NoSQL Database Evaluation with Applications

Akshay Gupta
G21030682
BIGDATA
THE GEORGE WASHINGTON UNIVERSITY

Abstract

We are well aware about the increase of volume of data globally and realizing the shortcoming to deal with such volumes with varying data formats. This generation of data is the result of data mainly come from user related information for example bank details, transactions, messages and more. In order to make the useful meaning from this type of data both data developer and data analytics require better tools at this era. As developer stage or receives data through various platform needs to device an approach so that every datatype can be aggregated while it is presented to an analyst. NOSQL tries to bridge the gap here for both the developer and analyst. It is hard to use the new database technology due to less developed community support and need to prove its dominance with respect to robustness established by RDBMS. We see that many applications are making use of NOSQL and it is anticipated NOSQL would drastically increase the performance for Bigdata in our near future.

Keywords: RDBMS, Map-Reduce, sharding, aggregate data models, cap theorem, Cassandra, MongoDB, Google’s Bigtable, Neo4J, SOA.

1. Introduction

Through the quantity analysis of the Big Data one can easily say that the rate of increase of world's data is doubling per year. This directly effects the size of the databases which are expanding at the exponential rates as well. It is very well anticipated that such expansion of the database will only surge in the coming times, with the fact that storage cost would drop steadily followed by an abrupt increase in the storage capacity.

1.1 Problem

Previously, a terabyte database was considered to be large, but at this point of time they are often considered as small, and now, in most of the big firms or the firms whose primary job role is record keeping roughly adds terabytes or even more data in their databases. The amount and type of data such as unstructured data, structured data, and semi-structured data is proliferating. Mobile applications, web technologies, social media always generates unstructured data that had led to the advent of various NoSQL databases (Puangsaijai, 2017).

The increase in data volume raises another immediate concern that is the processing time of the queries and the approach of sequential processing will not be able to deal with this scenario. For example, if we take a data rate of 1 terabyte per second, reading through a petabyte database will take more than 10 days roughly. To process time-series data in real time, many organizations are looking beyond the traditional data warehouse solutions and into emerging big-data technologies such as open source Map-Reduce frameworks or NoSQL databases (Burtica, 2012). Simultaneously finding heuristic approach to address the volume, variety and veracity of structured and unstructured data. In order to process these enormous databases, we require in depth-knowledge of how high-performance systems and parallelism work. The spread of data inside the databases across the globe is not limited to the grid environment but it equally involves storing data over cloud. And hence, we need to make sure we are well versed in cloud computing as well. The major problems which will be addressed in this paper would be the properties of the NoSQL database that directly affects our earlier stated problems. And some applications of NOSQL where NOSQL might be dominant to solve database issue. In traditional relational database it is mandatory to define your schema before adding any data through the application. But, with changing business logic of our application we need a dynamic schema to suffice the streamline flow. The fact that the relational database stores everything in defined columns works efficiently only with structured data and not with other data types.

1.2 Solution

In order to address the issue of management of data spread across the globe, an effective management approach for such volumes of data would be to allocate multiple resources to it.

This leads to the process of cloud computing where remote servers are hosted on the Internet to store, manage, and process data, rather than a local server.

There is no denying that the SQL databases have played an essential role in to collect information, which is evident through the opensource technologies like MySQL. Relational databases due to their robust nature in scalability, performance and continuous delivery have thrived against the modern contemporaries. All the large-scale websites like google, YouTube and twitter are being powered by the RDBMS based on SQL implementation for the web.

But these schemas do not always support the bigdata requirement which are as follows.

Firstly, the issue of having a dynamic schema as and when an application in upgraded depending on business logic can be resolved by having NoSQL databases and they are also a best fit for hierarchical data due to key-value pair storage format.

Second, we know that it is important the processes are executed as fast as possible and loading huge data of different data types on a relational database is not wise and hence it would be wise to have NoSQL databases that stores documents containing multiple format of data and processing them way faster.

Third, Horizontal scaling or sharding is needed to divide the data set and distributes the data over multiple servers. Redundancy and fault tolerance is achieved by horizontal scaling (Roy, 2018). And, only horizontal scaling can offer aid in such scenario which are best suited with NOSQL databases. That being said, the vertical scaling gives use high performance even on full throttle with low hardware cost.

This paper intends to talk more about the aspects of databases, cloud and distribution of data in depth in the further sections.

2. BACKGROUND

For any company or an individual whether to select the relational (SQL) or no-relational (NOSQL) data structure depending on the given task is very crucial. There are many differences between the both and by learning that one can suitable choose the right technology for his work.

There are four major differences between the both which are explained briefly here.

First, the language used in relational database is structured query language which makes it to be used widely but is also restrictive in nature as data manipulation can be done based on predefined schema. And, on the other NoSQL database provides adjusts itself to the incoming data. It gives each document its own structure.

Second key component is scalability, mostly the horizontal scaling goes with the NOSQL which means handling Big data by adding a greater number of servers to the databases.

Community support for SQL databases has grown much bigger with time with sophisticated chat group and firms. NoSQL on the other hand is relatively new to have such established groups.

Fourth is the structure as we know the relational database works on only rows and tables. NOSQL provides a broad spectrum of structure which includes key-value pairs, wide-column stores, graph databases, or document-based.

From the comparison done so far it is quite evident that NoSQL databases have the ability to adapt to growing data volume and different data types. Making it a suitable choice for Big Data projects.

3. NoSQL

NoSQL simply means not only SQL which implies that whenever someone design a software solution, there are multiple storage mechanism that one can choose from. Although there is no formal perspective definition, but we can have few common observations. According to which the NoSQL does not use relational model, it runs well on cluster, built for the modern era, is open source and is schema- less.

3.1 Why NoSQL databases?

It has become a frustrating process for application developers to find impedance mismatch between the SQL data structures and the in-memory data structures of the application. Having NOSQL databases allow them to develop their logic without worrying about the conversion of in-memory structures to relational structures.

The modern era companies are also moving away from using databases as integration point of encapsulating databases with applications and services.

The vital factor which has brought change in data storage due to rising web as a platform requires runt huge volume of data on clusters. And, SQL databases were not designed to run aptly on clusters.

The data accumulation needs of an enterprise resource planning (ERP) of an application have a multitude of data accumulation needs than the particular activity focused software companies.

To understand this let us consider to models.

3.1.1 Aggregate data models

Since there is a wide gap between relational and type of data structures that an application user use. This very fact has forced the developer community to use data structure as modelled by the developer so that developers can solve different domain problems and hence aggregate models. These aggregate models mostly work on Domain Driven Design (DDD) which defines a methodology to capture relevant knowledge for any software design and helps in isolating domain concepts and simultaneously identifying concept relationship. This makes DDD particularly appropriate for designing microservice architectures, because functional microservices focus on realizing distinct business capabilities (Florian Rademacher, 2018).

An aggregate is nothing, but a collection of data one can analyzes with as a unit. These data units or aggregates form limitation for acid properties in terms of databases. We can say that key-value, column-family and document are forms of aggregate oriented database.

Since in case of aggregates, the unit of data can reside on any machine and when the databases are queried, we get all the data along with it. Hence, aggregates simply data storage management over clusters. The performance of aggregate oriented databases is optimum when maximum data interaction is done on the same aggregate. For instance, in case of getting an order and all the related details, it would be wise if the order object is stored as an aggregate although calling the object for every order detail won't give neat results. Broadly two kind of relationship can be seen in aggregate oriented database models which are inter and inter relationships and, out of the two interaggregate relationships are difficult to handle. Databases with with no aggregate are better when interaction with data is done in many different forms. Aggregate oriented databases may involve computation of materialized views in order to produce data differently from the primary aggregates. This can be seen in a map-reduce computation platform where map-reduce work involves getting items sold per day.

3.1.2 Distribution models

The process of data distribution becomes way easier by using aggregate oriented databases because the distribution process has to forward only the aggregates and not the data related to aggregates. Distribution of data can be done into ways sharding and replication. Sharding distributes a collection of different data and distribute them to multiple servers so that each server can act as the single source for various subset of data. To shard a collection, a DB administrator needs to pick a "shard key", a property chosen to evenly split partitioned data (Kookarinrat, 2015). In the other process that is replication makes multiple copies of data across different services so that each unit of data can be found in different areas. Replication can be divided into two sub forms.

First is master-slave replication, here the authoritative node is in charge of writing with synchronized slave nodes that may handle reads as commanded by the master. Second is peer to peer replication, here any node is allowed to write and synchronize with other nodes to coordinate their cooped of data. The chances of conflict update are reduced in master slave replication but peer to peer replicative helps in avoiding single point of failure due to the fact that write authority is equally distributed amongst each node. Depending on the system requirement the type of replication can be selected. For example, Riak database based on the replication factor replicates and shards the data.

3.2 CAP theorem

For a distributed system, cap suggests that a web service must trade off one of the following consistency, availability, and partition. In sum, CAP can be stated as follows: in a network subject to communication failures, it is impossible for any Web service to implement an atomic read/write shared memory that guarantees a response to every request (Gilbert, 2012). Eric Brewer suggests that we can choose only two from consistency, availability or partition tolerance. Due to this fact many NOSQL databases presents tuning options to user so that they can set it according to their needs. As a case, we consider Riak which is a distributed key-value database. It provides with three fundamental variables w, r, n where

w= node count that should be reverted to write request before going to successful state

r = node count that responds to read request before marking success

n = node count or number of nodes corresponding to replication factor

For a 5 node Riak cluster, we can change w, r, n values to alter the consistency of the system. By setting both r and w equal to 5 we can make the system very consistent but in this case the cluster is vulnerable to network partitions since write success rate is directly proportional to node response rate. In order to make the same cluster highly available for either write or read we should make w =1 and r =1 although the consistency would be downgraded due to low availability of multiple copies. We know from cap theorem that incase of network partition that there would be a tradeoff between availability and consistency of data. Or, durability can be compromised for latency in case someone wants to cope with low quality replicated data. As we know NOSQL databases provides various option and pre tuning capabilities according to users need. So, it becomes important to understand the way your data would be consumed by the system. The user should be able to answer certain questions about their system which are whether it would be a read heavy or write heavy? Are random query parameters required to query the data? What happens to the process when the system encounters inconsistent data?

For a long time, we have been stick to our default RDBMS which have generic features no matter what product one chose. Plus, there is no options to choose certain features and remove any. That being said, we cannot say that these choices would be only good as a case of NOSQL database. It would be advantageous because the features would correspond our system requirements there by removing redundancy. But the bad part would be the fact that we need to be certain when we select the feature, or our system performance would be degraded. Taking the case of our regular RDBMS transactions, the development methods are so consumed to these features that one may not think what happens in the case when transactions does not take place. As of now, NOSQL databases are not involved much in transaction processes. We need to determine and distinct the safety and write feature of transaction that is what data is their which we can loose and what not thereby improving system efficiency. Zookeeper is a reliable distributed coordinator which can be deployed in some cases.

3.3 NoSQL databases

We can categorize NOSQL databases broadly in four categories which are explained below.

NoSQL databases can broadly be categorized in four types.

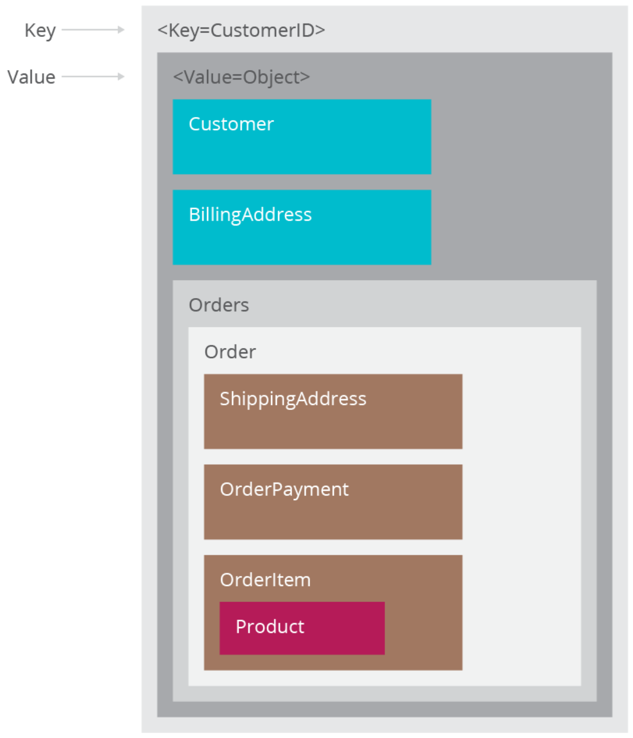


Figure 1: Key-value database

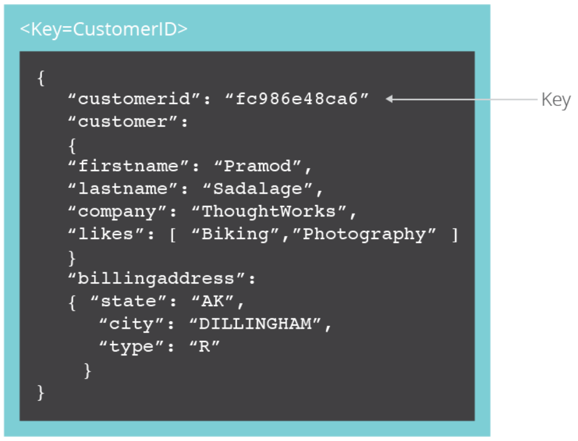
3.3.1 Key-Value database

This is relatively the simplest NOSQL data store to be used from the API point of view. The user can put, delete, or, update the value of the key from the data store. The type of the value that data store is blob, without figuring out what is present inside blow it is the duty of the application to comprehend what was stored in it. Because key-value stores have the feature of primary access, this attribute allocates them high performance and scalability. d can be easily scaled.

These are the example of some widely used key-value databases Riak, Redis (often referred to as Data Structure server), Memcached and its flavors, Berkeley DB, upscaledb (especially suited for embedded use), Amazon DynamoDB (not open-source), Project Voldemort and Couchbase.

There is a huge difference in all the different key-value datastores: Data persistency is missing in mem-cached but not in Riak, which can help Riak where only certain features are required to solve the problem. Consider a case where we have to implement caching of user preferences, execute them in Memcached, which means we lose all the data if nodes go down and the source system is required to be refreshed. Instead of storing the same data in Riak, one may not worry about losing the data, although, we should also come up with the solution of updating the stale data. Hence one must know about different products available for key-pair in order to justice with their application requirement

3.3.2 Document databases



```
{
   “customerid”: “fc986e48ca6”
   “customer”:
   {
   “firstname”: “Pramod”,
   “lastname”: “Sadalage”,
   “company”: “ThoughtWorks”,
   “likes”: [ “Biking”,”Photography” ]
   }
   “billingaddress”:
   { “state”: “AK”,
      “city”: “DILLINGHAM”,
      “type”: “R”
     }
}
```

Fig 2: Document database

Documents are the main concept in document databases. The database stores and retrieves documents, which can be XML, JSON, BSON, and so on. These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values. The documents stored are similar to each other but do not have to be exactly the same (Fowler, P. 93).

These databases store documents in the value are of the key value store, taking the analogy of key-value store in regard to document database where the value is determinable. To have easier transition from relational database one can use MongoDB which is a document-based database and provides rich query language, indexes and more.
constructs such as database, indexes etc allowing for easier transition from relational databases.
Following is list of some popular document databases; MongoDB, Terrastore, CouchDB , RavenDB.

3.3.3 Column family stores

In column family stores data is stored in multiple columns with unique row attribute allocated to row value linked with the column. This kind of data is mostly retrieved in aggregate form. As a general example one may visit a customer profile very often rather than checking his order details.

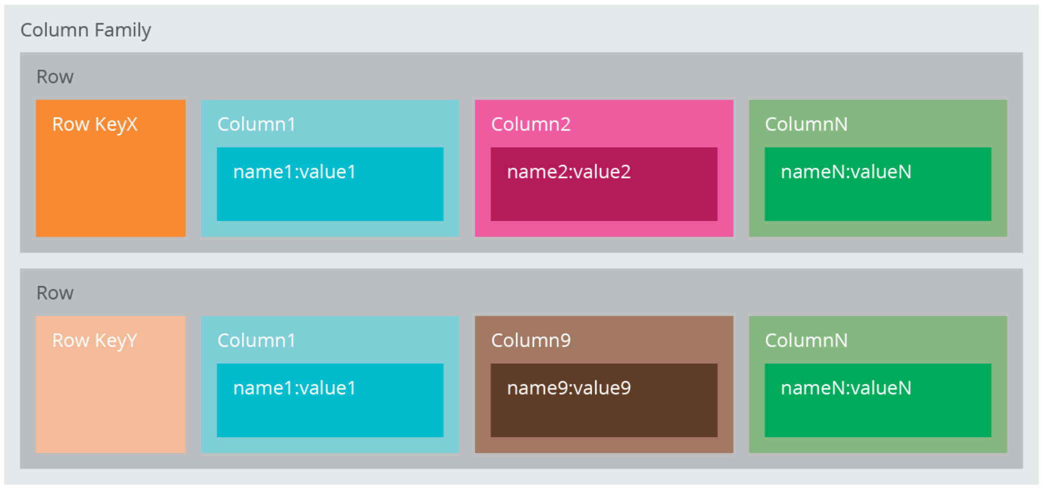


Fig 3: Column family stores

Each column can be viewed as container of rows in SQL table where the key attribute is used to identify the row and the row comprises of multiple columns. The advantage here is that each column is distinct and while adding the values in the table one can add values directly to specific columns without adding the whole row.
A super column is a column which consists of maps of column. In the super column name and value is the map of the columns. Equivalently super column is the container of all the columns.
One of the popular column-family databases is Cassandra, there are also some other popular column-based databases like HBase, Hyper table, and DynamoDB. Cassandra is a distributed,

column oriented, NoSQL database with high scalability, high availability and provides high performance with no single point of failure (Pandey, 2017). Here the cluster has no master node which means read and write operation authority is distributed equally amongst every node.

3.3.4 Graph database

These databases allow to store entities and relationships inside entities, which are nodes having certain properties.Graph databases allow you to store entities and relationships between these entities. Entities are also known as nodes, which have properties. Think of a node as an instance of an object in the application. Relations are known as edges that can have properties. Edges have directional significance; nodes are organized by relationships which allow you to find interesting patterns between the nodes. The organization of the graph lets the data to be stored once and then interpreted in different ways based on relationships.

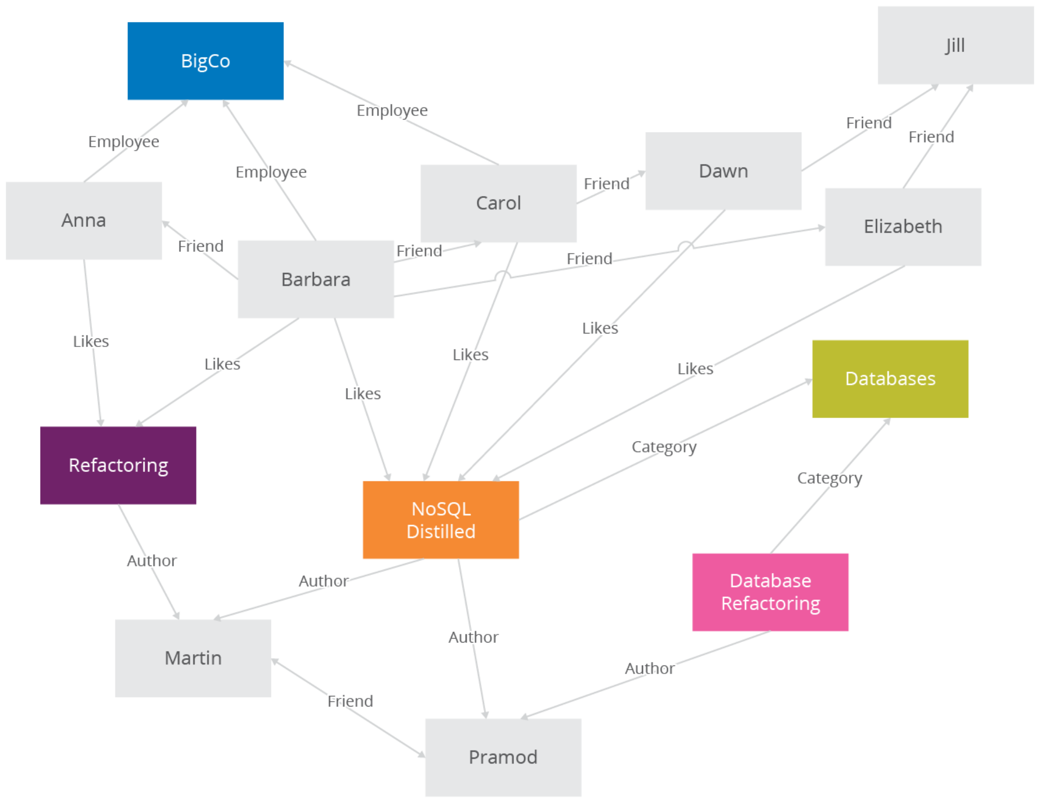


Fig 4: Graph database

Graph databases allow you to store entities and relationships between these entities. Entities are also known as nodes, which have properties. Nodes can be thought of instances of an object. Relations are known as edges that can have properties. Edges have directional property and the organization of nodes is done based on their relationship which allows us to explore meaningful patterns. The organization of the graph lets the data to be stored once and then interpreted in different ways based on relationships.

When someone store a graph like structure in SQL, we get single relationship ("who is the professor" is a common example). Increasing the number of relationships to the previous link leads to drastic data movement and schema change which is the opposite case of using a graph database. Also, in RDMS we have to predefine the graph model depending on the traversal we select; if there is change in data, traversal changes.

In graph databases, traversing the joins or relationships is very fast. The relationship between nodes does not need to be calculated during query time because it persists as a relationship. Traversing persisted relationships is faster than calculating them for every query. Graph databases [3], a kind of NOSQL databases, employ a graph as their data model. Graph database management systems (GDBMSs), such as Ne04j, usually provide a more efficient graph transversal framework with massive scalability (up to billions of nodes, edges, and related properties) [4] (liu, 2013).

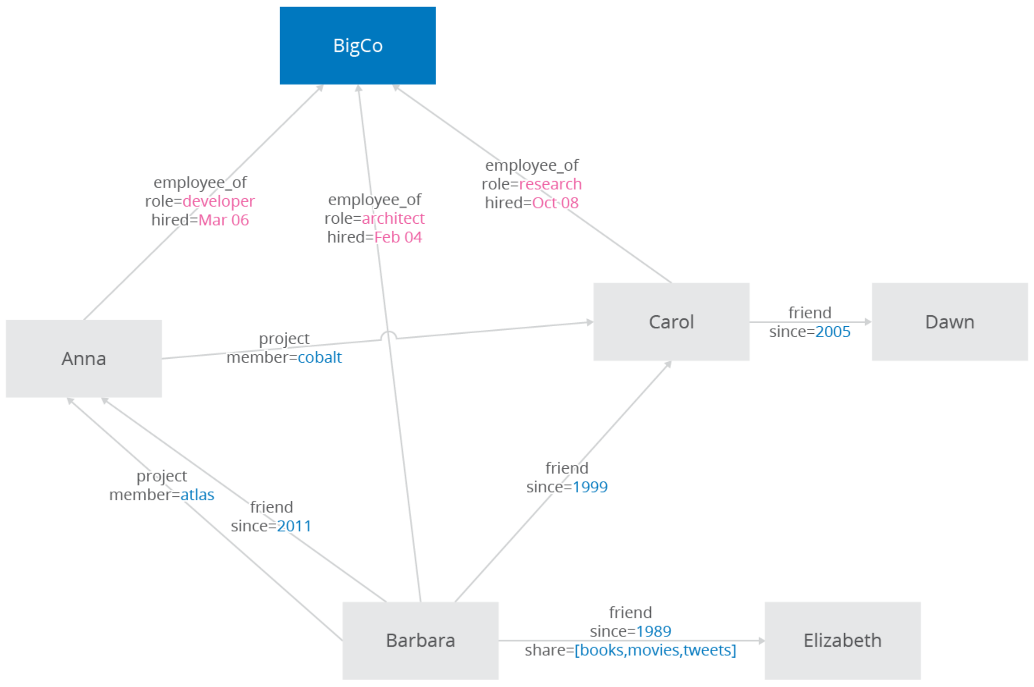


Fig 5: Graph database with added intelligence

There are different types of relationship between nodes, allowing anyone to represent relationship between domain entities and to have other relationships for values like path, quad-trees for spatial indexing, category and linked list for sorted access. There can be any number of relationships between nodes and so all the relationship can be represented in the same graph database.
Relationship is the fundamental entity in a graph as it provides most of the values required. Along with start node, end node, type relationships can have the properties of their own. One can add intelligence to the relationship to obtain necessary details like distance between the nodes, since when they became friends, common attributes between the nodes. These properties are the. Used to query the graph for all possible data. Since the major section of data retrieved from the graph comes from these relationships, it involves in-depth though process and design work to model the relationships. Adding new relationship is not the difficult part; changing existing nodes and their relationship is equivalent to data migration because each node and relationship in the existing data requires changing. Some of the graph databases such as Neo4J, Infinite Graph, FlockDB. FlockDB is a special case, it is a kind of database that supports only adjacency lists and here one can traverse only one level depth in the relationship graph.

3.4 Advantages of NoSQL

3.4.1 Elastic Scaling

It has become a common practice to buy bigger by database administrators in order to perform vertical scaling as the volume of data is increased. It was difficult to allocate multiple host when a big load is encountered. Due to the increase in cost of the hardware and storage needs, the demand is shifting towards scaling out rather than scaling up.

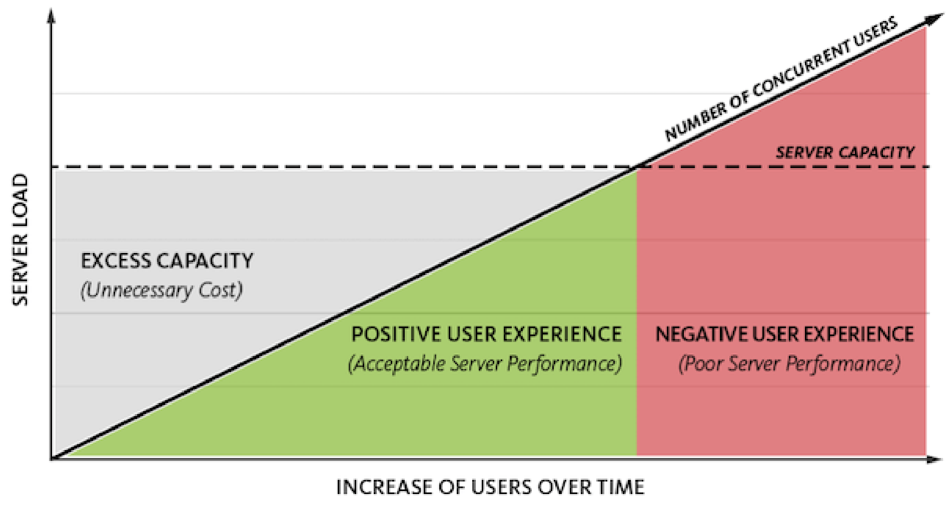


Figure 6: Server load vs increase of users over time

Figure 6 describes the case when number of user increases with time.The same problem occurs when databases move into the cloud. It is hard to scale out RDBMS on commodity clusters. NoSQL databases perform well in this case by growing easily to exploit the advantage of new nodes and using low cost equipment to make nodes.

3.4.2 Big Data

As we know we have to manage data in terabytes today which the consequence of too many transactions. NOSQL although have incorporated new feature practically do not suffice the case. For managing fast incoming data in queue we need a NOSQL database as described previously.

3.4.3 No DBAs

Despite having progressive updates in the new RDBMS, top of the line RDBMS require high skilled and costly DBA's. They are personally required to be present in the outline and continuous tuning of top of the line SQL frameworks. NoSQL databases are mostly composed from the beginning to require less or few administrations, it provides programmed repair, information dissemination, and straightforward information models leading to low organization and tuning needs.

3.4.4 Flexible data models

Even the minute changes to the info model of a SQL database must been viewed thoroughly and it may need less administration level. Document databases and key-value stores helps in storing data of any structure it needs in an informational structure. Databases like Cassandra, HBase, Bigtable allow to make changes without much constraint. Application and data mapping changes are can be done easily in NOSQL database.

3.5 Concern

3.5.1 Overhead and complexity

In general, NOSQL and RDBMS are not integrated and hence NOSQL require manual query programing which may take less time for simple task and too much time for bigger tasks. Also, figuring out complex queries for NOSQL can be a difficult task.

3.5.2 Reliability

RDBMS natively supports the concept of ACID (atomicity, consistency, isolation, and durability) while NOSQL database follow CAP theorem. In order to add ACID properties to the NOSQL database the user needs to do it manual programming although full ACID properties cannot be achieved.

3.5.3 Consistency

Because of the lack of ACID properties in NOSQL at once it has hard to achieve consistency. Compromising consistency enables better performance and scalability but it can cause problem for certain types of applications and transactions, most commonly seen in banks.
We know that there are many types of NOSQL databases, such as Wide Column/Column Families, Document Store, Key Value/Tuple Store, and Eventually Consistent Key Value Store etc; our data needs to partitioned and replicated using consistent hashing, and consistency is facilitated by object versioning [5], we need a suitable consistent hashing algorithm to ensure the consistency of replicas (Xiang, 2010).

3.5.4 Unfamiliarity with the technology

Only individuals who are related to learning NOSQL in depth knows the right product for their system and many companies have no idea of working with it efficiently and hence restraining its growth. That is why the availability of customer support and management tools is very less.

4. Popular NoSQLdatabases

4.1 Google's Bigtable

Bigtable follows the column-oriented storage to have high performance for applications. It can handle structured data as it is a distributed storage system. High scalability of google for bigdata is obtained through Bigtable. Bigtable is flexible regarding various aspects of the data model as it uses “Multi-dimensional sorted Map”. Bigtable serves as the storage platform for Google File System (GFS) which is highly scalable. The whole system comprises of files that are divided into chunks and are involved in multiple machines. The process is to increase the availability of records and reliability of the system on real-time processing. Distributed storage system, hierarchical namespace, its sparse, high scalability, strong consistency, persistence. Multi-dimensional and sorted-map, map are the key features of the Bigtable.

4.2 Amazon's Dynamo database

Dynamo database is a fully managed NOSQL database which gives predictable and high performance with consistent stability. Many services on Amazon’s platform require primary key access to store data which have high-reliability requirements. But it may not be optimum to choose consistency over availability for limited scalability. Dynamo DB is relatively consistent compared to RDBMS. It uses the whole spectrum of Amazon's highly accessible data centers, thus data stored in SimpleDB ensures dispersion of data geographically and routine data manipulation (Rmanathan, 2011).Structured and unstructured, scalable and decentralized, supporting only key-value no hierarchical namespaces or relational schema, efficient latencies, highly-available key-value storage system, availability, consistency, performance, data partitioned using consistent hashing, consistency facilitated by object versioning, trusted network, no authentication, incremental scalability, Symmetry, Heterogeneity, and load distribution are some of the characteristics of the dynamo database.

4.3 Apache Cassandra database

A vastly scalable NOSQL is apache Cassandra using its approach user can analyze data in certain manner due to its potential to attack generated Big Data. This is an open source project established in 2009 to handle bigdata of companies like Google, Facebook, and Amazon to handle bigdata. In recent times Cassandra is used to handle acute ‘Data Infrastructure’. Since it provides more flexibility related to fault handling and data management without compromising on the performance of the system that is why it is preferred over Bigtable, Dynamo and Cassandra. This blend of techniques results in high availability and scalability. Cassandra was designed particularly to resolve the issue of Facebook inbox search problem, letting the user to search through their account’s inbox folder. This was a major problem as millions of users write or search per minute and consequently high throughput would be required. Geographically distributed data centers and data replications assisted in enabling the search invisibility downcast. Recent survey shows that it is used in more of the databases particularly for Facebook and Instagram. Peer-to-peer systems structured and unstructured, decentralized storage system, symmetric system orientation, efficient latencies, linear scalability, map is indexed by a unique “row-key”, “column key” are the key properties of Cassandra database.

4.4 MongoDB

MongoDB belongs to document-oriented database type. The key storage component for the document databases like MongoDB are collections unlike tables in relational databases. MongoDB supports flexible schema, so it does not have any schema structure and performs DML operations without any validation unlike RDBMS (Prabagaren, 2014.) The collection in this database consists of document of JSON, BSON types. Documents with common structure are organized as collections. Predefinitions are not required to make them every time. Documents inside documents or list and array of documents are supported by MongoDB. Dates, arrays, strings or substring all these fundamental datatypes can be present in MongoDB. For a single deployment it has multiple storage engines to offer which helps in moving data between storage engine technologies while utilizing local replication. For ultra-low latency operations this database permits mix of in memory with disk-based motor. Large data can be stored in a schema adaptable away because of the highly scale, accessible and robust framework. But it is possible to have blockage during the write-intensive case due to the master-slave nature of MongoDB.

Following are the features that makes MongoDB a great choice. Flexibility in schema, local programming language and document format. It provides rich query language which includes dynamic, secondary indexes with continuous updates. It is easy to use as it comes with freely available online document. It also provides faster query processing with multiple servers that accounts for consistency.

5. Applications

5.1 Write-heavy enterprise application

Theoretical comparison based on properties of NOSQL are not backed by real studies and hence require real world enterprise system with corporate data to evaluate performance of the NOSQL database. By trading off consistency for availability results different NoSQL systems which have both different architectures and use-cases. We tested Cassandra, MongoDB, Couchbase Server and MS SQL Server and compared their performance while handling demanding and large write requests from a real company with an electrical measurement enterprise system (Lourenco, 2015).

The enterprise system deals with storing several measurements which comes from electrical components at a very high rate. Data is in the form of tuples containing mostly float point, timestamp and metadata types. Currently, there exists some bottleneck whole processing large batches of write- only queries which are passed to central database. It is required to find the feasibility to change the backend components using a NoSQL data structure. This let us quantify real numbers opposed to artificial facts in terms of throughput and storage requirements. It can be seen that although the workload did not reach the size of bigdata many bottlenecks came in the process. Some experiments are conducted to solve these issues based on NOSQL database. The data model used here is the main factor to resolve the issues here.

5.1.1 Present State

In the original case this distributed system uses a centralized MS SQL Server database to store all the data including all the metadata. Data is gathered from every possible resource and written parallelly through batch operations which means thousands of write operation per second. Preprocessing process is very light. Indexing and some redundant business constraints are disabled for such big tables, this portion would be experiment in the latter case. Right now, the distributed system has some bottlenecks which would be removed through NoSQL.

5.1.2 Proposed case – NoSQL

Those party of the original system would be altered that results into bottlenecks. First of all, the tables where most of the data was being written to would be kept in a NoSQL system. In order to take the advantage of horizontal scaling and maximize the write throughput the cluster were ran with sharding and partitioning the data. Instead of the regular pre-defined schema the data would be processed over Cassandra, MongoDB and Couchbase Server. Here we have chosen two document stores (MongoDB, Couchbase) and one wide column store (Cassandra) for improvement.

5.1.3 Experimental setup

The setup involves a homogenous cluster which uses four machines to host the database.

Not only this setup is cost effective, it is adequate to the budget and it mimics the real world scenario. For simulating real data sources two additional machines are used that performs the write -heavy queries. Figure 7 depicts the experimental setup, and Fig 8 summarizes the machine characteristics.

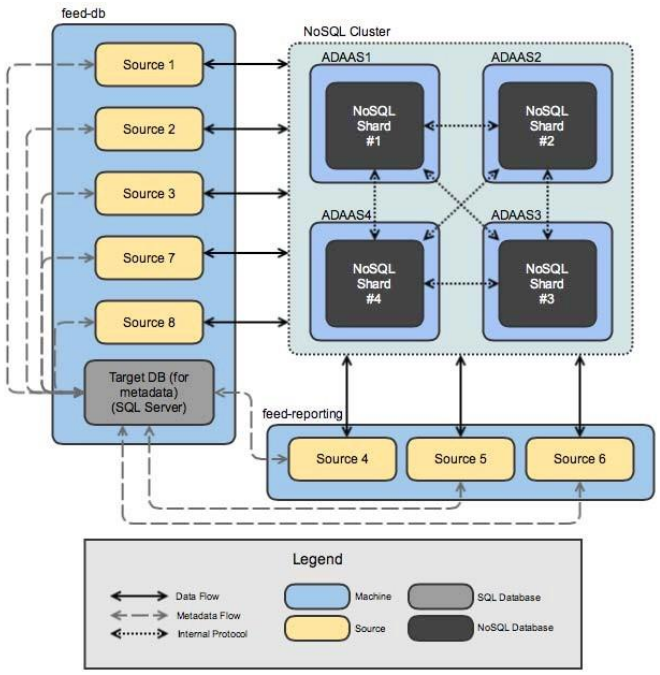
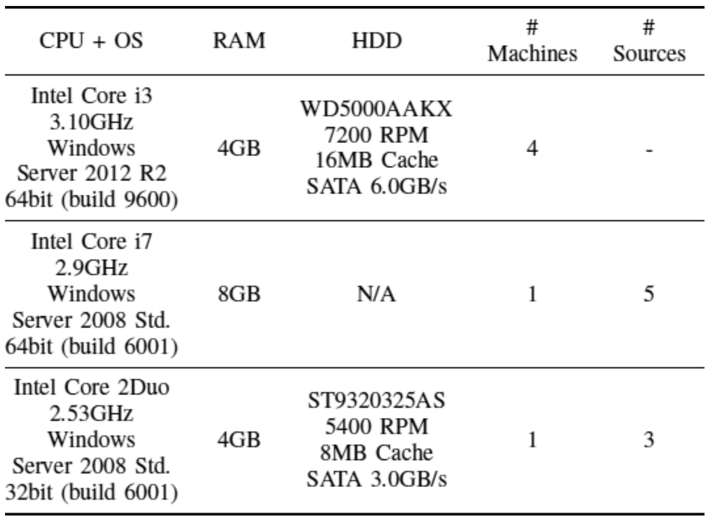


Figure 7: Experimental setup. Retrieved from https://ieeexplore.ieee.org/document/7207274. Reprinted without permission.



CPU + OS	RAM	HDD	# Machines	# Sources
Intel Core i3 3.10GHz Windows Server 2012 R2 64bit (build 9600)	4GB	WD5000AAKX 7200 RPM 16MB Cache SATA 6.0GB/s	4	-
Intel Core i7 2.9GHz Windows Server 2008 Std. 64bit (build 6001)	8GB	N/A	1	5
Intel Core 2Duo 2.53GHz Windows Server 2008 Std. 32bit (build 6001)	4GB	ST9320325AS 5400 RPM 8MB Cache SATA 3.0GB/s	1	3

Figure 8: Machine characteristics. Retrieved from https://ieeexplore.ieee.org/document/7207274. Reprinted without permission.

5.1.4 Results

After the experiment were conducted it was seen that RDBMS outperformed NOSQL which ran on a cluster of 4 nodes with 5 sharding factor. Protocol overhead might be the case, but more experimental analysis is required to prove that case. It is anticipated that twerking in the sharding factor or number of the nodes in the cluster might bring justice to NOSQL but for now SQL outperformed the NOSQL cluster for write heavy datasets.

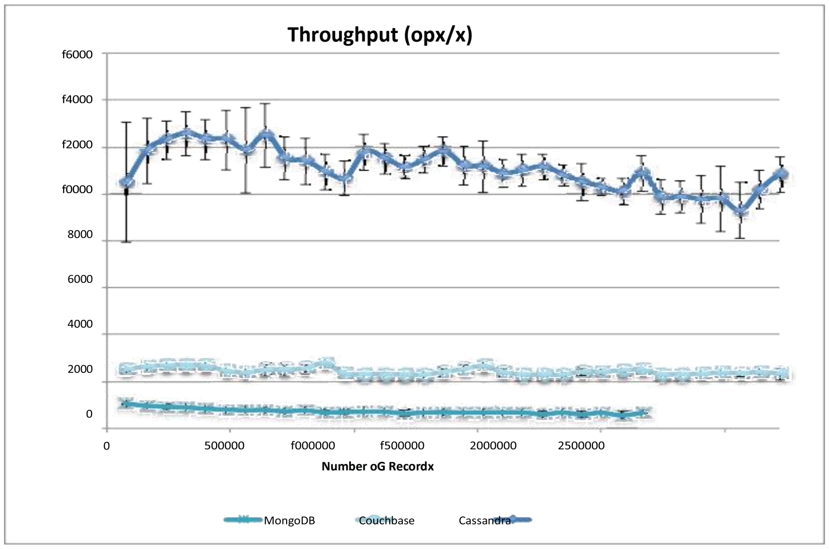


Figure 8: Througput for MongoDB, Couchbase and Casandra. Retrieved from https://ieeexplore.ieee.org/document/7207274. Reprinted without permission.

Although compared to all the NOSQL databases used for conducting this experiment, the column-family database that is Cassandra provided best results. It is anticipated from the shortcomings of the experiment that for large batch insert operation, the NOSQL databases needs to have wider support.

There are also some other points that favored SQL over NOSQL and some changes in the whole experiment setup might prove that NOSQL can perform better. The test server was kept in the distributed SQL which could be changed to clustered server complimenting it with SSD and machines with more resources. The bigdata set is large enough are not is yet to be decided. Although it is anticipated that by increasing the number of applications, the throughput would be reduced due to saturation of network connection and database.

5.2 Study of mental health data with MongoDB

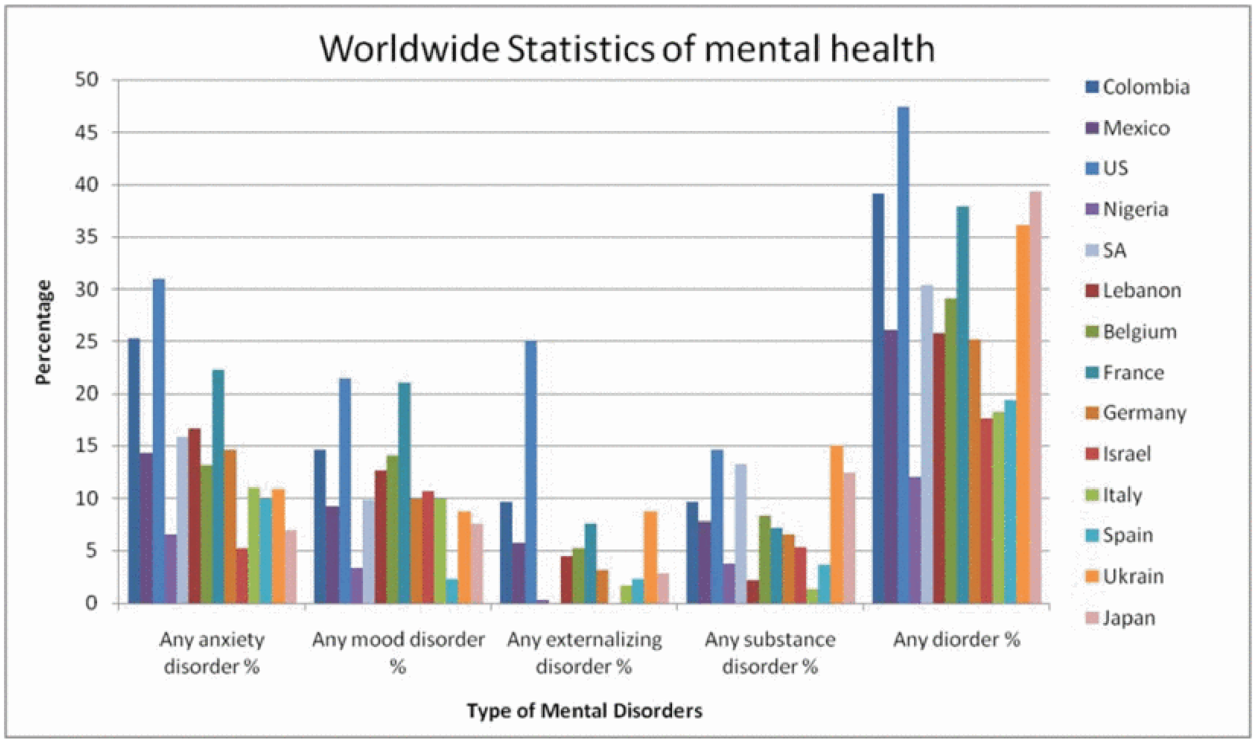


Figure 9: Worldwide statistics of mental health. Retrieved from https://ieeexplore.ieee.org/document/7755300. Reprinted without permission.

5.2.1 Introduction

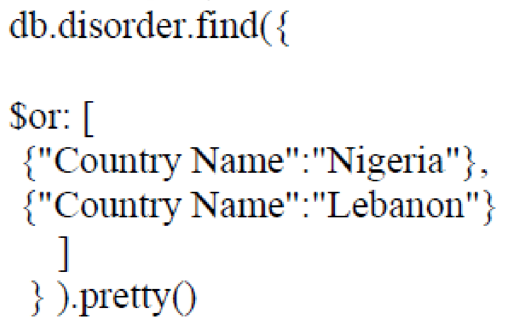
There has been a high growth in mental disorder which has a serious impact throughout the world. So, it of crucial importance to understand the undesirable outcomes of mental health. Mental health is an area with huge need for improvement, as the volume of mental health data is growing, and aim is to improve the classification and diagnostic an analysis to better understand the accumulation of patient data and making the data cost effective because inaccurate and vast amount of data can cause miscalculation, delays and problems (Dhaka, 2016). It requires big data tools to process the large data set of mental issues as the RDBMS couldn’t handle the database with regular techniques. By use adequate data mining approach and robust genetic algorithm, the data was acquired from world statistics of mental health using bigdata tools such as MongoDB. The approach is to analyze different mental disorders and their impact on mental health that requires raw data to be collected to extract clinical decisions. Right drug can only be selected after doing the genomic analysis. It is anticipated that the data structure staged on MongoDB would help to understand the mental issues better.

5.2.2 Algorithm

Selection process helps in finding the subsets from the subset. This set was registered in the database for all the population. And, subset would find related set of information required at particular instant of time. Cross over finds the change in traits and behavior from one person to other. Mutation is the permanent change that alters the DNA completely. So, for large set of genes it required process of deletion and insertion. This being an iterative approach where gene factors are passed on and on requires high staging hardware.

5.2.3 Analyzing Data with MongoDB

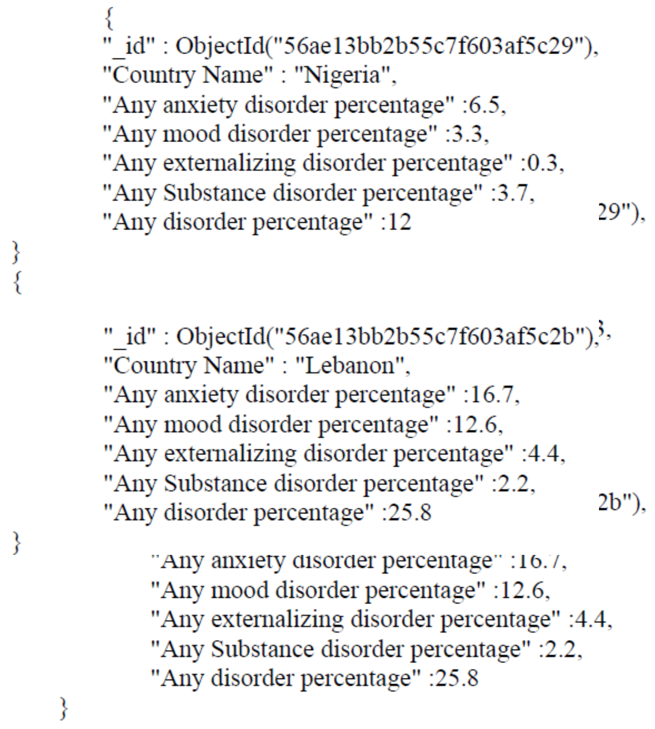
Required information was inserted in the database and unique id's were obtained using command command db.disorder.find().pretty()



```
db.disorder.find({

$or: [
 {"Country Name":"Nigeria"},
 {"Country Name":"Lebanon"}
    ]
  } ).pretty()
```

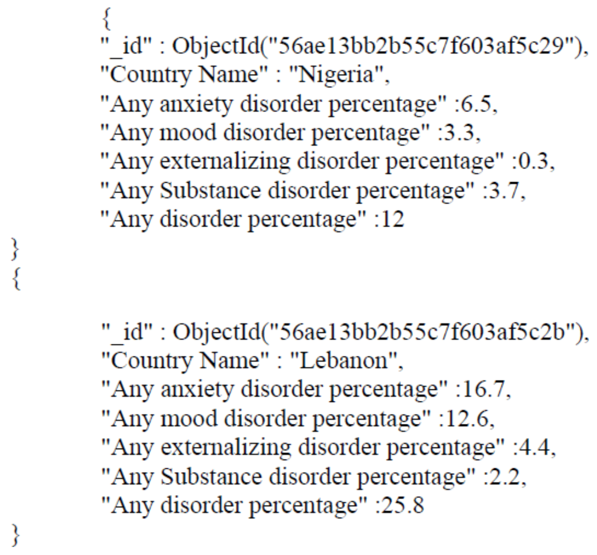
Figure 10: Querying. Retrieved from https://ieeexplore.ieee.org/document/7755300. Reprinted without permission.



```
                {
                "_id" : ObjectId("56ae13bb2b55c7f603af5c29"),
                "Country Name" : "Nigeria",
                "Any anxiety disorder percentage" :6.5,
                "Any mood disorder percentage" :3.3,
                "Any externalizing disorder percentage" :0.3,
                "Any Substance disorder percentage" :3.7,
                "Any disorder percentage" :12                    29"),
        }
        {

                "_id" : ObjectId("56ae13bb2b55c7f603af5c2b"),},
                "Country Name" : "Lebanon",
                "Any anxiety disorder percentage" :16.7,
                "Any mood disorder percentage" :12.6,
                "Any externalizing disorder percentage" :4.4,
                "Any Substance disorder percentage" :2.2,
                "Any disorder percentage" :25.8                  2b"),
        }
                    "Any anxiety disorder percentage" :16.7,
                    "Any mood disorder percentage" :12.6,
                    "Any externalizing disorder percentage" :4.4,
                    "Any Substance disorder percentage" :2.2,
                    "Any disorder percentage" :25.8
            }
```

Figure 11: Results obtained. Retrieved from https://ieeexplore.ieee.org/document/7755300. Reprinted without permission.



```
        {
        "_id" : ObjectId("56ae13bb2b55c7f603af5c29"),
        "Country Name" : "Nigeria",
        "Any anxiety disorder percentage" :6.5,
        "Any mood disorder percentage" :3.3,
        "Any externalizing disorder percentage" :0.3,
        "Any Substance disorder percentage" :3.7,
        "Any disorder percentage" :12
}
{

        "_id" : ObjectId("56ae13bb2b55c7f603af5c2b"),
        "Country Name" : "Lebanon",
        "Any anxiety disorder percentage" :16.7,
        "Any mood disorder percentage" :12.6,
        "Any externalizing disorder percentage" :4.4,
        "Any Substance disorder percentage" :2.2,
        "Any disorder percentage" :25.8
}
```

Figure 12: Updating. Retrieved from https://ieeexplore.ieee.org/document/7755300. Reprinted without permission.

From the figures 10, 11 and 12 we can say that we can aggregate all the information needed to collect varied samples of the mental issues. And it has produced unique relationships that can be leveraged to identify common flaws to mental orders.

5.3 Storage and parallel processing based on Neo4j

5.3.1 Introduction

To solve the problem of environmental pollution and fossil fuels, we use Energy Internet (EI) which is a new energy use system distinguished by mining of new energy technologies and IT based on third industrial revolution.

The Service Oriented Architecture[5] (SOA) is more and more widely used in power system. The article [6] based on the SOA architecture proposed a "device code-scheduling coding" double coding system, the introduction of the concept of Virtual Substation and virtual switch station in the scheduling of the concept of the hierarchical distribution network model (Lv, 2017). Here SOA design and development of various platforms like data exchange, business report system and more. It also makes use of data exchange format, scheduling mechanism and flexible exchange. In power system dispatch, the distributed energy of photons and wind, although, power system being complex the power network scales are exponentially rising in the light of bigdata. The aim is to analyze and characterize the sources of power generation and its transmission along with large data generations. RDBMS being simple and easy does not help to neatly describe the topology of power system. The joint queries of multi tables take a lot on time in real-world scenario.

5.3.2 Traditional System Architecture

This traditional power system data processing mainly make use of the three-layer power system architecture shown in figure.1. Having a private database of the operating status and history data at the data layer; The business logic analysis is realized through the service component in the analysis layer and the interaction with user requests is realized at the application layer.

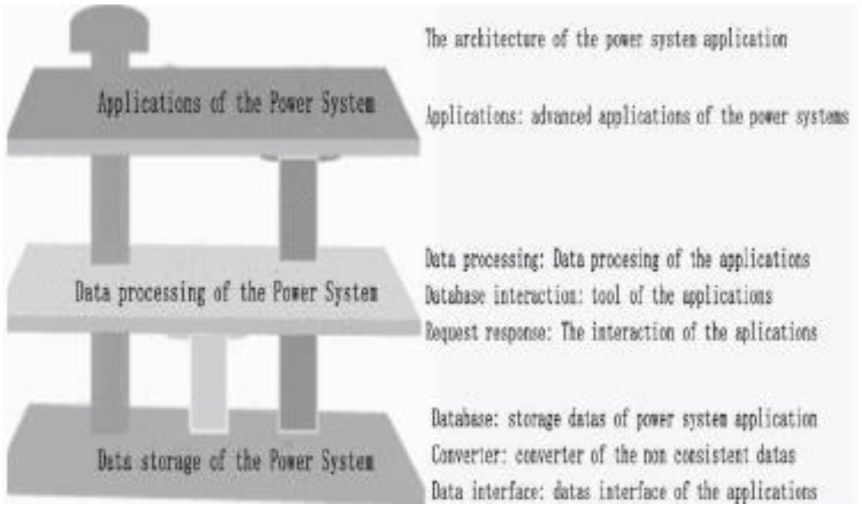

The architecture of the power system application
Applications of the Power System
Applications: advanced applications of the power systems
Data processing of the Power System
Data processing: Data procesing of the applications
Database interaction: tool of the applications
Request response: The interaction of the aplications
Data storage of the Power System
Database: storage datas of power system application
Converter: converter of the non consistent datas
Data interface: datas interface of the applications

Figure 13: Traditional architecture. Retrieved from https://ieeexplore.ieee.org/document/8029061. Reprinted without permission.

5.3.3 System Architecture (SOA)

This is cloud computing-based architecture which uses Hadoop to build of the power system. It uses Hive, flume, HDFS and neo4j to for data management which helps in efficient data storage, conversion and acquisition. Plus, we have Apache Hadoop for parallel computing framework.

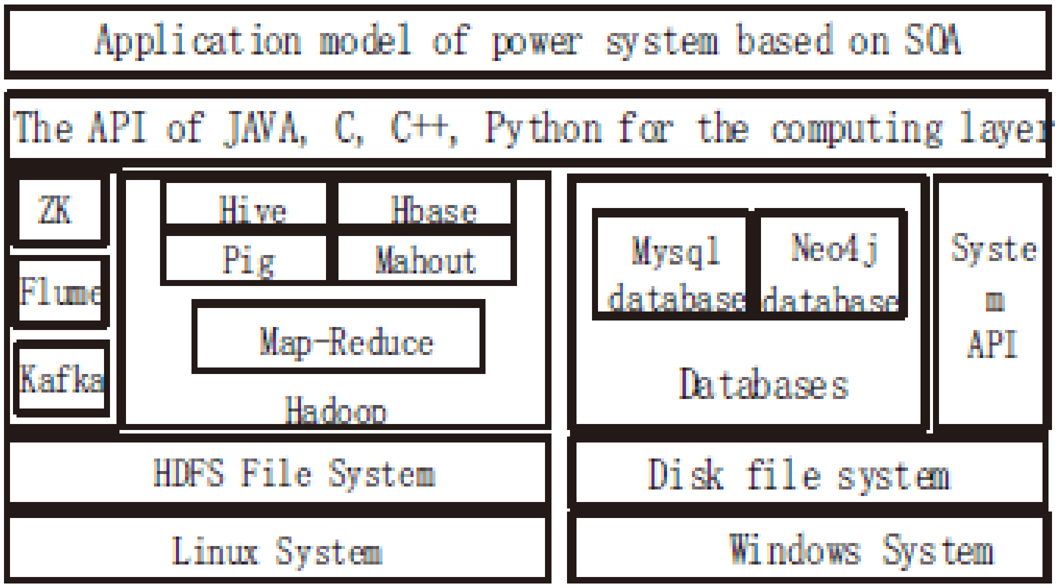


Figure 14: System Architecture (SOA) https://ieeexplore.ieee.org/document/8029061. Reprinted without permission.

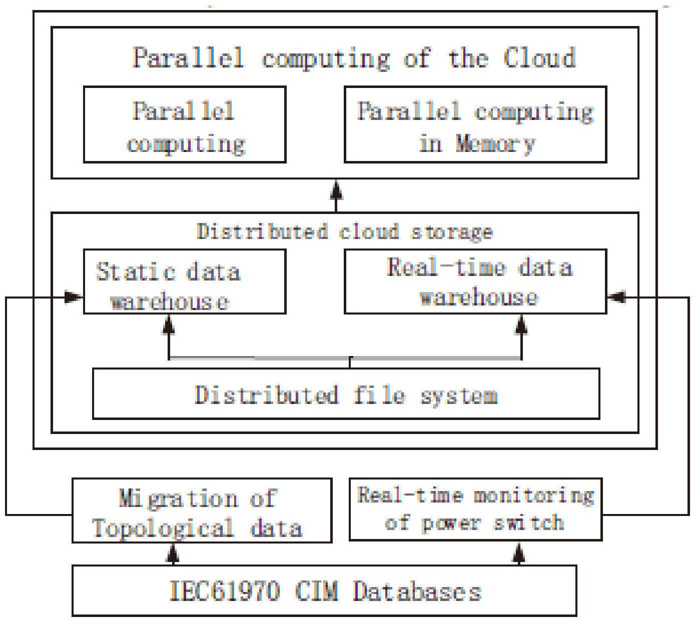

Parallel computing of the Cloud
Parallel computing
Parallel computing in Memory
Distributed cloud storage
Static data warehouse
Real-time data warehouse
Distributed file system
Migration of Topological data
Real-time monitoring of power switch
IEC61970 CIM Databases

Figure 15: Power management system. Retrieved from https://ieeexplore.ieee.org/document/8029061. Reprinted without permission.

5.3.4 Results

It was observed that by increasing the bust test cases the time scale kept stable for SOA architecture. Reason being, the distributed power network storage comprised of Neo4j and Hadoop provide parallel computing cluster support. High throughput, scality with less latency was achieved. Unlike in traditional architecture, here, the performance was poor when network size increased.

Conclusion

This paper discussed about the need of bigdata management. The major problems that cannot be solved over RDBMS databases and hence pushes corporates to explore NOSQL options. We saw that RDBMS has robust nature due to big community support and strong properties like ACID. Then this paper talks about the NOSQL database and how the properties of NOSQL database favors different datatypes through aggregation of structure, unstructured and another datatype together. Then, we discussed the Cap theorem and its tradeoff to achieve either consistency or availability which can be good for cases depending upon the company requirements as the performance is increased by many folds. Sharding, replication (master-slave, peer to peer) helps in obtaining tremendous results. We also discussed some popular databases like googles Bigtable, Cassandra, DynamoDB and MongoDB which solved huge audience data analysis crisis. We saw through applications that NoSQL approach is being adopted which may or may not be good against SQL depending on query complexity and data type. For general cases where data is too much NOSQL prevails over RDMS. It would be great if companies from every tier start experimenting NOSQL databases So that the practical output obtained can justify it’s presence better. Which would also lead to good community support forms for NOSQL platforms and the difficulty to try NOSQL databases would be reduced greatly.

References

Jose, B. (2017). Exploring the merits of NoSQL : A study based on mongodb. 266-271. Retrieved from https://ieeexplore.ieee.org/document/8076778.

Lourenco, J., & Abramova, V. (2015). No SQL in Practice: A Write-Heavy Enterprise Application. 584-591. Retrieved from https://ieeexplore.ieee.org/document/7207274.

Rademacher, F., & Sorgalla, J. (2018). Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective. 36-43. Retrieved from https://ieeexplore.ieee.org/document/8354426.

Gilbert, S., & Lynch, N. (2012). Perspectives on the CAP Theorem. 30-36. Retrieved from https://ieeexplore.ieee.org/document/6122006.

Liu, Y. (2013). Graph Data Warehouse: Steps to Integrating Graph Databases Into the Traditional Conceptual Structure of a Data Warehouse. 433-434. Retrieved from https://ieeexplore.ieee.org/document/6597176.

Xiang, P. (2010). Cache and Consistency in NOSQL . 117-120. Retrieved from https://ieeexplore.ieee.org/document/5563525.

Burtica, R. (2012). Practical application and evaluation of no-SQL databases in Cloud Computing. Retrieved from https://ieeexplore.ieee.org/document/6189510.

Roy, C. (2018). A Proposal for Optimization of Data Node by Horizontal Scaling of Name Node Using Big Data Tools. 1-6. Retrieved from https://ieeexplore.ieee.org/document/8529795.

Kookarinrat, P., & Temtanapat, Y. (2015). Analysis of Range-Based Key Properties for Sharded Cluster of MongoDB. Retrieved from https://ieeexplore.ieee.org/document/7370983.

Pandey, S., & S. (2017). Context Based Cassandra Query Language. Retrieved from https://ieeexplore.ieee.org/document/8204142.

Ramanthan, S., & Goel, S. (2011). Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable. 165-168. Retrieved from https://ieeexplore.ieee.org/document/6146861.

Prabagaren, G. (2014). Systematic approach for validating Java-MongoDB Schema. 1-4. Retrieved from https://ieeexplore.ieee.org/document/7033768.

Dhaka, P. (2016). Big data application: Study and archival of mental health data, using MongoDB. 3228-3232. Retrieved from https://ieeexplore.ieee.org/document/7755300.

Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional.