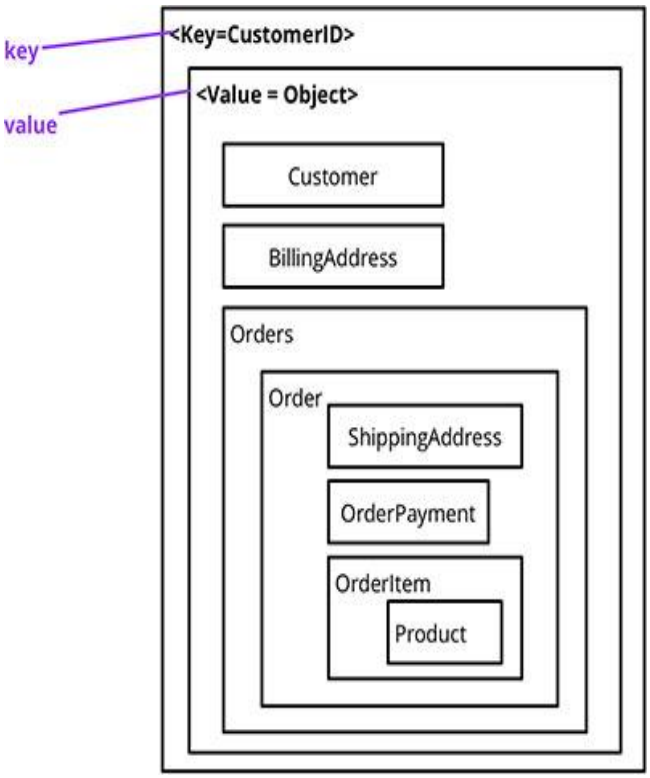

Figure1. Key-value Store

2. Column-family: A sparse matrix system that uses row and a column as keys.

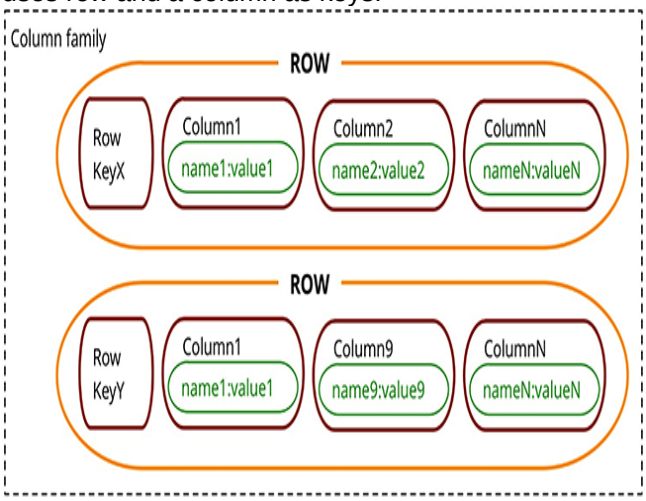

Figure2. Data stored in Column-family

3. Graph databases: Data is stored in graph structures with nodes (entities), properties (information about the entities) and lines (connections between the entities).

Graph based database is any database that provides index free adjacency. Every element contains a direct pointer to its adjacent elements and no index lookups are necessary. Graph databases employ nodes, properties, and edges. Nodes represent entities, properties are pertinent information that relate to nodes and edges represent the relationship between the two.

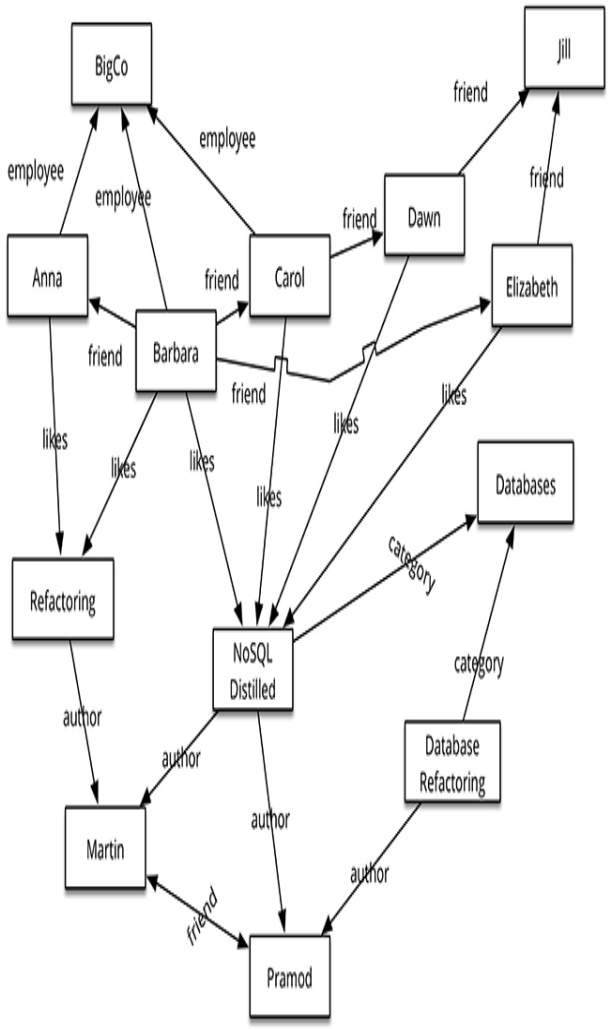

Figure3. Graph database

4. Document stores: Storing hierarchical data structures directly in the database. Documents may be addressed in the database via a unique *key* that represents that document. This key is often a simple string, a URI, or a path.

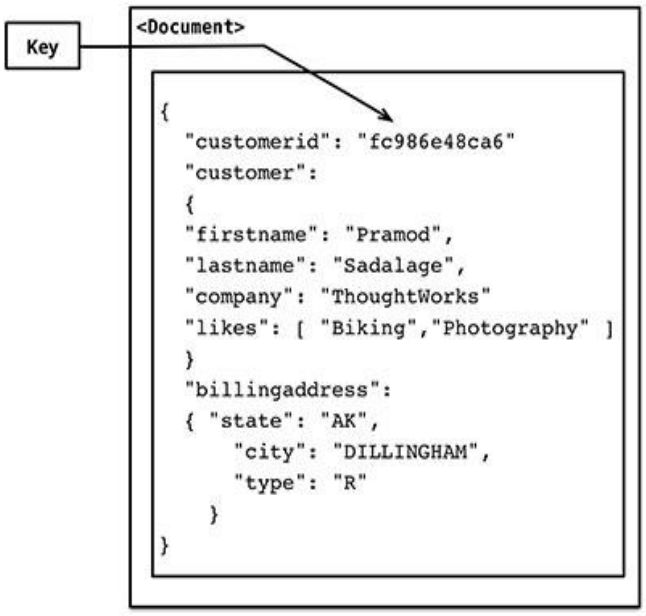

Figure4. Document database

III. HOW IS DATA STORED?

NoSQL stores data as aggregates. An aggregate is a collection of data that we interact with as a unit. Aggregates make it easier for the database to manage data storage over clusters, since the unit of data now could reside on any machine and when retrieved from the database gets all the related data along with it. Aggregate-oriented databases work best when most data interaction is done with the same aggregate, for example when there is need to get an order and all its details, it better to store order as an aggregate object but dealing with these aggregates to get item details on all the orders is not elegant.

Aggregate-oriented databases make inter-aggregate relationships more difficult to handle than intra-aggregate relationships. RDBMS database is better when interactions use data organized in many different formations.

Distribution Models:

Aggregate oriented databases make distribution of data easier, since the distribution mechanism has to move the aggregate and not have to worry about related data, as all the related data is contained in the aggregate. There are two styles of distributing data:

1. Sharding: Sharding distributes different data across multiple servers, so each server acts as the single source for a subset of data.

2. Replication: Replication copies data across multiple servers, so each bit of data can be found in multiple places.

Replication comes in two forms:

a) Master-slave replication makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads.

b) Peer-to-peer replication allows writes to any node; the nodes coordinate to synchronize their copies of the data.

Master-slave replication reduces the chance of update conflicts but peer-to-peer replication avoids loading all writes onto a single server creating a single point of failure. A system may use either or both techniques.

Relational data is tabular in nature. Data is stored in tables in rows and columns. Tables can be related to each other. Each table has a unique primary key which can be used to establish relations between tables and is useful in retrieving data from other tables.

Each table is a represents an entity or object. Columns are the attributes of an entity. The rows contain the values or data instances; these are also called records or tuples.

Relationships exist between the columns within a table and also among tables. These relationships take three logical forms: one-to-one, one-to-many, or many-to-many.

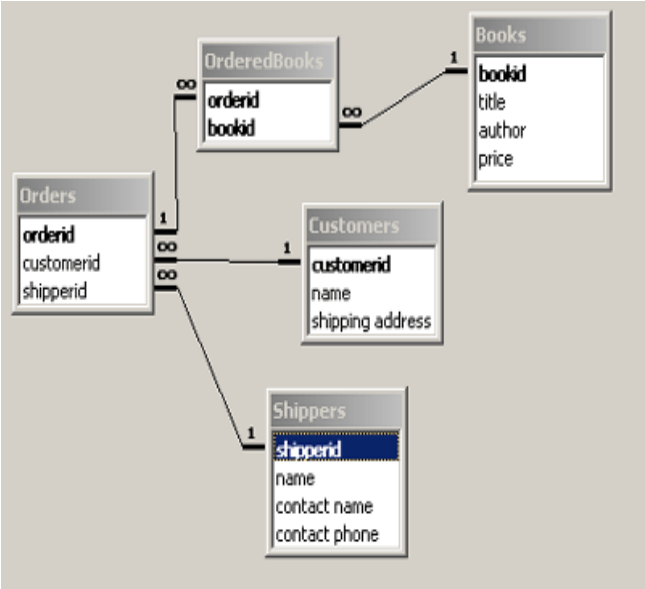

Figure5. Relational database

IV. DATABASE SCHEMA

Relational data is often referred to as structured data, because the tables have a pre-defined schema (column definitions) .While a pre-defined schema offers reliability and stability, a change to a schema on a table with pre-existing data is very difficult. Non-relational data, on the other hand, thrives on dynamic schemas and is often referred to as unstructured data. Non-relational data can easily accommodate changes in data type/structure due to its dynamic schema support.

V. DATA STORAGE

In relational database data is highly normalized, breaking the data into smallest possible logical table to prevent duplication of records and improve space utilization. Though normalization leads to cleaner data management, it often adds up little complexity on data management as retrieval of single information may require spanning across multiple tables.

And today's world cost of data storage is trivial issue. Non-relational data, on the other hand, is stored as flat collections, where data might often be duplicated. A single chunk of data is seldom partitioned off, rather stored as an entity, thus allowing easier reads/writes to that single entity.

VI. SCALABILITY

In today's world how a database scales to meeting the ever growing demands is one of the key factors in selection of the database. To support more concurrent usage, SQL databases scale vertically – this means adding more horsepower to the computer for faster operations on the same set of data. Since the data is stored in relational tables, the performance bottleneck of operations that span a lot of tables is overcome by adding more computational power to the machine.

While SQL databases can scale insanely high, eventually there will be an upper limit to vertically scalability. NoSQL databases, on the other hand, scale horizontally. Non-relational data storage being inherently distributed in nature, scalability for NoSQL databases is achieved horizontally by adding more commodity database servers in the resource pool to share the load**.**

VII. HOW DATA IS RETRIVED?

Relational databases manipulate data using a Structured Query Language (SQL). SQL is incredibly powerful for supporting database **CRUD** (CREATE, READ, UPDATE, DELETE) operations and is an industry-standard. Non-relational databases manipulate data in chunks (like documents) and use Unstructured Query Language (UNQL), which is not standard and may vary between database providers. SQL databases employ pre-defined optimizations like column index definitions to help speed up query operations, while NoSQL databases enjoy simpler but narrow data access patterns.

VIII. HOW IS DATA MAPPED?

Developers using object oriented programming languages are often dealing with one or more data entities (complex structures with nested data, lists or arrays) to fuel the application user interface. In case of relational storage, the data entities often need to be broken down for normalization and lean storage across multiple related tables. Most programming platforms have come up with an easy solution: an **ORM** (Object Relational Mapper) layer. This acts as the mapping layer between the tabular relational data source and the object oriented data entities developers deal with. In non-relational data storage however, there is no need to normalize the data; complex data entities can be stowed away as is in a single unit. Objects used in applications are often serialized into JSON and stored as JSON documents in NoSQL databases.

IX. PERFORMANCE AND STABILITY

If your data operations demand high transactional rates or complex data queries needing control over execution plans, then old SQL database can be your best friend in terms of performance and stability. SQL databases allow fine-grained control over transactional atomicity, and easy rollbacks. Most NoSQL databases do not provide transaction support by default, which means the developers have to think how to implement transactions, does every write have to have the safety of transactions or can the write be segregated into “critical that they succeed” and “it’s okay if I lose this write” categories. We can deploy external transaction managers like zookeeper.

X. DATA INTERGRITY

SQL database are renowned for providing data integrity by implementing ACID properties. ACID stands for Atomic, Consistent, Isolated, and Durability. Most of the database vendors honour ACID properties. ACID is pessimistic as it requires consistency at the end of each operation whereas, BASE is optimistic as it accepts the database consistency to stay in state of flux. BASE stands for Basically Available, Soft State and Eventually Consistent. The e_ventual consistency is an acknowledgement of an unbounded delay in propagating a change made on one machine to all the other copies which might lead to stale data.

For instance, a distributed database system maintains copies of shared data on multiple machines in a cluster to ensure high availability. When data gets updated in a cluster there might be some interval of time during which some of the copies will be updated, while others won’t be. Eventually the changes will be propagated to all machines. That’s the reason why it’s called Eventually Consistent. BASE trades consistency for availability and doesn’t give any ordering guarantees at all. Eventual consistency has nothing to do with a single node system since there’s no need for propagation. If the database system supports eventual consistency, then the application will need to handle the possibility of reading stale or inconsistent data.

CAP Theorem

According to CAP theorem in a distributed system you can choose only two of Availability, Consistency or Partition tolerance. NoSQL databases provide a lot of options where the developers can choose to tune their databases as per their needs and requirements. Availability of choice has its own consequences. It is good thing as now we can design our system as per our specific requirements. But, it can have some drawbacks, as now we need to make wise choices or else the database product might be not be properly utilized.

XI. LOCATION INDEPENDENCE

The term “Local Independence” means the ability to read and write data in database irrespective to the location where the operation was performed physically. And also to have the ability to propagate that write operation to all the other locations where the distributed data is stored so that the user’s or machines in other locations can have access to that data. To implement such functionality in relational database architecture is very difficult. Some techniques such as Master/Slave architecture and data Sharding can be employed to meet the need for location independence for read operations but to achieve it for writing data uniformly is a tedious task, especially for high volume data. There are many scenarios where location independence is of great importance. Location independence can be of utmost importance when it comes to providing service to customers from varied geographies, as the data on these sites needs to be kept locally for faster access. As the business world is evolving, the competition is increasing day by day, delay for a friction of second may can also cause you loss in business. It’s rightfully said that “If your data doesn’t grow nor will your

business grow". So along with maintaining accurate distributed data uniformly, the availability of data is also of utmost importance for companies.

XII. TO THE CLOUD

The ubiquitousness of cloud computing, in reality, has benefited both SQL and NoSQL databases. Relational storage in the cloud often comes as a service that is replicated, highly available and distributed for greater fault tolerance through horizontal sharding techniques. A NoSQL database hosted as a cloud service benefits from inherent auto-sharding, flexible elasticity for seasonal demand, integrated caching and tremendous computing power to capture, store and analyze big data.

XIII. HOW TO CHOOSE THE DATABASE?

In order to determine which database to choose for a particular application we need to understand the needs and requirements of that application, consider all the challenges that the application may face when it's on horizon. Different business applications have different needs, for some availability and scalability is of utmost importance while for other's accuracy and consistency is important. Depending on the needs and priority of the business application we can choose relational database if accuracy, integrity and consistency of data is required but, if availability, performance and scalability is essential as in case of big data problem then NoSQL can emerge as a solution.

Both have their own pros and cons. SQL is the most popular choice as it is the most reliable option available in market and has been there for a long time. We are use to it so when we opt for NoSQL database where we have to decide which architecture to choose and how to manage, store and retrieve data. Making the right decision is very important because if we take wrong decisions then our application can turn around to be useless in other words it won't be able to perform the way it is supposed to be and incur heavy losses for the company. NoSQL database can be easily attacked and precious data can be at risk. NoSQL databases are not very secure as it is still evolving whereas, SQL is quite secure. Since it has been around for a long time, years of research and study has made it a solid solution that guarantees secure and reliable transactions. So for critical data such as bank transactions very security and consistency of data is of utmost importance SQL database is a suitable option.

We can have experienced technical team for SQL databases as it is a well-developed technology but, for NoSQL database to find efficient technical resource is difficult. People are used to querying relational database where they don't have to worry about the algorithm running in background that is applied by the system to retrieve the data. In NoSQL database we need to make wise decisions as which database architecture to use according our needs, how to store, manage and retrieve data.

When we are choosing NoSQL database for our application we need to make sure that there is a solid and capable community around the technology. It will serve as an invaluable resource for individuals as well as for the team that is going to manage the application. The vendor not only requires the involvement of technical resource but, also needs to reach out the user base as good local user groups may provide opportunities for communicating with other individuals or technical teams that might provide great insight as to how to make best use of a particular database.

XIV. CONCLUSION

Relational database has ruled the data world for about 40 years. But for the past couple of years due to Big data problem NoSQL databases have been gaining much attention. They have emerged as a solution for the ever growing data needs of scalability and performance for high volume data applications. NoSQL databases provide flexible architecture for different kinds of data storage needs. They trade consistency for availability. They are highly scalable and flexible. Many companies are co-deploying NoSQL and RDBMS to process different data flows in the ways they are best designed to do. All the choice provided by the rise of NoSQL databases does not mean the demise of RDBMS databases. We are entering an era of polyglot persistence, a technique that uses different data storage technologies to handle varying data storage needs. Polyglot persistence can applied across an enterprise or within a single application.

REFERENCES

[1] http://hebrayeem.blogspot.com/2014/01/making-sense-of-NoSQL.html
[2] http://www.thoughtworks.com/insights/blog/NoSQL-databases-overview
[3] http://www.thegeekstuff.com/2014/01/SQL-vs-NoSQL-db/
[4] http://dataconomy.com/SQL-vs-NoSQL-need-know/
[5] http://en.wikipedia.org/wiki/NoSQL
[6] http://news.dice.com/2012/07/16/SQL-vs-NoSQL-which-is-better/
[7] http://www.dummies.com/how-to/content/nonrelational-databases-in-a-big-data-environment.html
[8] http://planetcassandra.org/what-is-nosql/
[9] https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models
[10] www.julianbrownie.com
[11] SQLVsNoSQL-BattleoftheBackends
[12] www.sitepoint.com/sql-or-nosql-google-app-engine-part-2/
[13] http://en.wikipedia.org/wiki/Graph_database