NoSQL Databases: Modern Data Systems for Big Data Analytics - Features, Categorization and Comparison

Original Scientific Paper

Atul O. Thakare
CSE Department, Koneru Lakshmaiah Education Foundation,
Guntur District, Andhra Pradesh, India
aothakare@kluniversity.in

Abhijeet R. Thakare
MCA Department, Shri Ramdeobaba College of Engineering and Management,
Nagpur, Maharashtra, India.
thakarear@rknec.edu

Omprakash W. Tembhurne
School of Computing, MIT Art Design and Technology University,
Pune, Maharashtra, INDIA
omprakash.tembhurne@mituniversity.edu.in

Soora Narasimha Reddy
CSE Department, Kakatiya Institute of Technology and Science,
Hasanparhty, Warangal, Telangana, India
snreddy75@gmail.com

Abstract *– Because of the massive utilization of the world wide web and the drastic use of electronic gadgets to access the online world, there is an exponential growth in the information produced by these hardware gadgets. The data produced by different sources, such as smart transportation, healthcare, and e-commerce, are large, complex, and heterogeneous. Therefore, storing and querying this data, coined "Big Data," is challenging. This paper compares relational databases with a few of the popular NoSQL databases. The performance of various databases in executing join queries, filter queries, and aggregate queries on large datasets are compared on a single node and multinode clusters. The experimental results demonstrate the suitability of NoSQL databases for Big Data Analytics and for supporting large userbase interactive web applications.*

Keywords*: Unstructured Data, NoSQL Database, Horizontal Scaling, Vertical Scaling, CAP Theorem, Weak Consistency*

1. INTRODUCTION

Traditional Relational and Object-Relational database approaches are ineffective at delivering the flexibility and scalability needed when processing enormous amounts of fast-moving structured, unstructured, and semi-structured data and supporting a large number of concurrent users. NoSQL databases fully handle these problems. The following two subsections go into further detail about these ideas.

1.1. LIMITATIONS OF RELATIONAL DATABASES IN SUPPORTING MODERN WEB APPLICATIONS

Since the beginning of computers, the relational model has been the standard for storing and retrieving data. The limits of relational databases were made clear by the exponential increase in internet users and the widespread adoption of new-generation online applications, which ultimately called for the developing of a new generation of No-SQL databases.

The reason why relational DBMSs are not well suited to support the requirements of modern web applications is their complete dependence on the fixed predefined schema, strict consistency rules (adherence to ACID properties) [1], need for joining several tables for query processing (which is difficult and slow when data grows huge and is stored distributedly across multiple servers), poor performance in terms of availability and scalability while dealing with large volumes of unstructured or semi-structured workloads.

1.2. PROBLEM STATEMENT AND SOLUTION FEATURES OF NOSQL DATABASES

In order to handle massive amounts of data and a very large user base, modern web-based applications have incorporated scale factors that have never been used before. Modern web applications must be able to serve vast numbers of concurrent users, respond quickly to a considerably large user base scattered throughout the globe, provides constant availability, manages a wide range of data, and update swiftly for new up-

dates and features. The advent of these new challenges posed by the term coined as "big data" (made up of structured, semi-structured, and unstructured data and described by its volume, variety, and velocity features) created a need for out-of-the-box horizontal scalability for today's data management systems. Additionally, we are dealing with a new community of applications where flexibility, scalability, availability, and cost are more crucial to application success than consistency in read-and-write operations. To ensure that our application exhibits the features mentioned above, the first requirement is that the data management systems be designed to run on distributed systems.

The last ten years have seen the rise of cloud computing as the most practical and economical alternative for meeting the ever-increasing storage and processing demands of modern online applications. As a result, most web applications nowadays are hosted in the cloud, where the available resources can change size in response to the application's needs. Additional servers can be provided to the website to handle increased demand in the event of a sudden rise in traffic (scaling out) [2], thus ensuring the system's constant availability and good performance. When the traffic returns to normal, the additional servers can be removed. NoSQL systems are faster and better equipped as they can take advantage of "scaling out," i.e., adding more nodes to the distributed system and distributing the additional load over newly added nodes.

The focus of this paper is to present the differences between relational databases and No-SQL databases as well as broad categorization of No-SQL data stores and to compare and analyze three popular No-SQL solutions – Cassandra, MongoDB, and CouchBase, and to highlight their differences with relational databases and other No-SQL data stores.

2. SQL VS. NOSQL

In order to tackle Big Data, the world has moved away from the "one size fits all" philosophy of RDBMS and toward the more flexible approach of No-SQL systems [3]. NoSQL databases give more freedom to design systems by studying the application's features and the type and volume of data it is dealing with, which is the cause for the shift in emphasis from traditional RDBMS to NoSQL database systems for handling Big Data.

2.1. NO-SQL SILENT FEATURES

No-SQL data stores are devised to scale horizontally and run-on commodity hardware. The term "horizontal scalability" means the ability to distribute the data and the processing workload of database operations over many servers with no shared memory. On the other side, "vertical scalability"— the only way of expanding the capabilities of centralized RDBMS systems—means boosting the resources of the specialized server, which comes at a significant cost. Moreover, No-SQL data stores can benefit significantly from cloud infrastructures from an implementation standpoint since they can be scaled and made available according to the application's needs [4].

2.2. DEVIATION FROM RELATIONAL DATABASES

The transactions in relational databases obey the ACID principle, which stands for Atomic, Consistent, Isolated, and Durability [5]. Many rows spanning many tables are updated as a single operation. This operation either succeeds or fails in its entirety, and concurrent operations are isolated from each other, so they cannot see any partial updates. Following these rigid consistency requirements could needlessly limit databases' ability to address the performance issues for a specific application. In distributed databases, following ACID properties strictly for all the transactions where many nodes handle the operations within the transaction creates complications. In relational databases, it is essential not to allow any application to view the inconsistent state of the data. Satisfying strict consistency requirements in distributed databases will force the system to ensure that communication channels maintain strict consistency of data and total synchronization of replica copies with the consistent data across the clusters of nodes. Performing this complicated task at each transaction without compromising on the availability of the system is highly difficult.

Because for each transaction, the system will have to wait till the updates inflicted by the transaction are propagated to all the other nodes hosting some portion of the affected data. Unlike relational databases, many NoSQL systems are not ACID compliant [6]. In some NoSQL databases which bypass the strict consistency rule of RDBMS, within the cluster, all writes are received by the MASTER, and synchronization of the SLAVE replicas with that of the MASTER data is carried out periodically. Hence the updates are eventually propagated. SLAVE nodes are always available to respond to read queries, because of which there are reduced guarantees of returning consistent results for all the read requests. The idea is that by relaxing on the strict ACID constraints, we can enable the database systems to improve on the other desirable characteristics like availability, scalability & fault tolerance. As already discussed, many of today's modern applications won't mind not following strict ACID properties. But, its performance will be badly affected if the uptime for all the requesting clients is not maintained. In replacement to ACID, No-SQL databases follow BASE [7] semantics which are explained below:

- (BA) Basically Available: an application is ready to accept read/write requests all the time.
- (S) Soft state: Results may not always be based on consistent data (no consistency guarantee).
- (E) Eventual consistency: The system assures that data will become consistent at some later point.

When a distributed database system is installed on a group of server computers, high availability may be achieved by maintaining replica copies of the data on several machines within the group and updating those copies whenever a write operation is carried out on the MASTER machine [8]. Consider a scenario in which the network links between the cluster nodes are broken, and the network is divided into several fragments that cannot be reached by one another. In this situation, the database system is kept accessible for clients by making one network segment active and disconnecting the others. Additionally, this stops the nodes of disconnected segments from responding to client queries, preventing the delivery of inconsistent results.

After receiving the missed writes (updates) from the active cluster nodes, the inaccessible cluster segments begin serving the directed traffic of client requests. In a distributed database, the capacity of the clusters to continue operating despite communication breakdowns is referred to as partition tolerance [9]. The CAP theorem, which argues that in partition tolerance, one must choose between consistency and availability, explains the complicated trade-off between consistency and availability in distributed databases [10]. Another way around, the system cannot have all the following three properties at any given time: consistency (all servers having a consistent version of the data), availability (each request receives a timely response), and partition- tolerance (as the information is distributed and replicated, even if there is a failure in a part of the system, the system continues to work). NoSQL systems frequently compromise consistency to some degree to achieve high availability [11].

Table 1. Comparison of RDBMS and NoSQL

Feature	RDBMS	NoSQL
Data	Structured	Structured, Unstructured, Semi-structured
Schemas	Fixed	Dynamic
Scalability	Vertical	Horizontal
Compliance	ACID properties	BASE properties
Architecture	Centralized	Distributed
Consistency	Strict	Eventual
Query Language	SQL	OO API, SQL like
Performance	Slow	Fast
Best suited for	banking, financial transactions	large-scale web applications, Sensor data

2.3. NO-SQL DATA MODELS

Most No-SQL databases run on distributed systems and fall into four categories.

1. Key-Value Stores

a. The data is stored in the form of key-value pairs.

b. Keys are identifiers (unique in the namespace), and values are data associated with the keys. Keys are used for looking up data.

c. For fast lookups, keys are hashed [12].

d. In key-value pair, the value may be data or another key.

e. Supports querying, modifying data through primary key, and mass storage.

f. Provides higher concurrency and higher query speed.

g. Best suited when high-speed and highly scalable caches are needed.

h. e.g., Amazon Dynamo, Azure Cosmos, Riak, Redis, etc.

i. Suitable for applications that use a single key to access data, e.g., online shopping cart.

2. Document Stores

a. Documents contain contents as well as formatting information (JSON, XML).

b. Documents contain information in key-value pairs.

c. Keys are a string of characters, and values can be any basic data type or structure.

d. Collections are the list of documents. Documents in the same collection can follow different structures.

e. A document can have embedded documents or arrays inside it.

f. Supports indexing, designed for scalability and high performance.

g. In addition to CRUD (create, read, update, delete) operations, it supports filtering collections, joining multiple collections, performing groupings, aggregations, etc.

h. Best suited when fast and constantly growing data.

i. e.g., MongoDB, Couchbase, CouchDB, RavenDB etc.

j. Suitable for content management systems, e.g., social networking sites, blogging platforms, etc.

3. Wide Column-Databases

a. In Column- Databases basic unit of storage is a column, i.e., data is organized in column families.

b. Column data is stored one after the another. Hence, the last item of the first column is followed by the first item of the second (next) column, and so on.

c. A query language for column-family databases supports CRUD and creates column-family operations [13].

d. An index is created on columns, reducing the I/O cost for the queries accessing columns of data.

e. More suitable for data warehouses where most of the analytical queries involve aggregations that need to scan data from a few columns but for all the rows.

f. e.g., HBase, Cassandra, Accumulo, Hypertable, etc.

4. Graph Databases (GD)

a. A collection of nodes (vertices) and relationships (edges) tagged with the information forms a graph [14].

b. A node is an object that has an identifier and a set of attributes. A relationship is a link between two nodes that contain features about that relation.

c. It provides fast operations as it models adjacency between objects.

d. Convenient to use for representing social networking media, creating recommendation systems, and pattern mining.

e. It is best suited when the relationship between entities is more important than the entities themselves.

f. The data of popular applications like Facebook, Twitter, LinkedIn, etc., are modeled using graphs.

g. Neo4J, Infinite Graph, InfoGrid, HyperGraphDB, etc.

3. POPULAR NOSQL DATABASES

A few of the popularly used NoSQL databases are described below:

1. MongoDB is developed in the cloud. It is a scalable, open-source No-SQL database that is document-oriented, schema-free, and simple to use. It aims to fulfill the needs of expansive web applications by implementing highly parallel and globally scattered database systems. MongoDB supports auto sharding, where it splits the data collections and stores the different data chunks among the available servers [15]. Additionally, it provides features like high performance, partition tolerance, automatic scalability, and replication (uses Master-Slave replication) [16].

2. Cassandra is a distributed storage system for structured data management that can handle large volumes of data. Scalability, high performance, high availability, high reliability, applicability, and replication are a few of its essential characteristics [17]. Its feature of executing map-reduce jobs in hadoop clusters is ideally suitable for mission-critical applications [18]. It is incrementally scalable, and data is partitioned and distributed among the nodes of a cluster in a fashion that allows repartitioning and redistribution.

3. CouchDB: CouchDB is sometimes called a "Cluster of unreliable commodity hardware." Initially, CouchDB was implemented in C++ but later ported to the Erlang OTP for implementing thoroughly lock-free concurrency of read-write requests. CouchDB databases consist of documents made up of fields with a key, i.e., name and value. The value may be a number, boolean, date, string, ordered list, or associative map [19]. Documents may contain references to other documents (embedded documents). CouchDB is distributed; its other essential concepts are schema-free, views, distribution, replication, map-reduce, etc. The cluster is conveniently scaled horizontally and has no single point of failure. Clusters are designed to allow live changes means there is no downtime during database updates and software-hardware upgrades [20]. If you're scaling reads over numerous servers, a write must happen on them. It also offers incremental replication with bidirectional conflict detection and resolution.

4. HBase is a Hadoop-based open-source system that runs on the fault-tolerant Hadoop Distributed File System (HDFS). HDFS uses a master-slave architecture that consists of name nodes (manages the file system) and data nodes (stores and replicate data) [21]. In a hadoop cluster, coordination between different nodes is maintained by one type of node called Zookeeper (single point of failure for HBase). The zookeeper stores the location of the META table. The client will query the META table to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location. HBase has two types: read cache (BlockCache) and write cache (MemStore). In HBase, data is stored in tables, but it is schema-less. Tables have lexicographically indexed multidimensional row keys and several column families, each having a set of column qualifiers, which stores the fundamental data element [22]. Hence a combination of the table name, row key, column family, and column qualifier define the access. The system stores different versions of a data item, each assigned with an individual timestamp. This feature of HBase attributes to its high write performance. In addition to hadoop services, HBase also has servers for managing metadata about the distribution of table data. The primary storage unit (shard) in HBase is Region which is managed by RegionServers (responsible for administrative activities). As the size of the data grows, new regions are created. The subset of the rows of a column family, ordered by the row key, are assigned to a particular region. An asynchronous write-ahead log (WAL) is used in HBase cluster replication which eventually targets consistency. The META table is an HBase table that lists all regions in the system. This unique HBase Catalog table holds the location of all areas in the cluster.

3.1 COMMON KEY DESIGN CHARACTERISTICS OF NO-SQL DATABASES

Following are the critical characteristics of NoSQL databases that support Big Data Analytics.

1. Scalability: ability to efficiently meet the needs for varying workloads in terms of resources and performance. A Spike in the number of users triggers the application running on a cloud to acquire additional servers. As the spike subsides and traffic becomes normal, additional servers are released [23].

2. Flexibility: Before starting a project, RDBMS database designers must design all the tables, table relationships, and schemas required to support an application. Unlike relational databases, which demand the schema to stand defined before adding any data, No-SQL is schema-less and hence more capable of handling significant variations in the data structures.
3. Availability: Most of the No-SQL systems are almost always ready to accept new read or write requests. In the back end, No-SQL databases are usually deployed on a distributed system consisting of multiple low-cost servers, each having an identical copy (replica) of the database. The large-scale web application also runs on a cluster of servers. As the backup servers keep replicated copies of data from the primary server, if a primary server goes down for maintenance or fails, the secondary servers elect the new primary, and thus the system remains available. In case of a sudden increase in the number of users, cluster expands in terms of compute, storage & bandwidth capacity to maintain system performance.
4. Replication: To prevent automatic fail-over caused by events like server or network failures, MongoDB utilizes an architecture called replica set to distribute copies of data among computers in the cluster. Scaling the number of database reads is another benefit of replication. Database reads in read-intensive applications can be distributed among the computers in the replica set cluster. Replica sets typically include a primary server and a backup server. If a master-slave arrangement is used, the primary server can handle both read and write requests, while the secondary servers can only handle read requests. Each write on the primary will be transmitted to all the secondary servers. In other words, reads from an alternative location will succeed only when they receive all the changes made to the primary. If the primary server fails, one of the secondary servers will be elected as a new primary [24].
5. Cost: Most of the No-SQL databases are available as open source and hence are free, thereby avoiding the issues like licensing, charging the users, etc.
6. Sharding (Partitioning) Data splitting (per-collection basis) over many servers with an emphasis on order preservation will increase performance. To do this, we partition the dataset into various servers and replicate each portion over many servers. We can significantly increase the read and write speed of the system since different users are accessing distinct portions (shards) of the dataset. A server can be a slave for a few shards and a master for few others.
7. Map-Reduce: MapReduce is a programming model in which computations are expressed as a map and reduce functions, which are impulsively parallelized across numerous machines within a cluster-based computing environment. It can perform calculations on large volumes of data in a reasonable amount of time by distributing and parallelizing it across multiple devices in the cluster. The computations take a set of input key-value pairs and produce a set of intermediate key-value pairs accepted by the set of reduced functions. The reduced functions merge the values by grouping using the intermediate keys (using operations like counting, summing, or averaging) to produce possibly smaller values.

3.2 HADOOP BIG DATA FRAMEWORK

- Hadoop is a valuable open-source framework for developing distributed applications that process large amounts of data [25]. Hadoop is intended to run on clusters of commodity hardware (or cloud services such as Amazon's EC2), hence is capable of handling hardware malfunctions and failures. In hadoop, large data sets are divided into more number of smaller (64 MB) blocks which are spread to live in the cluster of several machines using the Hadoop Distributed File System (HDFS).
- HDFS achieves this using its two components, NameNode (stores metadata) and DataNodes (stores portion of actual data). Depending on the degree of replication, replica copies of each block will be maintained in the hadoop. If NameNode is the master daemon, Data Nodes are slave daemons. Hadoop NameNode uses the rack information to distribute replicas across racks (avoiding multiple copies of the same block on the same rack) to ensure fault tolerance in case of rack failure. The other side of this rack awareness replication policy is the increased I/O cost due to the movement of blocks across racks.
- Cluster machines can access the distributed dataset in parallel, thus providing high throughput [26]. For computations, hadoop uses distributed data processing framework called MapReduce, which uses the move code to data principle [27]. Hence, a portion of data is computed on the same node where it resides. MapReduce has two phases: the map phase and the reduce phase. The map phase uses one or more mappers to process the input data, and the reduce phase uses zero or more reducers to process the data output during the map phase.

1. **Map phase**
   a. Split the input data into several data segments.
   b. Generate and assign a separate map task for each data segment.
2. **Distribute the map tasks across the clusters of nodes.**
3. **Run the map tasks in the distributed framework.**
   a. Each map task runs on the disjoint set of input key-value pairs.
   b. Each map task outputs partially consolidated data in output key-value pairs.

c. Output key-value pairs are also called intermediate key-value pairs.

4. Reduce phase

a. The outputs of map tasks (intermediate data set) are sorted and segmented.

b. Segmentation is done so that values associated with the same key belong to the same segment and are sent to the same reducer.

c. Reducers reduce the number of values associated with a particular key.

- A hadoop cluster can have only one JobTracker and several Task-Trackers. JobTracker accepts the client's MapReduce job submission, creates, and coordinates job tasks to the TaskTrakers, finds the location of data from the NameNode, schedules, and monitors the functions on the TaskTracker, and handles the failed TaskTracker tasks [28, 29].

4. EXPERIMENTATION

The experiment is performed by installing the Hadoop cluster on three machines with 11th Gen Intel(R) Core (TM) i5 processor, 8 GB RAM, and SSD. The Hadoop ecosystem is installed with Hbase, Cassandra, MongoDB, and CouchDB as NoSQL databases and MySQL as a relational database on the ubuntu platform on VM VirtualBox. Experimentation involves the execution of a few basic CRUD (Create, Read, Update, Delete) queries along with complex join, filter, and aggregate queries using a single node and the multinode cluster. The task of creating big datasets is automated through PL/SQL scripts. The generated datasets are imported into multiple database systems, and semantically equivalent queries are executed on each. The results are used to gain insights into the performance efficiency of different database systems in different scenarios. MySQL and HBASE experimentations are performed on the employee database, whose schema is shown in following Fig. 1.

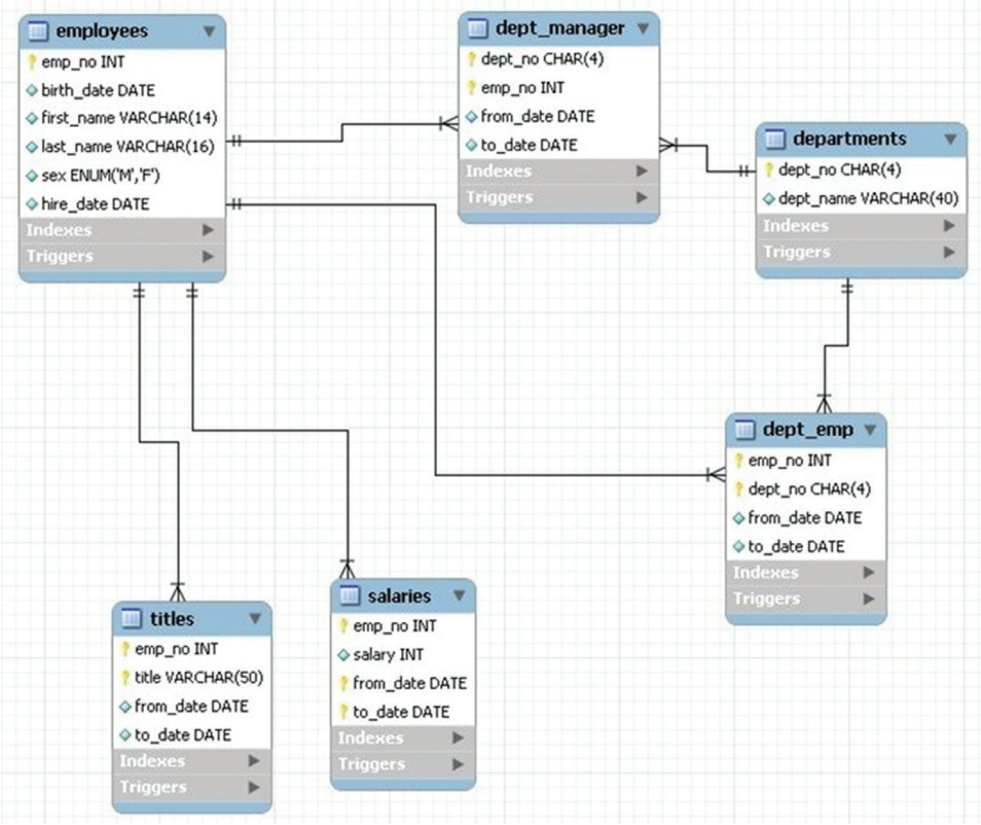

Fig. 1. Employee Dataset schema

A few examples of select and update queries executed on MySQL and HBASE are,

Find the total salary dispensed by each department.

- List the employees who joined after 2019.
- List all employees in terms of increasing or decreasing salaries.
- Increment the salary of each employee by 10 % of the current salary.

MongoDB, Cassandra, and CouchDB-based experimentations are performed on the Movie database (structure derived from https://www.kaggle.com/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows. Data is generated using PL/SQL script, which generates random values (from the predefined set of values extracted from the web-crawled data) having attributes: Title, ReleaseDate, Runtime, Director, Star1, Star2, Star3, and GrossIncome.

A few examples of select, join, and aggregate queries executed on the Movie database are as follows:

- Report the director-wise total income from the short movies (maximum runtime 2 hours).
- Display the year-wise topmost movies in terms of Gross Income.
- Find out the number of movies in which the director is also a leading actor (Star1).
- List all the movies released in the year 2019 in ascending order of their release date.

4.1 EXPERIMENTATION RESULTS

As shown in Tables 2 & 3, although the insertion performance of HBASE (using HIVE) is less than MySQL for the examined datasets, the batches of filter, join, and aggregate queries run several times faster on HBASE multinode cluster in comparison to MySQL. With the increase in the number of nodes in the HBase cluster, the performance difference in further processing the batch of queries is expected to increase substantially with the scaling up of HBASE cluster performance. The HBASE execution time diminishes by approximately half when the cluster size doubles.

Table. 2. Table-wise average Insertion time of MySQL and HBASE (* - Single Node, # - Multinode Cluster)

Database	No of rows	employees	salary	titles	dept_emp
MySQL	100000	1375 ms	1328 ms	1015 ms	3024 ms
HBASE*	100000	40256 ms	40307 ms	40187 ms	40078 ms
MySQL	200000	3284 ms	3313 ms	2671 ms	4684 ms
HBASE*	200000	77461 ms	77465 ms	76554 ms	77123 ms
HBASE#	100000	2 m 51 s	2 m 25 s	2 m 45 s	2 m 55 s

Table 3. MySQL versus HBASE: Average execution time for the filter, join and aggregate queries. (* - Single Node, # - Multinode Cluster)

Database	No of rows	Where Queries (avg)	Join Queries (5000)	Aggregate Queries (5000)
MySQL	100000	5 m 3 s	7 m 35 s	9 m 3 s
HBASE*	100000	12 s	8 m 45 s	10 m 35 s
MySQL	200000	9 m 52 s	18 m 38 s	14 m 4 s
HBASE*	200000	22 s	16 m 40 s	21 m 52 s
HBASE#	100000	1 s	2 m 45 s	3 m 35 s

Table 4. HBASE Performance -- Static vs. Dynamic Cluster (SSN – Static Single Node, DSN – Dynamic Single Node)

Database	No of rows	Where Queries (avg)	Join Queries (5000)	Aggregate Queries (5000)
HBASESSN	100000	12 s	8 m 45 s	10 m 35 s
HBASESSN	200000	22 s	16 m 40 s	21 m 52 s
HBASEDSN	200K–300K	36 s	25m 47s	32m 25s

With HBASE, for the same number of nodes, static cluster performance in terms of scale-up and resource utilization is better than dynamic cluster (shown in Table 4), where the changes in the configuration of the cluster are applied automatically in the online mode. MongoDB accomplishes a higher throughput on dynamic clusters for all three types of queries (shown in Table 5).

Table 5. MongoDB Performance -- Static vs. Dynamic Cluster (SSN – Static Single Node, DSN – Dynamic Single Node, SC – Static Cluster, DC – Dynamic Cluster)

Database	No of rows	Insertion	Where Queries (avg)	Aggregate Queries
MongoDBSSN	100000	13.8 s	0.0250 s	11.84 m
MongoDBSSN	200000	16.05 s	0.0104 s	24.43 m
MongoDBSSN	400000	26.13 s	0.0103 s	49.73 m
MongoDBDSN	400000	28.5 s	0.0137 s	52.06 m
MongoDBSC	400000	13.93 m	13.31 m	1.81 h
MongoDBDC	400000	11.7 m	0.012 s	52.28 m

Table 6. MySQL vs CouchDB

Database	No of rows	Where Queries (avg)	Join Queries (5000)	Aggregate Queries (5000)
MySQL	100000	4.7 m	38.7 m	44.8 m
CouchDB*	100000	15.3 m	7.2 m	13.9 m
MySQL	200000	5.4 m	8.8 m	87.2 m
CouchDB*	200000	15.3 m	8.8 m	12.0 m
MySQL	300000	5.7 m	109.8 m	127.2 m
CouchDB*	300000	15.6 m	9.9 m	12.1 m
MySQL	400000	5.6 m	147.1 m	170.1 m
CouchDB*	400000	15.9 m	11.3 m	11.8 m

The experimental observations (shown in Table 6) show that filter queries take more time in CouchDB whereas join and aggregate queries run faster in CouchDB than in MySQL database. With the inbuilt cache and use of map-reduce, CouchDB read-write performance is good and is suitable for interactive applications (Table 6).

MongoDB with Apache Storm cluster with static/dynamic data is far more time efficient than MySQL, and its performance scales up nicely with the expansion in the number of nodes in the cluster. Because of this, MongoDB has high success in running a large number of queries in a short time. MongoDB's performance for join queries (complex queries in general) is far superior in comparison to its SQL as well as NoSQL counterparts because of its extensive use of subdocuments and embedded lists, thereby avoiding the computations for searching the matched documents in the other collections (results are shown in Table 7). With update-heavy workloads, MongoDB performance dips, but on read-heavy workloads, it's performance remains leading compared to other databases.

MongoDB's scaling performance is better than Cassandra's insertion time and retrieval time (total and average time) for various data sizes on single and multimode clusters are shown in Tables 8 and 9.

Table 7. MySQL vs. MongoDB

Database	No of rows	Insertion	Where Clause	Aggregate Clause	Join Clause
MySQL	100000	1.45 h	2.11 m	31.5 s	4.2 m
MySQL	200000	2.88 h	7.99 m	51.1 s	14.91 m
MySQL	400000	6.01 h	14.78 m	1.95 s	41 m
MongoDB*	400000	303.11 s	0.023 s	1.408 h	@
MongoDB#	400000	292.008 s	1.009 s	2.405 h	@

The number of queries per clause= 10000,
*: MongoDB performance using Apache storm Single Node with static data
#: MongoDB performance using Apache storm Cluster with dynamic data.
@: Data Stored in one collection only with embedded documents and lists of other papers. All related data are in a single collection only. No joins of multiple collections were performed.

For the Cassandra database, in tables 8 and 9, the total and average insertion time and total and average retrieval time for a number of rows (in the 2nd column) are shown (in the 3rd and 4th column [from the left], respectively) for both single node and multinode cluster.

Experimentation shows on importing the Movie dataset and running the batches of queries (read-intensive, write-intensive, and read-write mixed) in HBase, MongoDB, and Cassandra that, mostly HBASE is more efficient in handling write-intensive workloads. In contrast, Cassandra is more efficient while dealing with read-intensive workloads. For read-intensive workloads (above 80% reads), MongoDB gives better performance than all the other databases. HBase performance is better on workloads having mixed read and write requests. On balanced read-write workloads (50 % each), MongoDB shows better scaling behavior when compared to Cassandra (linear). When tried on read-intensive workloads, Cassandra shows significantly high disk I/O compared to CouchDB and MongoDB.

Table 8. Cassandra read-write performance statistics on a single node cluster

Performance statistics on Cassandra			
Retrieval type	Number of Rows	Insertion time total) (avg) ms	Retrieval time (1000 queries) (total)/(avg) ms
static	100000	79152 (0.26384)	72047 (72)
static	200000	164342 (0.8217)	21873 (81)
static	300000	242731 (0.829)	83212 (83)
dynamic	700001-800000	110947 (1.1)	138176 (138)
dynamic	800001-1000000	218140 (1.0)	156368 (156)
dynamic	1000001-1300000	315755 (1.0)	198109 (198)

Table 9. Cassandra read write performance statistics on three node cluster

Cassandra Cluster 3-node cluster (replication factor: 3)			
Retrieval type	**Number of Rows**	**Insertion time (total)/(avg) ms**	**Retrieval time (1000 queries) (total)/(avg) ms**
static	1-100000	159034ms/1.590ms	54591ms/54.591 ms
static	100001-300000	229374ms/1.147ms	54903ms/54.903ms
static	300001-600000	346375ms/1.1546ms	66778ms/66ms
dynamic	600001-700000	154942ms/1.549ms	100469ms/100.469ms
dynamic	700001-900000	271857ms/1.359ms	77959ms/77.959ms
dynamic	900001-1200000	470318ms/1.568ms	73065ms/73.065ms

In general, using the traditional optimization techniques in the distributed databases, and deploying the databases on scalable distributed frameworks like the cloud, makes the application more efficient in query execution as well as in maintaining system performance in the presence of a fluctuating number of application users which eventually leads to fluctuating sizes of the query workloads. It is observed that within the available scope of further decomposition of the workload and workload queries, increasing the number of cores on a single node or the number of nodes in the cluster directly impacts the throughput and speed of query execution. However, the relative performances of different types of databases vary with the change in the read-write latencies, which is somewhat characterized by patterns of workload queries (read-intensive, write-intensive, read-write-mixed).

Experimentation shows that HBASE is efficient in handling write-intensive workloads and read-write mixed workloads, whereas Cassandra, MongoDB is more efficient when dealing with read-intensive workloads. From the performance behavior, it can be safely concluded that MongoDB is best suitable for read-intensive workloads. In contrast, HBASE is a better choice for writing dominant and balanced read-write workloads. With the inbuilt cache supporting the map-reduce framework, CouchDB read-write performance is suitable for interactive applications.

5. CONCLUSION

This paper contains a comparison of SQL databases and few NoSQL databases concerning their architectures, underlying working principles, advantages and disadvantages, and suitability in different application domains. Also, it demonstrates the characteristics of popular NoSQL databases using experimental results. NoSQL databases are designed to store and analyze big data generated by sources like social media, e-commerce websites, sensors, etc. They are instrumental in several IOT-based smart systems (e.g., Health Monitoring Systems, Traffic Monitoring Systems, etc.). To handle large volumes, velocity, and variety of data and to cater to advanced requirements like scalability, availability, and fault tolerance, databases need to have the capacity to compute, store, access, and analyze data in a distributed fashion. As per our experimental results, it is evident that NoSQL databases fit these demands. Compared to relational databases, they are more suitable for dealing with big data tasks on scalable, elastic and fault-tolerant platforms like the cloud.

6. REFERENCES

1. E. Lotfy, A. I Saleh, H. A. El-Ghareeb, H. A. Ali, "A Middle Layer Solution to Support ACID properties for NoSQL Databases", Journal of King Saud University-Computer and Information Sciences, Vol. 28, No. 1, 2016, pp. 133-145.
2. R. Cattell, Scalable, "SQL and NoSQL data stores", ACM Sigmod Record, Vol. 39, No. 4, 2011, pp. 12-27.
3. R. Hecht, S. Jablonski, "NoSQL evaluation: A use case-oriented survey", Proceedings of the International Conference on Cloud and Service Computing, Hong Kong, China, 12-14 December 2011, pp. 336-341. 10.1109/CSC.2011.6138544.
4. C. Bazar, C. S. Iosif, "The transition from Rdbms to NOSQL. A Comparative Analysis of three Popular Non-Relational Solutions: Cassandra, Mongodb and Couchbase", Database Systems Journal, Vol. 5, No. 2, 2014, pp. 49-59.
5. J. Sivakumaran, S. Z. Ali, "RDBMS Current Challenges and Opportunities with NoSQL to NewSQL", Proceedings of the 3rd Middle East College Student Conference, Muscat, Sultanate of Oman, 31 December 2017.
6. M. A Mohamed, O. G. Altrafi, M. O. Ismail, "Relational vs. NoSQL Databases: A Survey", International Journal of Computer and Information Technology, Vol. 3, No. 3, 2014, pp. 598-601.
7. J. Guia, V. G. Soares, J. Bernardino, "Graph Databases: Neo4j Analysis", Proceedings of the 19th International Conference on Enterprise Information Systems, Porto, Portugal, 26-29 April 2017, pp. 351-356.

8. M. Houcine, G. Belalem, K. Bouamrane, "Comparative study between the MySQL relational database and the MongoDB NoSQL database", International Journal of Software Science and Computational Intelligence, Vol. 13, No. 3, 2021, pp. 38-63.

9. M. Stonebraker, "SQL databases v. NoSQL databases", Communications of the ACM, Vol. 53, No. 4, 2010, pp. 10-11.

10. D. G. Chandra, "BASE analysis of NoSQL database", Future Generation Computer Systems, Vol. 52, 2015, pp. 13-21.

11. D. Glushkova, P. Jovanovic, A. Abelló, "Mapreduce performance model for Hadoop 2. x.", Information Systems, Vol. 79, 2019, pp. 32-43.

12. C. Candel, D. S. Ruiz, J. J. Garcıa-Molina, "A Unified Metamodel for NoSQL and Relational Databases", Information Systems, Vol. 104, 2021, p. 101898.

13. R. Sellami, B. Defude, "Complex queries optimization and evaluation over relational and NoSQL data stores in cloud environments", IEEE Transactions on Big Data, Vol. 4, No. 2, 2017, pp. 217-230.

14. M. A. Elsabagh, "NO SQL Database: Graph database", Egyptian Journal of Artificial Intelligence, Vol 1, No. 1, 2022, pp. 1-7.

15. R. Gyorodi, G. Gyorodi, A. Pecherle, "A comparative study: MongoDB vs. MySQL", Proceedings of the 13th International Conference on Engineering of Modern Electric Systems, Oradea, Romania, 11-12 June 2016, pp. 1-6.

16. K. Orend, "Analysis and Classification of NoSQL databases and Evaluation of their Ability to Replace an Object-Relational Persistence Layer", Architecture, Vol. 1, 2010, pp. 1-100.

17. Apache Cassandra (TM) 4.0, The Documentation, https://cassandra.apache.org/doc/latest/ (accessed: 2021)

18. Chebotko, A. Kashlev, S. Lu, "A Big Data Modeling Methodology for Apache Cassandra", Proceedings of the IEEE International Congress on Big Data, New York, NY, USA, 27 June - 2 July 2015, pp. 238-245.

19. N. D. Bhardwaj, "Comparative Study of Couchdb and Mongodb–NoSQL Document-Oriented Databases", International Journal of Computer Applications, Vol. 136, No. 3, 2016, pp. 24-26.

20. R. Zafar, E. Yafi, M. F. Zuhairi, H. Dao, "Big Data: The NoSQL and RDBMS review", Proceedings of the International Conference on Information and Communication Technology, Bandung, Indonesia, 25-27 May 2016, pp. 120-126.

21. Apache HBase, The Documentation, https://hbase.apache.org/ (accessed: 2021)

22. A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, U. Saxena, "NoSQL databases: Critical analysis and comparison", Proceedings of the International conference on computing and communication technologies for smart nation, New York, NY, USA, 12-14 October 2017, pp. 293-299.

23. A. Krechowicz, S. Deniziak, G. Łukawski, "Highly scalable distributed architecture for NoSQL datastore supporting strong consistency", IEEE Access, Vol. 9, 2021, pp. 69027-69043.

24. J. K. Chen, W. Z. Lee, "An Introduction of NoSQL Databases based on their categories and application industries", Algorithms, Vol. 12, No. 5, 2019, pp. 106-123

25. J. Anuradha, "A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology", Procedia computer science, Vol. 48, No. 1, 2015, pp. 319-324.

26. J. Zakir, T. Seymour, K Berg, "Big Data Analytics", Issues in Information Systems, Vol. 16, No. 2, 2015.

27. J. Pokorny, "NoSQL databases: a step to database scalability in web environment", Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services, New York, NY, USA, 5-7 December 2011, pp. 278-283.

28. J. Dittrich, J. A. Quiane-Ruiz, "Efficient Big Data Processing in Hadoop MapReduce", Proceedings of the VLDB Endowment, Vol. 5, No. 12, 2012, pp. 2014-2015.

29. B. Jose, S. Abraham, "Performance analysis of NoSQL and relational databases with MongoDB and MySQL", Materials today: Proceedings, Vol. 24, No. 3, 2020, pp. 2036-2043.