A Study of NoSQL Database

Biswajeet Sethi[1], Samaresh Mishra[2], Prasant ku. Patnaik[3]
[1,2,3]School of Computer Engineering, KIIT University
Bhubaneswar, India

***Abstract*—Some of the applications of web service 2.0 need big data handling. This requires the existing relational database to scale horizontally in order to achieve demand for high performance, especially for applications which require high scale of user data and of high concurrency. These issues are important consideration for designers to come up with a new group of databases, popularly known as NoSQL. The growing demand for cloud computing and the development of Internet motivates the NoSQL movement. This paper deals with features and data models of NoSQL databases used in cloud computing environment along with strength and limitation of each of the model. In addition this paper talks about classification of NoSQL databases based upon CAP theorem.**

Keywords—NoSQL; CAP; Document Oriente; Column Family; Big Data.

I. INTRODUCTION

Efficient Storage and retrieval of data with availability and scalability is the main purpose of NoSQL databases. NoSQL does not stand for no to SQL; it means “NOT ONLY SQL” [10]. NoSQL database is just an alternative to traditional relational database. The industry of database has seen an introduction of many non relational databases such as MongoDB [11], Hbase [9], Neo4j [8] in last few years. Depending upon the business requirement and strategy a cloud vendor can go with any of the database type. Still some designers of pre relational database claim the NoSQL databases not to be efficient enough in handling data integrity. This paper is organized as follows: Section 2 describes the importance of NoSQL databases. Section 3 highlights on the NoSQL data models. Section 4 highlights on the transaction in NoSQL databases. Section 5 puts light on the comparison for NoSQL databases. Finally, we conclude this paper in Section 6.

II. IMPORTANTCE OF NOSQL

A. Background

For last couple of years, SQL vs. NoSQL has been emerged as a heated argument over the Internet. The argument “SQL vs NoSQL,” actually talks about relational versus non-relational databases. Because of normalized data model and enforcement of strict ACID properties, traditional relational database is considered to be a schema based transaction oriented database. It requires a strict predefined schema prior to storing data into it. Redefining a schema in case of a future change, once after data got inserted into the database is disruptive. Whereas in the era of Big Data, there is a constant need for adding new types of data to enrich the applications. Again the storage solution of relational database can make a big impact on speed and scalability. Web services like Amazon and Google have terabytes and petabytes of data stored in their big data centers and have to respond to massive read-write requests without a noticeable latency. To scale a relational database, data needs to get distributed on multiple servers. Before providing to the application the desired information has to be collected from many tables and combined. Similarly while writing data also; it has to be performed on many tables in a coordinated manner. For any application, it could be a bottleneck to handle tables across multiple servers. In relational databases ‘join’ operation slowdowns the system to a crawl, especially when millions of users are doing lookups against tables with millions of rows of data. Large scale web services such as Google, Amazon, Yahoo, Facebook found these to be the cases to develop their own non-relational database in order to meet the scalability and performance needs.

B. Features of NoSQL

NoSQL databases may not require a predefined table schema, typically scale horizontally and usually avoid join operations. Because of schema less nature and involvement of smaller subset analysis of NoSQL system, this database can be better described as structured data stores. Three important basic features of NoSQL databases are scale-out, flexible data structure and replication, which are explained as follows.

- *Scale-out:* Scaling out refers to achieve high performance in a distributed environment by using many general-purpose machines. NoSQL databases allow the distribution of the data over a large number of machines with a distributed processing load. Many NoSQL databases allow automatic distribution of data to new machines when they are added to the cluster. Scale-out is evaluated in terms of scalability and elasticity.

- *Flexibility:* Flexibility in terms of data structure says that there is no need to define a schema for databases. NoSQL databases do not require a predefined schema. This allows the users to store data of various structures in the same database table. However, support for high-level query languages such as SQL is not supported by most of the NoSQL databases.

- *Data Replication:* One of the features of NoSQL databases is data replication. In this process a copy of the data is distributed to different systems in order to achieve redundancy and load distribution. However there is a chance of losing data consistency among the replicas. But it is believed that sometimes this consistency may be achieved eventually. Consistence and availability are the factors for evaluating replication [3].

III. NOSQL DATA MODELS

These are some categories of NoSQL database models discussed as follows [1][2][4].

A. *Key-Value Data Stores*

In order to handle highly concurrent access to database, the category of NoSQL designed is key-value stores. It is the simplest, still the most powerful data store. In a key-value store each data consists of a pair of a unique key and value. In order to save data a key gets generated by the application and value gets associated with the key. And this key-value pair gets submitted to the data store. The data values stored in key-value stores can have dynamic sets of attributes attached to it and is opaque to the database management system. Hence key is the only means to access the data values. The type of binding from the key to value depends on the programming language used in the application. An application needs to provide a key to the data stores in order to retrieve data. Many key-value data stores use a hash function. The application hashes the key and find out the location of the data in the database. The key-value data stores are row focused. Which means it enables the application to retrieve data for complete entities.

Fig.1 describes retrieval of data from a key-value database. The application has specified a key 'Emp102' to the data store in order to retrieve data. Using the hash function the application hashes the key in order to trace the location of data in the data store. The design of the key should support the most frequent queries fired on the data store. Efficiency of the hash function, design of the key and size of the values being stored are the factors which affect the performance of a key-value data store. The operations performed on such data stores are mostly limited to read and write operations. Because of the simplicity of the key-value data store, it provides users with fastest means of storing and fetching data. All other categories of NoSQL are built upon the simplicity, scalability and performance of key-value data stores. Redis, Voldemort and Membase database systems are examples of prominent key-value data stores.

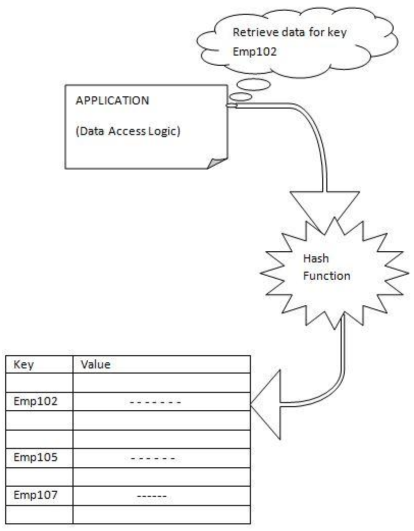
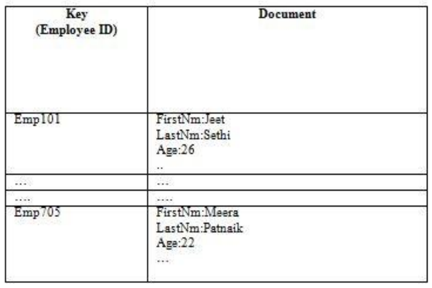


Fig. 1. An example of a key-value data store.

B. *Document Oriented Data Stores*

At an abstract level document oriented database is similar to key-value data store. It also holds value, which an application can read or fetch by using a key. Several document databases automatically generate the unique key while creating a new document. A document in a document database is an entity, which is a collection of named fields. The feature which distinguishes the document oriented database from a key–value data store is transparency of the data held by the database. Hence the query possibility is not restricted with the key only. In order to support scenarios where the application requires querying the database not only based on its key but also with attribute values, can switch for document databases. A document needs to be self-describing in a document oriented database. Information is stored in a portable and well understood format such as XML, BSON or JSON.

As shown in Fig. 2, the document database stores data in form of key-value pairs. But the data stored in the database is transparent to the system unlike key-value databases. The application can query the database not only with the key i.e. 'Employee ID' but also with the defined fields in the document i.e. FirstNm, LastNm, age etc. Document data stores are efficient approach to model data based on common software problems. But it comes at the cost of slightly lower performance and scalability in comparison to key-value data stores. Few of the most prominent document stores are Riak, MongoDB [11], CouchDB.



Key (Employee ID)	Document
Emp101	FirstNm:Jeet LastNm:Sethi Age:26 ..
…	…
….	….
Emp705	FirstNm:Meera LastNm:Patnaik Age:22 …

Fig. 2. An example of a document data store.

C. *Column Family Data Stores*

Sometimes an application may want to read or fetch a subset of fields, similar to the SQL's projection operation. Column family data store enables storing data in column centric approach. The column family data store partitions the key space. In NoSQL a key space is considered to be an object which holds all column families of a design together. It is the outer most grouping of the data in the data store. Each partition of the key space is known to be a Table. Column families are declared by these tables. Each column family consists of number of columns. A row in a column family is structured as collections of arbitrary number of columns. Each column is a map of a key-value pair. In this map, keys are the names of columns and columns themselves are the values. Each of these mappings is called a cell. Each row in a column-family database is identified by a unique row key, defined by the application. Use of these row keys makes the data retrieval quicker. In order to avoid overwriting of the cell values few of the popular column-family databases add timestamp information automatically to individual columns. Every time there is an update, it creates a new version of the cells which have been affected by the update operation. Always the reader reads the value which is last written or committed. A row key, column family, column and timestamp constitute a key. Hence the exact mapping can be represented as

(row key, column family, column, timestamp) -- > value.

A generalized structure of a column family database has been shown in Fig. 3 as follows.

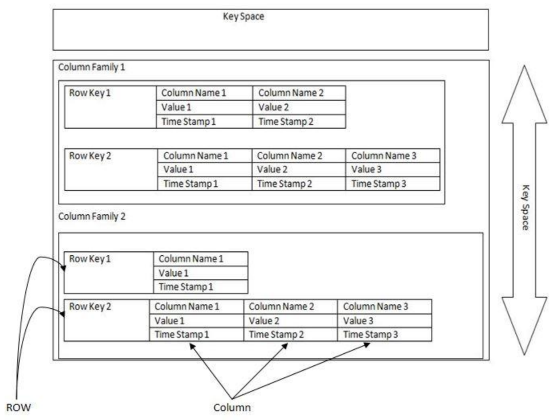


Fig. 3. Column-family data store.

This data model has been popularly accepted as "sparse, distributed, consistent multidimensional sorted map" [4]. An advantage of using a column family data store over a traditional database is in handling NULL values. In a relational database, when a value for an attribute is not applicable for a particular row, NULL gets stored. While in a column family database the column can be simply removed for corresponding row in case the data is not available. That's why Google calls it a sparse database. One of the key features of this database is that it can be distributed in billion of cells over thousands of machines. The cells are sorted on basis of row keys. Sorting of keys allows searching data for a range of keys. Since the data in such kind of model get organized as a set of rows and columns, representation wise this database is most similar to the relational database. But like a relational database it does not need any predefined schema. At runtime, rows and columns can be added flexibly but oftentimes the column families have to be predefined, which leads the data store to be less flexible than key-value or document data stores. Developers should understand the data captured by the application and the query possibilities before deciding the column families. A well-designed column-family database enables an application to satisfy majority of its queries by visiting less number of column families as possible. Compared to a relational database holding equivalent amount of data, a column family data store is more scalable and faster. But the performance comes at the price of the database being less generalized than a relational database as it is designed in support for a specific set of queries. Hbase [9] and Hypertable database systems are based on the data model described above. Whereas another database system Cassandra differs from the data model, as it is having a new dimension added called super column [1]. As shown in Fig. 4 a super column consists of multiple columns. A collection of super columns along with a row key constitute a row of a super column family. As in columns, the super column names and the sub column names are sorted. Super column is also a name-value entity but with no timestamps.

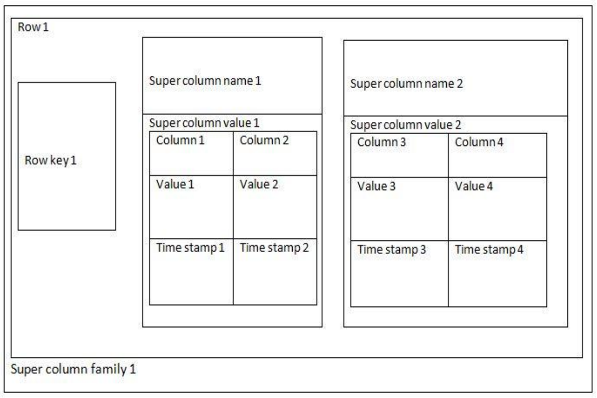


Fig. 4. Column-family data store (Cassandra).

D. Graph Database

Graph databases are considered to be the specialists of highly linked data. Therefore it handles data involving a huge number of relationships [8]. There are basically three core abstractions of graph database. These are nodes, edges which connect two different nodes, and properties. Each node holds information about an entity. The edges represent the existence of relationship between the entities. Each relationship is having a relationship type and is directional with a start point (node) and an end point. The end point can be some other node than that of the start node or possibly the same node. Key-value properties are associated not only with the nodes but also with the relationships. The properties of the relationships provide additional information about the relationships. The direction of the relationship determines the traversal path from one node to the other in a graph database.

Fig. 5 represents a part of the 'Employee' database structured as graph database. Each node in this graph database represents an employee entity. These entities are related with each other through a relationship of relationship type "knows". The property associated with the relationship is "Duration". The key difference between a graph and relational database is data querying. Instead of using cost intensive process like recursive join as in relational database, graph databases use traversal method. While querying through graph database, a start node has to be specified by the application. Traversal starts from the start node and progresses via relationships to nodes connected to the start node, based upon some rule defined by the application logic. The traversal method involves only nodes which are relevant to the application not the entire data set. Hence, a huge increase in number of nodes does not affect the traversal rate much. Social networking, data mining, managing networks, and calculating routes are few of the fields where graph database has been used extensively. Neo4j [8], GraphDB are popular graph databases in use today.

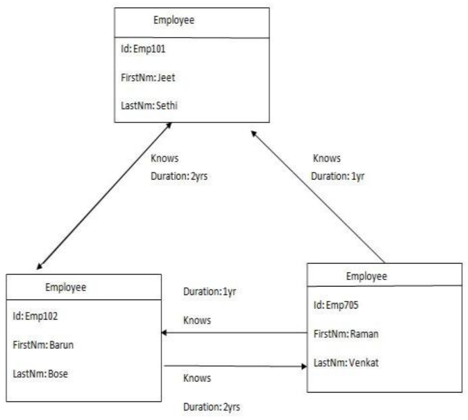


Fig. 5. An example of graph database.

IV. TRANSACTION IN NOSQL DATABASES

When we talk about SQL vs. NoSQL, the competition is actually not between the databases. The comparison is between the transaction models of both the databases. Transaction is defined to be the logical unit of a database processing formed by an executing program. The transaction of SQL database is based upon strict ACID properties. Where ACID is the abbreviation for Atomicity, Consistence, Isolation and Durability. But designers of the NoSQL database came up with a decision, that ACID property is too restrictive to achieve the demands of big data. Hence Professor Eric Brewer in the year of 2000 came up with a new theorem known as CAP theorem [2]. CAP is the abbreviation for Consistency, Availability and Partition tolerance. The theorem says that the designers can achieve any two of these properties at a time in a distributed environment. The designers can ensure Consistency and Availability at the cost of Partition tolerance, i.e. CA based database. If the designer goes for availability and partition tolerance at the cost of Consistency, then it is an AP based database. And if ensure Consistency and Partition tolerance at the cost of availability then the database is CP based. The transaction of NoSQL can be classified as follows.

- *Concerned about consistency and availability (CA):* This kind of database system ensures its priority more towards data availability and consistency by using replication approach [2]. Part of database doesn't bother about partition tolerance. In case of occurrence of a partition between nodes, the data will go out of sync. The relational database, Vertica, and Greenplum database systems fall under such category of databases.

- *Concerned about consistency and partition tolerance (CP):* The priority of such database system is to ensure data consistency. But it does not support for good availability. Data gets stored in distributed nodes [2]. When a node goes down, data becomes unavailable to maintain consistency between the nodes. It maintains partition tolerance by preventing resynchronization of data. Hypertable, BigTable, HBase are few database systems which are concerned about CP.

- *Concerned about availability and partition tolerance (AP):* The priority of such database system is to ensure data availability and partition tolerance primarily. Even if there is a communication failure between the nodes, nodes remain online. Once after the partition gets resolved, resynchronization of data takes place, but without the guarantee of consistency. Riak, CouchDB, KAI are few databases which follow this principle.

Afterwards CAP theorem gets expanded into PACELC [1]. PACELC is an abbreviation for partition, availability, consistency, else, latency, consistency. According to this model the tradeoff between availability and consistency is not only based upon partition tolerance, but it is also dependent on the existence of network partition. It suggests latency to be one of the important factors, since most of the distributed database systems use replication technology for ensuring availability. Later eBay introduced a new theorem known as BASE theorem [3]. BASE aims to achieve availability instead of consistency of databases. BASE is the abbreviation for basically available, soft state and eventually consistent.

- *Basically Available:* Basically available says that even if a part of the database becomes unavailable, other parts of the database continue to function as expected. In case of a node failure, the operation continues on the replica of the data stored in some other node.

- *Soft State:* Soft state says that on the basis of user interaction a data may be dependent on time. These data may also have possible expiration after a certain period of time. Hence to keep the data relevant in a system it has to be updated or accessed.

- *Eventually Consistent:* Eventual consistency says after any data update, data may not become consistent across the entire system but it will become consistent with time eventually. Therefore, the data is said to be consistent in the future.

V. COMPARISION OF NOSQL DATABASES

There is not any hard and fast rule to decide which NoSQL database is best for an enterprise. Business Model, strategy, cost and transaction model demand are few of the important factors that an enterprise should consider while choosing a database. Following are few of the facts which may help in choosing a database for an enterprise.

- If the applications simply store and retrieve data items which are opaque to the database management system and blobs by using a key as identifier, then a key-value store is the best choice. But if the application likes to query the database with some attribute value other than the key, it fails. Also while updating or reading an individual field in a record key-value store is a failure.

- When applications are more selective and need to filter records based on non-key fields, or retrieve or update individual fields in a record as it, then document database is an efficient solution. Document data stores offer better query possibility than key-value data stores.

- When the applications need to store records with hundreds or thousands of fields, but retrieves a subset of those fields in most of the queries that it performs, in that case column-family data store is an efficient choice. Such data stores are suitable for large datasets that scale high.

- If the applications need to store and process information on heavily linked data with highly complex relationship between the entities, graph database is the best choice. In a graph database, entities and relationship between the entities are treated with equal importance.

Table 1 represents a list of databases, their corresponding data models, along with transaction model and query language used by these databases. Cassandra for facebook, HBase [9] for Google, DynamoDB for Amazon is few of the databases which were developed by different companies in order to meet their demand for high data storage requirement. On the other hand database systems such as Neo4j, Riak, and MongoDB were developed in order to serve other organizations. In terms of transaction model, most of the databases such as DynamoDB, Riak, Cassandra and Voldermort give more preference to availability over consistency. Whereas Tokyo Cabinet, Hbase prefer consistency over availability. NoSQL database was designed in order to handle large volume data processing, excluding some of the support system of RDBMS like ad-hoc query. Though many of NoSQL databases mentioned in Table. 1 support ad-hoc queries but the level of programming expertise in writing queries needs to be much higher than that of a relational database.

TABLE I. A COMPARISION OF DIFFERENT NOSQL DATABASES

Database Tool	**Data Model**	**Transaction Model (CAP)**	**Ad-HOC query**
DynamoDB	Key-value	AP	Built in API
Riak	Key-value	AP	CorrugatedIron
Voldermort	Key-value	AP	No
Tokyo Cabinet	Key-value	PC	No
CouchDB	Document	AP	Cloudant, Lucene
MongoDB	Document	AP	BSON based format
RavenDB	Document	ACID	Built in, Limited
Cassandra	Column-family	AP	HIVE, PIG
Hbase	Column-family	PC	HIVE, PIG
Neo4j	Graph	CA	Chyper

VI. CONCLUSION

In the database domain the NoSQL database is considered to be quite new. However these are being developed on known and existing theory. NoSQL databases systems still have various limitations. There is neither a common standard nor any common and familiar query language for querying NoSQL databases. Each database behaves in a different way and does things differently. Relatively these databases are immature and constantly evolving. NoSQL database does not support strict ACID properties, hence there is no guarantee that all data will be written successfully to the data store. This paper describes the limitation of relational database along with different categories of NoSQL data models. Since there is no evaluation available to find the right tool, this paper compares the strength and limitation of each the data model. Limitations of NoSQL databases and its use in a cloud computing environment are the areas which need detailed research in future.

REFERENCES

[1] Maria Indrawan, "Database Research: Are We At A Crossroad?," 15th International Conference on Network-Based Information Systems, pp. 45-48, 2012.

[2] Jing Han, Haihong E, Guan Le,Jian Du, "Survey on NoSQL Database," IEEE, pp. 363- 366, 2011.

[3] Shalini R., Savita G., Subramanian A., "Comparison of Cloud Database: Amazon's SimpleDB and Google's Bigtable," International Conference on Recent Trends in Information Systems, IEEE, pp. 165-168, 2011.

[4] R Hecht, S Jablonski, "NoSQL Evaluation," International Conference on Cloud and Service Computing, IEEE, pp. 336-338, 2011.

[5] Alexandru Boicea, Florin Radulescu, Laura Ioana Agapin, "MongoDB vs Oracle - database comparison," Third International Conference on Emerging Intelligent Data and Web Technologies, 2012.

[6] Jing Han, Meina Song and Junde Song, "A Novel Solution of Distributed Memory NoSQL Database for Cloud Computing," 10th IEEE/ACIS International Conference on Computer and Information Science, 2011.

[7] Guoxi Wang,Jianfeng Tang, "The NoSQL Principles and Basic Application of Cassandra Model," IEEE, pp.1332-1333, 2012.

[8] Neo4j, http://neo4j.org.

[9] Hbase, http://hbase.apache.org.

[10] Mahdi Negahi Shirazi,Ho Chin Kuan,Hossein Dolatabadi, "Design Patterns to Enable Data Portability between Clouds' Databases," 12th International Conference on Computational Science and Its Applications, pp. 117-118, 2012.

[11] Mongodb, http://www.mongodb.org.