View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by St. Cloud State University

St. Cloud State University
theRepository at St. Cloud State

Culminating Projects in Information Assurance

Department of Information Systems

5-2016

A Critical Comparison of NOSQL Databases in the Context of Acid and Base

Deepak GC
St Cloud State University, gcde0401@stcloudstate.edu

Follow this and additional works at: https://repository.stcloudstate.edu/msia_etds

Recommended Citation
GC, Deepak, "A Critical Comparison of NOSQL Databases in the Context of Acid and Base" (2016). *Culminating Projects in Information Assurance*. 8.
https://repository.stcloudstate.edu/msia_etds/8

This Starred Paper is brought to you for free and open access by the Department of Information Systems at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of theRepository at St. Cloud State. For more information, please contact rswexelbaum@stcloudstate.edu.

A Critical Comparison of NOSQL Databases in the Context of ACID and BASE

by

Deepak GC

A Starred Paper

Submitted to the Graduate Faculty

of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Information Assurance

St. Cloud, Minnesota

April, 2016

Starred Paper Committee:

Dr. Jim Q Chen, Chairman

Dr. Susantha Herath

Dr. Balasubramanian Kasi

Acknowledgements

I would like to thank all of my professors for helping me through college and research. A very special thank you to all the starred paper committee members. Lastly, I would also like to acknowledge this to my parents for everything they have done for me.

Abstract

This starred paper will discuss two major types of databases – Relational and NOSQL – and analyze the different models used by these databases. In particular, it will focus on the choice of the ACID or BASE model to be more appropriate for the NOSQL databases. NOSQL databases use the BASE model because they do not usually comply with ACID model, something used by relational databases. However, some NOSQL databases adopt additional approaches and techniques to make the database comply with ACID model. In this light, this paper will explore some of these approaches and explain why NOSQL databases cannot simply follow the ACID model. What are the reasons behind the extensive use of the BASE model? What are some of the advantages and disadvantages of not using ACID? Particular attention will be paid to analyze if one model is better or superior to the other. These questions will be answered by reviewing existing research conducted on some of the NOSQL databases such as Cassandra, DynamoDB, MongoDB and Neo4j.

Table of Contents

List of Tables

List of Figures

Figure Page

Chapter 1: Introduction

Databases are important components of Information Systems and are concurrent to the existence of computer technology. Storing and retrieving of data in a meaningful manner requires the use of appropriate databases. This paper discusses two major types of databases – Relational and NOSQL (Not Only SQL) – and analyzes the different models used by these databases. SQL stands for Structured Query Language often referring relational database systems. Relational databases have always been the staple when it comes to databases. They are popular because of their consistency and functionality. However, in the present context of constantly changing data landscape, these databases have limitations when it comes to non-structured and vast amount of data. This has given rise to a new type of database known as NOSQL.

The focus of discussion in this paper is the use of appropriate model by each of these databases. Relational databases use the ACID (Atomicity, Consistency, Isolation and Durability) model for transaction processing because of strong reliability and consistency. The same cannot be said about NOSQL databases because of different priorities and emphasis. In fact, NOSQL databases use the BASE (Basic Availability, Soft state, Eventual consistency) model. Over the years, there have been many researches conducted in this field to justify the use of BASE in NOSQL databases and that of ACID in relational databases. But, there is no unanimous answer to the question. Therefore, this paper presents a survey of existing literature in an effort the to explain this complex issue.

Problem Statement

NOSQL is a new concept in the database world where relational databases have been around for decades, which makes it important to know how the model used by NOSQL compares to ACID properties. Also, the properties of BASE implemented by the four NOSQL databases should be evaluated in the context of ACID. Since ACID is a benchmark model for data integrity, comparing NOSQL against ACID properties will give good indication of NOSQL data integrity.

Nature and Significance of the Problem

The problem stems from the fact that NOSQL follows a different approach that is yet to take off unlike ACID, which has been known to provide consistency and reliability to transactions in relational databases. On the other hand, RDBMS (Relational Database Management System) has been around for a while whereas NOSQL is comparably, a new phenomenon. This study will make light into the advantages and disadvantages of using BASE and also point out any data integrity in following it.

Objective of the Study

The objective of this study is to highlight what BASE has to offer and compare that with ACID. The study will enlist the positive and negative effects of this approach in NOSQL databases. Any negative effects found will also be evaluated in terms of data integrity of the NOSQL database as a whole. The purpose is to evaluate if BASE gives comparable data integrity as ACID.

Study Questions

How does the BASE model used in NOSQL databases compare to the ACID model? How do these differences affect consistency, performance or scalability among others? Do these differences, if any, compromise data integrity more than ACID would?

Limitations of the Research

This research is strictly restricted to existing studies that range from scholarly articles to research white papers to blogs and other credible sources. The research's aim is not to conduct any specific lab tests or experiments.

Definition of Terms

Different terms used throughout this research are described here. This will give a clear understanding of what the terms mean and clarify any misunderstanding going forward. Any acronyms used will also be described here.

RDBMS: RDBMS stands for Relational Database Management Systems. RDBMS in this research stands for a model database management system. Relational databases follow relational model invented by Edgar F. Codd that identify three components in a data model that are data structure, operators and integrity rules [1]. Relations, attributes, tuples and keys give structure in RDBMS. Operations such as select, insert, update and delete are common operators in relational databases. Keys (primary) among other different referential integrity rules primarily define integrity rules.

ACID: ACID stands for Atomicity, Consistency, Isolation and Durability. ACID was an idea developed in the 1970s by Jim Gray and is a property required to run RDBMS without any errors [2]. ACID is a measure of determining if a database is good where transaction in a database run as a single logical operation [3]. Atomicity means that transaction(s) either succeed or fail as a unit where a failed transaction will result in the database being left in the state before that transaction started. Consistency means that the database conforms to a stable state before and after a transaction. Isolation means that an incomplete transaction is not visible to other transaction(s), a transaction has to wait for other to finish in order for it to start. Durability means that effects of a completed transaction are permanent in the event of a later system failure.

NOSQL: NOSQL systems are mostly non-relational database systems that are distributed and are understood as Not Only SQL. NOSQL databases are known to provide easier scalability, storage flexibility, and greater data manipulation and performance improvement. There are various types of NOSQL database systems among which Key-value stores; Wide-column stores, Graph databases and Document stores are identified most commonly [1]. MongoDB, Cassandra, DynamoDB and CouchDB, Neo4j, Riak are the more popular NOSQL databases used commonly in today's environment.

CAP theorem: CAP theorem was proposed by Dr. Eric Brewer in 2000 AD which stated that three important components namely Consistency, Availability and Partition-tolerance were needed for the successful implementation of a networked shared-data system [2]. The theorem further states that two of the three components have to be at least satisfied for CAP theorem compatibility. Consistency means having a single copy of up to date data. Availability means that the data is readily available for updates. Partition-tolerance means that the data in not susceptible to network-partitions needed for stability.

BASE: BASE is an acronym for basically available, soft state, eventually consistent. BASE is regarded as an alternative to ACID in the database community in very large systems that are unable to follow ACID. BASE is derived from CAP theorem.

Summary

This chapter introduced readers to the database in information systems and different types of databases. A basic introduction to RDBMS and NOSQL databases was also provided. This chapter also defined other terms that will be used throughout this paper providing readers a clear understanding of what those terms mean. This chapter provided an overview of the research. The next chapter will focus on relevant literature and scholarly articles written in this area.

Chapter 2: Background and Review of Literature

There have been many researches done in the ACID, BASE and CAP (Consistency, Availability and Partition Tolerance) area where their differences, advantages and disadvantages have been identified. Some researchers have also identified various advantages and disadvantages of these models. This chapter will focus on those literature and scholarly articles written in this area. This will provide the reader and this research a base to where this research will continue. Review of literature in this section will also bring light into the suggested research.

Background Related to the Problem

RDBMS and NOSQL are two different database systems. RDBMS provide certain functionalities that give them greater stability, performance and consistency. These database systems are extremely successful and have become synonymous to data access. RDBMS came into existence out of research at IBM and the University of California at Berkeley in the 1970s [4]. Relational databases solved many problems like providing logical view of data and also provided a specific language to fetch data. However, these database systems could not accommodate non-structured data and specifically big data. Big data is defined as a collection of data sets that is large and complex that becomes difficult to process using traditional data processing applications. The rise of web applications produced more complex data and the size of data grew tremendously which is when NOSQL was born.

ACID compliance in RDBMS is of paramount importance where consistency is required which ensures that transactions are completed in a single instance before changes are committed to the master database [5]. ACID has a long history of keeping data safe and consistent in its long existence. BASE on the other hand focuses more on high availability, thus differentiating itself from ACID. Even though CAP theorem, also known as Brewer's theorem desires to have two of three properties: consistency, availability and partition tolerance; it's ultimate goal is to maximize consistency and availability for the specific application [6].

These different approaches are better fit for their particular type of databases; ACID is better suited for RDBMS and BASE is better suited for NOSQL databases. There are no one size fits all. This is why it is important that these different approaches be compared to find out if one is better than the other, to know if one approach provides better data integrity than the other. Database servers are of upmost important when it comes to data integrity. Information Week's State of the database technology survey conducted in November 2012 found out that 14% of all breaches that year were because of compromised database servers [7]. As NOSQL databases go more mainstream and unstructured data grow profoundly, data integrity in these technologies must be evaluated. Since, NOSQL databases mostly use BASE as their model, it is important that we compare and contrast how BASE fares against ACID.

Literature Related to the Problem

It is important to understand how ACID works in a database environment. An example from Service Architecture states an example in such a manner, "Imagine more than one person trying to buy the same size and color of a sweater at the same time -- a regular occurrence. The ACID properties make it possible for the merchant to keep these sweater purchasing transactions from overlapping each other, hence saving the merchant from erroneous inventory and account balances" [8]. ACID properties make it possible for a safe sharing of data by avoiding inaccuracies in data. ACID properties also allow for greater flexibility and efficiency.

According to an article by Mihalcea [9], which explains how ACID makes transactions safe and secure and can be described as successful by having all operations succeed or else the transaction fails. This is also shown below:

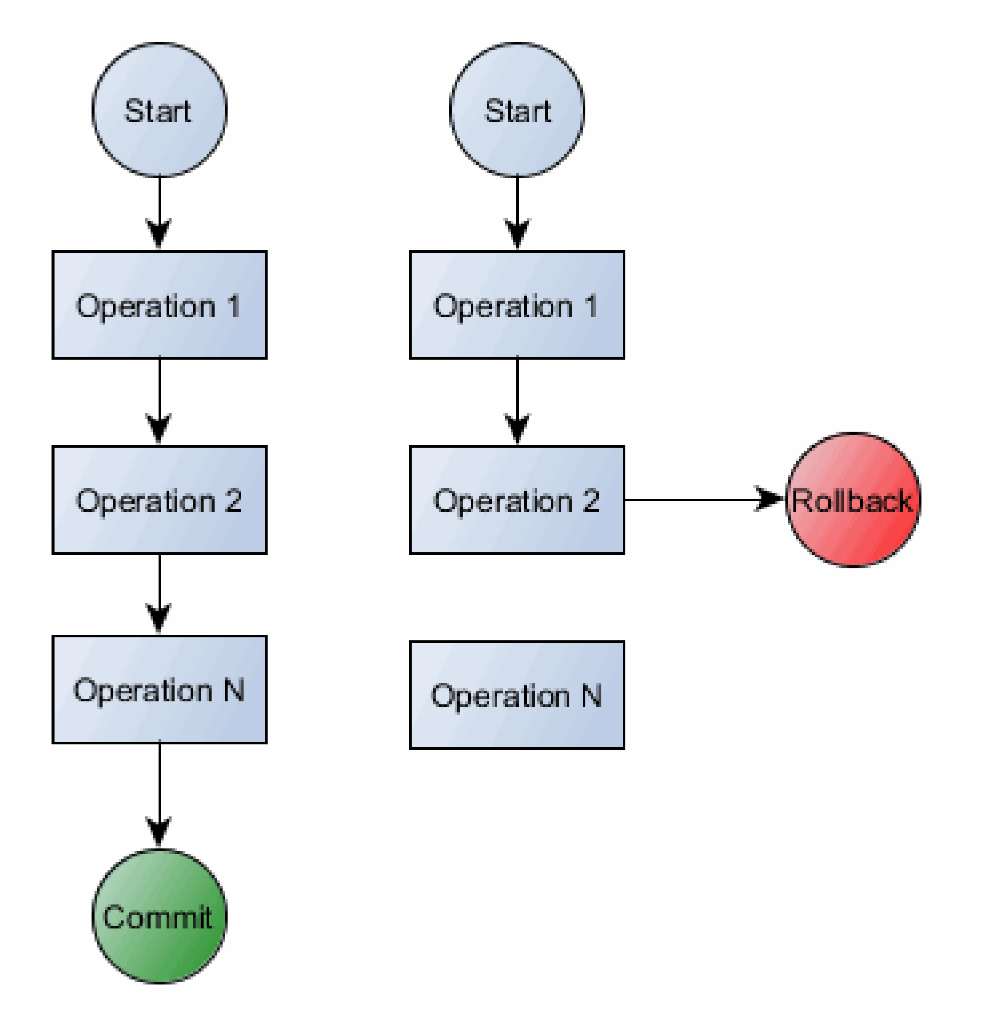


Figure 1

Transaction processing in ACID [9]

As the figure above shows, transactions are a collection of read-write operations that either succeed or fail as a whole leaving behind no incomplete transaction. Another thing is that a transaction cannot leave a system in an inconsistent state regardless of concurrent transactions. This provides a safety blanket to the database in a sense that any unwarranted event such as power failure, error or crash won't allow to be in a state with partial change. The database stays in consistent state when the transaction begins and end. Data integrity is also achieved in ACID transactions because of how relational database transactions are processed. The same cannot be said about using BASE, where database consistency will be in a state of flux [10]. However, this makes BASE highly scalable compared to that of ACID. Consistency in BASE is not reached right away after finishing the transaction, but rather in some time during the operation thus being eventually consistent [11].

One of the biggest challenges with RDBMS is their scalability, where many NOSQL databases are better suited. An article by Kopp [12] differentiates RDBMS and NOSQL scalability where horizontal distribution of load and data along with their table-based nature limits RDBMS. Figure below shows exactly how scalability is difficult in RDBMS. The example here shows that in NOSQL, Client1 could read from node 1 and write 20 entities to node 2 without having to check consistency between the two nodes, whereas in RDBMS, node 1 and node 2 need to have consistent data in order to perform either one of the operations because of ACID compliance [12].

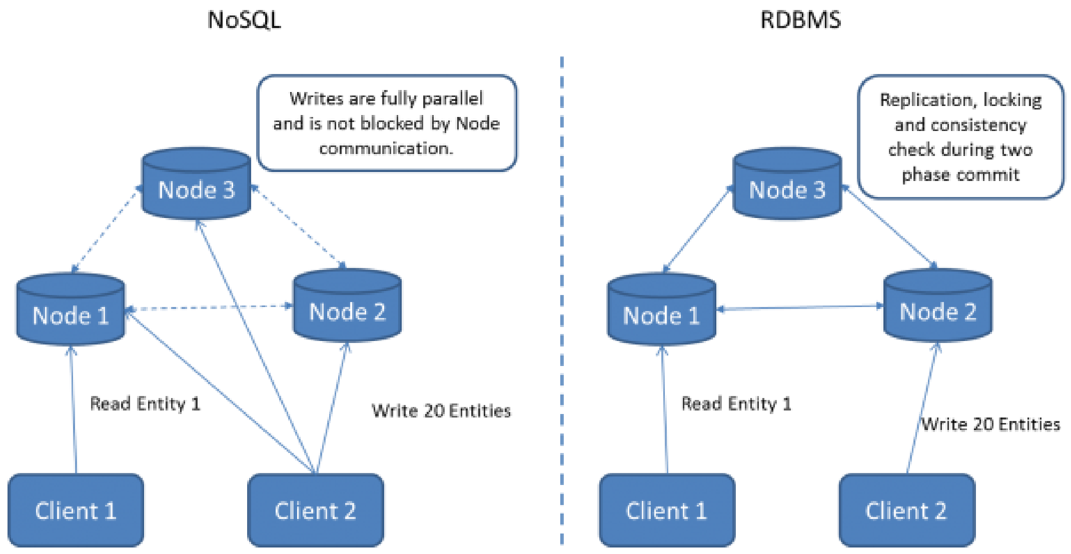


Figure 2

Read-write operations in NOSQL and RDBMS [12].

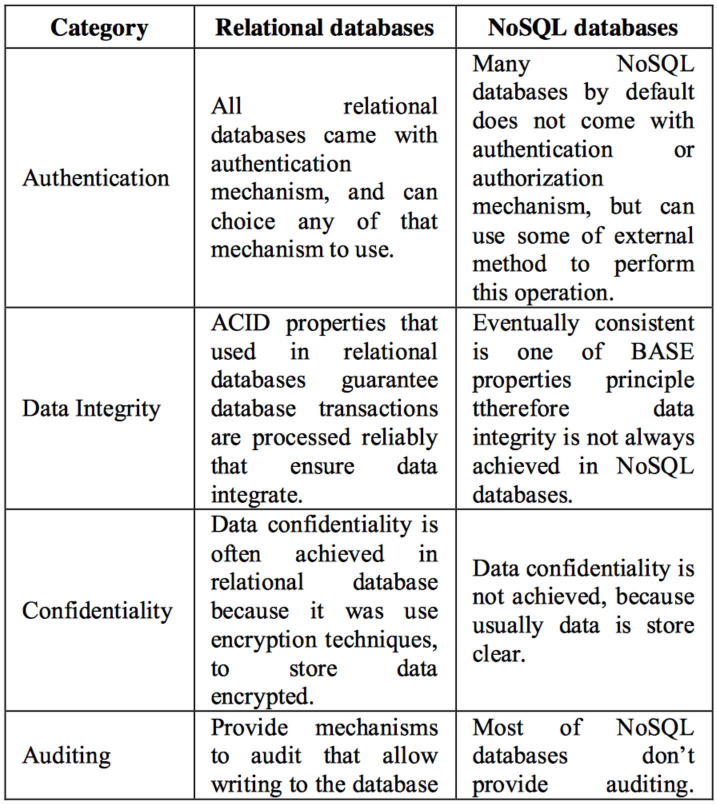
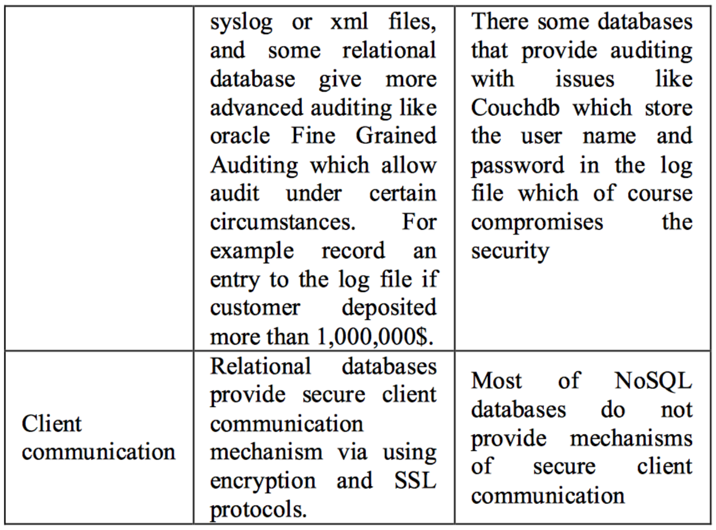
There are many similar differences between NOSQL and RDBMS where because of the model that they use; the transactions are processed differently giving one edge over the other in certain area. This research’s focus is similar to all the literatures that have been reviewed thus far.

Literature Related to the Methodology

Englund et al. [13] wrote a research paper about ‘Investigating NOSQL from a SQL Perspective’ where they compare and contrast NOSQL and SQL. In the research, they go into detail of how these different databases are composed of, how they operate and also list and describe various NOSQL databases. They also do performance testing on some of the NOSQL databases. The research is a good example of methodology that will be used in this research. This research will do similar comparison of ACID and BASE and also point out some data integrity features that are present or absent in some NOSQL databases that follow BASE approach.

Another research done by Hammes et al. [5] compares SQL and NOSQL databases in the cloud. This research paper examines design, performance and execution between RDBMS and NOSQL database systems. They do this in the same cloud environment using structured and unstructured data. The research by Hammes et al. [5] similar to the research mentioned in the earlier paragraph is significant to this research because of the methodology followed in these approaches. This research paper will do similar comparison between ACID and BASE model using literatures and articles.

Mapanga et al. [2] in their analysis of NOSQL databases have also done similar comparisons of ACID and BASE along with other comparisons. Other comparisons include query languages and NOSQL database categories. This research will use similar methodology to compare ACID and BASE by pointing out advantages and disadvantages.

Category	Relational databases	NoSQL databases
Authentication	All relational databases came with authentication mechanism, and can choice any of that mechanism to use.	Many NoSQL databases by default does not come with authentication or authorization mechanism, but can use some of external method to perform this operation.
Data Integrity	ACID properties that used in relational databases guarantee database transactions are processed reliably that ensure data integrate.	Eventually consistent is one of BASE properties principle ttherefore data integrity is not always achieved in NoSQL databases.
Confidentiality	Data confidentiality is often achieved in relational database because it was use encryption techniques, to store data encrypted.	Data confidentiality is not achieved, because usually data is store clear.
Auditing	Provide mechanisms to audit that allow writing to the database syslog or xml files, and some relational database give more advanced auditing like oracle Fine Grained Auditing which allow audit under certain circumstances. For example record an entry to the log file if customer deposited more than 1,000,000$.	Most of NoSQL databases don't provide auditing. There some databases that provide auditing with issues like Couchdb which store the user name and password in the log file which of course compromises the security
Client communication	Relational databases provide secure client communication mechanism via using encryption and SSL protocols.	Most of NoSQL databases do not provide mechanisms of secure client communication

Figure 3

Comparison of different categories in NOSQL and RDBMS [11]

"Relational vs. NoSQL Databases: A Survey" by Altrafi et al. [11] is another scholarly article that delves into relational and NOSQL comparison. They do a review of NOSQL movement and identify issues that are present in these databases. An important part of this research is to data integrity issues that could be present in NOSQL databases because of using BASE either directly or indirectly.

Summary

This chapter described the background related to the problem, which showed how ACID and BASE have different priorities when it comes to their properties. This chapter also reviewed some important literature relating to the problem and methodology. Several literatures showcased current researches done in this area. Next chapter will focus specifically on the methodology of this research where the framework of the study will be explained.

Chapter 3: Methodology

This part of the research will discuss how the relevant information will be collected and how the research will be conducted. There are no empirical data collected in this study. Evidences will mostly be collected in the form of literature review.

The research specifically chose four database types which are Cassandra, MongoDB, DynamoDB and Neo4j. There is a reason for choosing these databases. Cassandra is the most popular column orientated database type that is available in the market. Cassandra is widely used, open source and can be run on multiple platforms which makes it a column orientated database of choice. Similarly, MongoDB is a highly used document orientated database that implements document features. It is open source whilst also provides extended support on enterprise editions making it a document orientated database of choice. DynamoDB on the other hand a strictly pay-only service provided by Amazon Web Services and is also used widely in the industry when it comes to key value orientated database. It is an interesting database choice as it is sold as a service. The last database type, Neo4j is a graph orientated database which is one of the major players in the Graph database arena. Neo4j is the graph orientated database of our choice as it is known to be ACID compliant and popular.

Cassandra, DynamoDB, MongoDB and Neo4j will each be reviewed, analyzed and dissected to find out what each is composed of, their configuration and their architecture. Any query language used by the four NOSQL databases will also be discussed. Before the four specific database systems are discussed, it is important that types of major NOSQL databases that are around be discussed.

Further, ACID/BASE properties of each are discussed and compared. Properties of ACID and BASE will be explored to see how each fit. Each NOSQL database will be dissected and seen how they compare to ACID properties. This is something that is different from any of the research studies that have been done before.

Similar to research done by Altrafi et.al. [11], this research tries to understand NOSQL databases and also conduct ACID/BASE comparisons following similar footsteps as to previous researchers described in the Literature Related to Methodology section of Chapter II.

The parameters that are discussed in this chapter are as follows:

- Types of NOSQL databases.
  - Cassandra
  - MongoDB
  - DynamoDB
  - Neo4j
- One NOSQL databases from each type.
  - Introduction
  - Configuration/Architecture
  - Query language
- ACID analysis of NOSQL databases.
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- ACID comparison between NOSQL databases.
- BASE comparison between NOSQL databases.

Process

The literature review in this study encompasses various scholarly articles, white papers, blogs and various web sources. The scholarly articles were obtained from searches conducted on the St. Cloud State University library website. The research uses journal articles from International Journal of Modern Communication Technologies & Research, Communications of the ACM, Computer and International Journal of Computer and Information Technology among others. Various sources also include research papers presented to institutions. Since, not very many research has not been carried out in this area of research, the research also relies on White papers and trade journals published by industry experts. About 50% of the research contains study of journal articles and research papers. About 30% includes white papers and trade journals. The rest involves other web sources such as technology blogs and informative websites.

This research paper aims to comprehensively describe four major NOSQL databases in use today and attempts to incorporate as much information it could find relating to architecture, configuration, query language and ACID/BASE properties. The research was limited to the scope of Starred paper and refrained from being excessively broad. Assessing the literature review conducted by this research, it becomes clear that comparison of NOSQL databases are adequate to what the research aimed to accomplish, however, more scholarly journals could be cited as they become available. Future researchers should be able to use this research as a stepping stone in conducting a comprehensive research on just one major NOSQL database type. As research continues to grow in the NOSQL landscape because of their growing popularity, more comprehensive research will continue in this area.

The research did not conduct any lab experiments or hands on implementation as that would require time and resources outside of the scope of this Starred paper. A more extensive research such as a Thesis could perform lab experiment or hands on implementation of either one or more of the databases types discussed in this paper.

Summary

This chapter described the methodology that will be used to conduct this research. The next chapter will focus on the main portion of the study where researched materials are presented.

Chapter 4: Analysis of Results

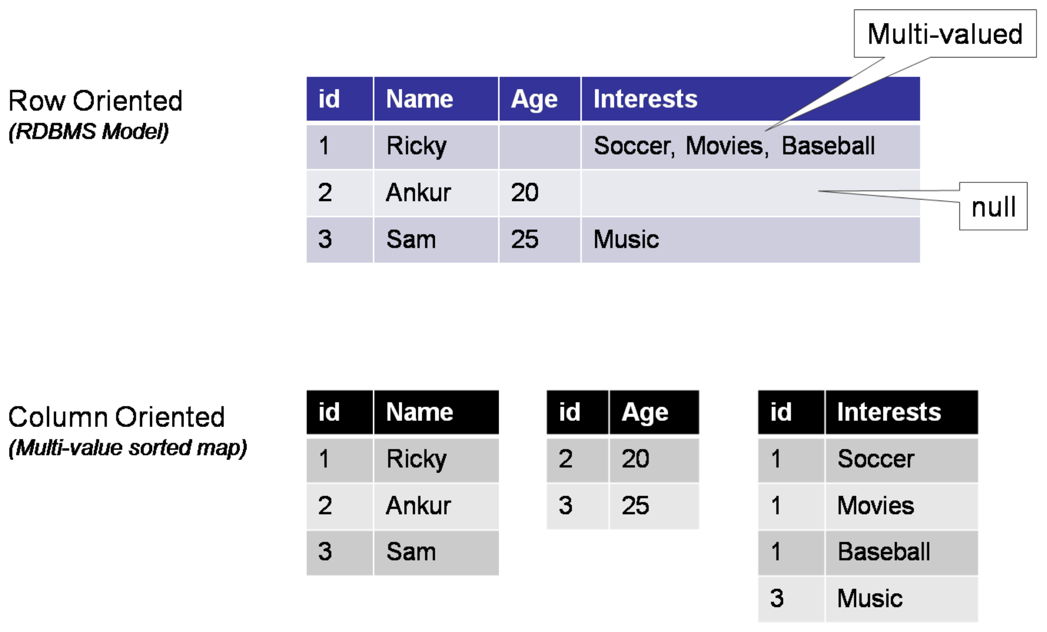
There are four major NOSQL database types; however, there are many that overlap, so drawing a line to distinct each other in some cases could be problematic. Some NOSQL databases combine two or more types of NOSQL databases to form one.

NOSQL Database types

- Column Orientated
- Document Orientated
- Key-value Orientated
- Graph Orientated

Column Orientated

Column orientated databases use a multi-dimensional, sparse, distributed map to store data. This type of databases is also known as column family stores or wide column stores. A record could be in one column or multiple columns and also columns can be nested inside other columns known as super columns. Columns can be grouped together in one column family or multiple column families. Data retrieval happens by using primary key per column family. Contrary to relational databases where a particular data is stored in rows of a table, column orientated databases store data in column(s). On the surface, column orientated databases might appear to look like a relational database, however they are different as these don't have any pre-structured table to work with the data. This property of column orientated database makes retrieving of large amounts of a particular attribute faster. Column orientated databases are in essence a two-dimensional array where each key, a record/row has one or more key/value pairs attached to it [14]. This allows for the management for large unstructured data. Cassandra, Google's Big Table and HBase are some of the more popular wide column database systems.


Row Oriented
(RDBMS Model)

id	Name	Age	Interests
1	Ricky		Soccer, Movies, Baseball
2	Ankur	20	
3	Sam	25	Music

Multi-valued
null

Column Oriented
(Multi-value sorted map)

id	Name
1	Ricky
2	Ankur
3	Sam

id	Age
2	20
3	25

id	Interests
1	Soccer
1	Movies
1	Baseball
3	Music

Figure 4

Column orientated compared to Row orientated database [15]

Document Orientated

Document databases originated from the need to develop a database system that did not rely on schemas. Relational database was introduced to query data using SQL (Structured Query Language) that rely on schemas where objects are considered as sets of relationships. Because of the emergence in cloud computing, a need to store unstructured data and a need for agile development methods, new databases have emerged among which document databases are one of them. In document orientated databases, each record and its associated data is treated as a document also known as semi-structured data. In a document database, everything related to a database object is encapsulated together giving them agility and less dependency. Some of the formats that the documents encapsulate and encode data are JSON (JavaScript Object Notation), XML (Extensible Markup Language) and BSON (Binary JSON). Document orientated databases work in a similar way to column-orientated database but differ in providing deeper nesting and complex structures that is document within document and so on [14]. Some of the popular document orientated databases are MongoDB, CouchDB and Couchbase.

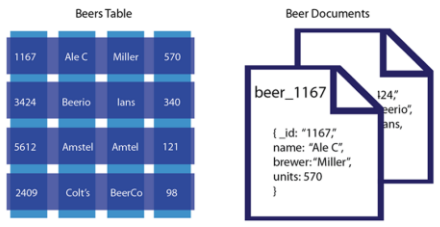


Figure 5

Relational database compared to a document database [16]

Key-value Orientated

Key-value orientated databases are the simplest and backbone implementation of NOSQL database where keys are matched to values similar to dictionary or hash. In computing terminology, dictionaries contain a collection of objects or records that can have many fields that contain data. There is neither relation nor structure in key-value orientated databases. Key-value databases store pair of keys and values, which can be used to retrieve value when a key is known. Key-value stores work differently than relational databases as relational databases use pre-defined data structure as a series of tables containing fields with well-defined data types. On the other hand, key-value stores treat data as a single non-transparent collection that may have different fields for every record. Scalability is one of the big traits of key-value stores as they require less or no redesign and could be fast in most cases [17]. Some of the popular Key-value databases are DynamoDB, Riak and Redis.

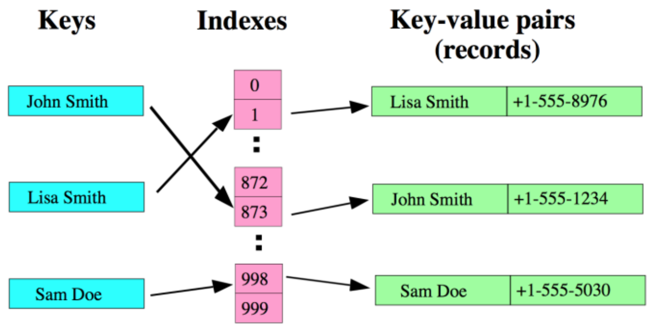


Figure 6

Key-value orientated database [18]

Graph Orientated

Graph databases are totally different from the three previous NOSQL database types. Graph orientated databases are databases that rely on explicit graph structure where nodes and edges connect to each other through relations in a tree like structure. Each node knows it adjacent nodes and there is an index for searches. Nodes store data about each entity in the database, relationships describe relationship between the nodes and property is just the opposite node of the relationship [19]. Some of the popular graph orientated databases are OrientDB, MarkLogic and Neo4j.

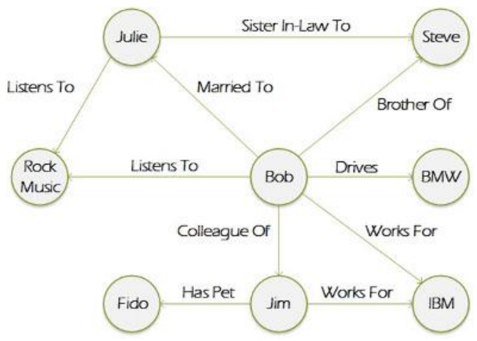


Figure 7

Graph orientated database [20]

Cassandra

Cassandra is one of the first and widely used NOSQL systems. It is an open source database management system supported by Apache Software Foundation, used to manage large structured data. It is a distributed system that relies heavily on availability and having no single point of failure. Apache Cassandra was born at Facebook, Inc. and built on Amazon's Dynamo and Google's Big Table. Cassandra is a row store, a hybrid between column-orientated and key value database system with adjustable consistency.

In the beginning, Google and Amazon realized that relational databases were not sufficient in processing data that were amassed because of increasing user bases of systems, cloud computing, mobile devices and increasing online presence and large systems. Individually, they tried to figure out different ways to tackle problems of scale, which is when they each developed Big Table by Google and Dynamo by Amazon. These challenges could only be overcome by relaxing the guarantees provided by relational data model to achieve scalability. Keeping in line with Brewer's CAP theorem, these systems trade off consistency, and availability and partition tolerance in lieu of scalability. Cassandra systems are known to provide high availability with no single point of failure and eventual consistency thereby abiding to BASE properties.

Architecture of Cassandra

Cassandra is designed to handle big data spreading across multiple nodes and having no single point of failure. A single point of failure means that system and hardware failures are bound to happen and will happen. Since failure cannot happen, availability is of upmost importance. Cassandra addresses this problem by deploying peer-to-peer nodes in homogenous nodes where data is distributed among all nodes in the cluster [21]. Information are exchanged between the nodes frequently where commit log is written sequentially on each node to confirm data durability. Data is indexed into memetable, an in-memory structure. When this memory structure is full, it is then written to disk in an SSTable data file, these writes are then automatically partitioned and distributed throughout the cluster. SSTable stands for Sorted String Table. SSTables are periodically consolidated to discard obsolete data. Cassandra allows an authorized user to connect to any node in any cluster and query using CQL (Cassandra Query Language), similar to SQL syntax. A cluster consists of one keyspace per application. Client read-write requests can be sent to any node in the cluster where the node client connects to act as a coordinator. Coordinator is a proxy between the client and the node(s) that own the data.

Cassandra consists of some key-structures that are listed as follows:

Node: Basic foundation of Cassandra where data is stored.

Datacenter: A datacenter can be physical or virtual. This is a collection of related nodes.

Cluster: A cluster consists of one or more datacenters.

Commit log: Data is first written to commit log and then to SSTable.

Table: A table is a collection of columns that are ordered. Row serves columns and have primary key, where the first part of the key is the column name.

SSTable: Sorted String Table is maintained for each Cassandra table and can only be appended. SSTables are unchangeable data file where memetables are written periodically.

Cassandra configuration consists of the following components:

Gossiper: A peer to peer communication protocol used to communicate between nodes in a cluster. The gossiper is responsible for making sure each node in a system knows the current state of other nodes including nodes that are unreachable [22].

Partitioner: A partitioner decides what nodes take precedence on receiving data and also how data is distributed across nodes in a cluster. Partition key is used to uniquely identify each row of data and distributed across the cluster using token calculated by using the hash function of the partitioner. Murmur3Partitioner is the default partitioning approach in Cassandra [21].

Replication factor: Replication factor is the number of replicas across the cluster which do not exceed the number of nodes. A replication factor of one means there is only one replica of each row on one node. Replication factor should be set at two or more.

Replica placement strategy: Replica placement strategy determines what nodes that the replicas can be placed. Cassandra utilizes this feature to ensure fault tolerance and reliability. Cassandra has two replication strategies, namely *SimpleStrategy* and *NewtworkTopologyStrategy*. *SimpleStartegy* is used for a single datacenter whereas *NewtworkTopologyStrategy* is used for multiple datacenters.

Snitch: A snitch determines what datacenter and racks a node belongs to. Snitch provide Cassandra with important network topology information which helps Cassandra in distributing replicas efficiently by grouping machines into datacenter and racks. Commonly in production world, *GossipingPropertyFileSnitch* is used as it defines node's datacenter and rack and also uses gossip to convey information to other nodes.

Cassandra.yaml configuration file: This is the main file for setting various configuration items such as cluster properties, tuning properties, caching parameters, backup and security properties among others. A node usually stores its configuration information in the `/var/lib/cassandra` directory. YAML stands for "YAML Ain't Markup Language".

System keyspace table properties: Storage configuration properties are set on a per keyspace or per table basis programmatically or using client application such as CQL [21].

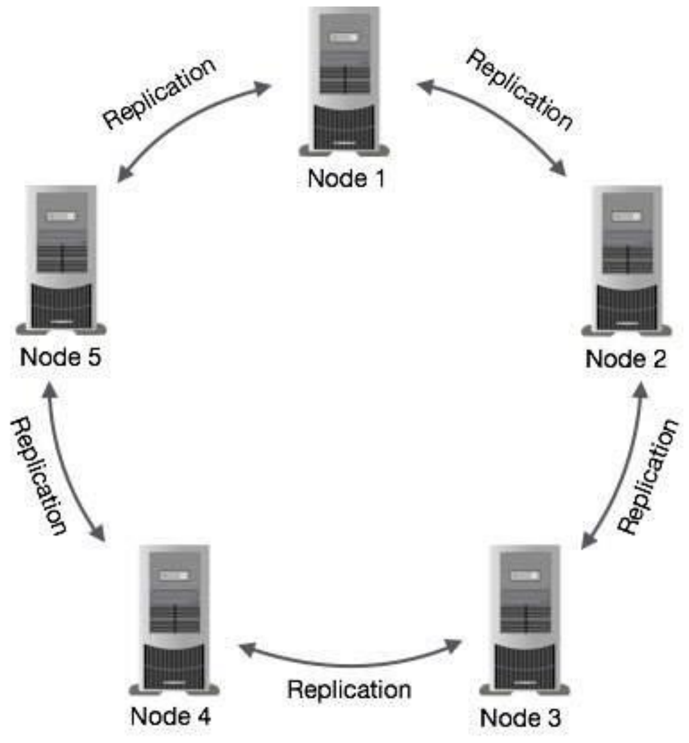


Figure 8

Replication in Cassandra [23]

Common Cassandra CQL statements

CQL consists of statements, as with SQL, some CQL statements make direct change to data, some look up data and some change the way data is stored.

```
cqlsh> CREATE KEYSPACE test with strategy_class =
'SimpleStrategy' and strategy_options:replication_factor=1;

cqlsh> USE test;

cqlsh> CREATE COLUMNFAMILY users (
   ...     key varchar PRIMARY KEY,
   ...     full_name varchar,
   ...     birth_date int,
   ...     state varchar
   ... );

cqlsh> CREATE INDEX ON users (birth_date);

cqlsh> CREATE INDEX ON users (state);

cqlsh> INSERT INTO users (key, full_name, birth_date,
state) VALUES ('bsanderson', 'Brandon Sanderson', 1975,
'UT');

cqlsh> INSERT INTO users (key, full_name, birth_date,
state) VALUES ('prothfuss', 'Patrick Rothfuss', 1973,
'WI');

cqlsh> INSERT INTO users (key, full_name, birth_date,
state) VALUES ('htayler', 'Howard Tayler', 1968, 'UT');

cqlsh> SELECT key, state FROM users;

        key | state |

 bsanderson |    UT |

  prothfuss |    WI |

    htayler |    UT |

cqlsh> SELECT * FROM users WHERE state='UT' AND birth_date
> 1970;

        KEY | birth_date |         full_name | state |

 bsanderson |       1975 | Brandon Sanderson |    UT |
```

Figure 9

Common CQL Statements [24]

As can be seen in the above example, CQL resembles very closely to SQL in basic statements. This is done purposefully to make it easier for users coming from SQL background. However, the similarity lessens beyond the basics. Keyspace in Cassandra require more specifications like strategy and replication factor spread across datacenters, different than in a relational database. SQL statements like JOIN, GROUP BY and FOREIGN KEY among others do not work in Cassandra even though data sets could have relationships. This dilemma is achieved by organizing any queries that will be run in the future into a column family so that reads are efficient. Other obvious difference is the ability to set Time to live in a row in Cassandra, accomplished using TTL command [25] such as:

```
/* Expiring Data in 24 Hours */

INSERT INTO myTable (id, myField) VALUES (2, 9) USING TTL
86400;
```

Figure 10

Setting Time to Live in Cassandra [25]

Deleting a data in Cassandra doesn't actually delete a data immediately instead tombstone is used to delete data marked to be deleted. Tombstone exist for a specific time period as set in the *gc_grace_seconds* value on the table. Then, the data is permanently removed during the compaction process. This process validates eventual consistency in Cassandra as data is not deleted immediately.

Tracing queries like `TRACING ON`; and `TRACING OFF`; in Cassandra display some useful debugging information such as network path of a query and latency.

CQL has many useful statements that could be further explored but that would require a whole new research.

MongoDB

MongoDB is a schema free document orientated database developed by MongoDB Inc., formally known as 10gen Inc. released initially in 2009. It is one of the popular NOSQL databases and the most popular in document orientated database and used by companies such as Facebook, Craiglist, eBay and Foursquare among many others. MongoDB was originally developed as a component of a Platform as a Service in 2007, however the software company decided to shift focus and release it as an open source developmental model in 2009 and is free. MongoDB also offers commercial support under a propriety license.

Since MongoDB doesn't rely on schema, this differentiates itself from relational database structure. This affordability makes it flexibile when it comes time to scale the database.

MongoDB has drivers for almost all popular programming languages including C++, JavaScript and C. It doesn't follow relational databases' table system and favors JSON-like documents with dynamic schemas, known in MongoDB sphere as BSON. MongoDB encapsulates and encodes data in the notion of a document in some standard format such as JSON, XML, BSON, YAML, binary forms (PDF, MS Word). Document is similar to row or record in relational database yet more flexible and can be retrieved based on the contents such as collections, tags, non-visible metadata and directory hierarchies.

Architecture of MongoDB

MongoDB stores data as documents in BSON (Binary JSON) where BSON documents contain one or more fields, where each field contains a value of a specific type of data including arrays, binary data and sub-documents. Document with almost similar structure are organized as collections. A collection could have multiple comments, multiple tags and multiple categories expressed as embedded array. Documents in MongoDB provide all data in one document whereas in relational databases, information for a given record is spread across multiple tables.

In MongoDB, the idea of schema is dynamic which means that each document can have different fields. This gives flexibility in terms of modeling of unstructured and polymorphic data.

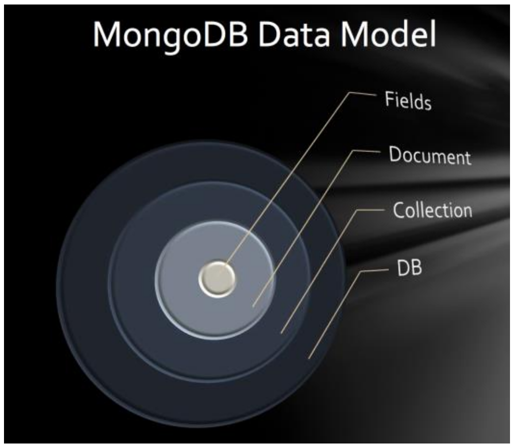


Figure 11

MongoDB data model [26]

Schema: MongoDB provides schema flexibility but designing it is still necessary. Developers and DBAs need to know what type of queries will be processed, how objects are managed and how documents change over time. MongoDB employs a dynamic schema where a new field can be added to a document without having to update the central system catalog, without affecting other documents in the system and without having to take the system offline.

Mongo Shell: MongoDb distributons come with an interactive JavaScript shell that can be used to issue all the supported commands including administrative processes.

Query model: Since MongoDB has and supports drivers for almost all major programming languages, MongoDB query model is implemented as methods or functions within an Application Programming Interface (API) of a programming language different from relational databases where SQL is used solely as a language. There are various queries that can be run in MongoDB, some of which are described below:

Key-value queries – Results are based on any field in the document such as primary key.

Range queries – Results are based on values defined as disparity such as less than, greater than, between or equal to.

Geospatial queries – Results are based on juxtaposition, intersection or inclusion criteria such as line, point, circle, etc.

Text search – Results are based on relevance using Boolean operators such as AND, OR, NOT.

Aggregation framework – Results are based on values returned by query such as min, max, average.

MapReduce queries – Results are based on complex JavaScript queries that are executed across the database.

Some basic MongoDB administrator steps and query examples are shown below.

```
# Install MongoDB

mkdir /data/lib

# Start Mongod server

.../bin/mongod # data stored in /data/db
```

```
# Start the command shell

.../bin/mongo

> show dbs

> show collections

# Remove collection

> db.person.drop()

# Stop the Mongod server from shell

> use admin

> db.shutdownServer()

# create a doc and save into a collection

> p = {firstname:"Dave", lastname:"Ho"}

> db.person.save(p)

> db.person.insert({firstname:"Ricky", lastname:"Ho"})

# Show all docs within a collection

> db.person.find()

# Iterate result using cursor

> var c = db.person.find()
> p1 = c.next()

> p2 = c.next()

> p3 = db.person.findone({lastname:"Ho"})

# Return a subset of fields (ie: projection)

> db.person.find({lastname:"Ho"}, {firstname:true})

# Delete some records

> db.person.remove({firstname:"Ricky"})

# To build an index for a collection

> db.person.ensureIndex({firstname:1})

# To show all existing indexes

> db.person.getIndexes()

# To remove an index

> db.person.dropIndex({firstname:1})

# Index can be build on a path of the doc.

> db.person.ensureIndex({"address.city":1})



# A composite key can be used to build index
```

```
> db.person.ensureIndex({lastname:1, firstname:1})
```

Figure 12

MongoDB administrator steps and queries [27]

Indexing: Index is a data structure which collects information about various fields in the documents of a collection. As such, MongoDB provides support for many types of indexes that can be declared in any field in the document including arrays. Some of the indexe type in MongoDB are Unique indexes, Compound indexes, Array indexes, TTL (Time to Live) indexes, Geospatial indexes, Sparse indexes and Text Search Indexes. Indexing is something that incurs an overhead which is why optimal balance of deleting unused indexes and effectiveness of an index should be measured periodically in MongodDB databases.

Sharding: MongoDB employs a technique called sharding which enables horizontal scaling out of databases on hardware or cloud infrastructure across multiple nodes. Sharding can also be interpreted as distributed storage. Sharding is performed by defining shard key and on a per collection basis. Sharding enables databases to overcome hardware limitations by spreading data across multiple physical partitions known as shards. Data is automatically balanced across sharded clusters as sharded servers are added or removed which is built into the database. This is one important distinction from relational databases making them highly scalable. Mapreduce queries can also be run on sharded clusters and is run parallel across shards [26]. There are many sharding policies available to distribute data across clusters depending on query and locality.

Range based sharding – Documents are distributed according to shard key value.

Hash based sharding – Documents are distributed per MD5 hash of the shard key value.

Location aware sharding – Documents are partiotioned according to user specified criteria such as specific datacenter or hardware.

In sharding model displayed below, there are three MongoDB config servers, one MongoS (Mongo Shard) server and three replica sets or shards. Replica sets can be divided as one primary and two secondary. Mongod should be installed in all the config servers, MongoS routing server and replica sets. Mongod is the primary daemon process for the MogoDB system. In shard setting, client connects to MonogoS where MongoS forwards the request to the appropriate shard server.

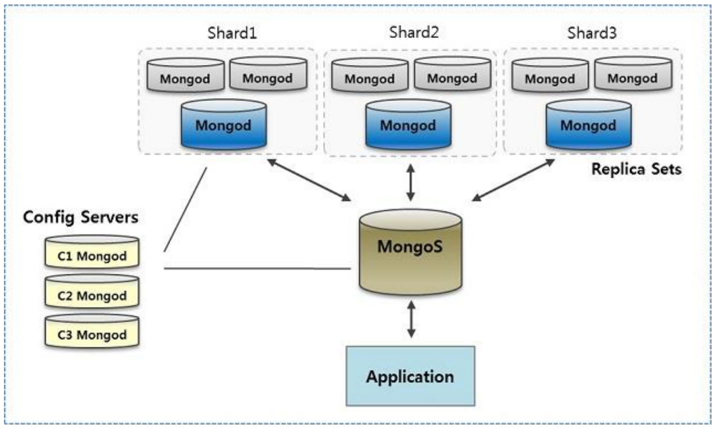


Figure 13

MongoDB sharding model [28]

In the sharding model, one partition key is specified for every collection stored in the sharding cluster.

```
# To define the partition key

db.runcommand({shardcollection: "testdb.person",

            key: {firstname:1, lastname:1}})
```

Figure 14

Defining Partition Key in MongoDB [27]

Map/Reduce in MongoDB

Parallel data processing can be performed in MongoDB by using a Map/Reduce framework similar to Hadoop Map/Reduce. Hadoop's MapReduce can be written in Java while MongoDB's is in JavaScript. Contrary to Hadoop's model, MongoDB Map/Reduce works by taking input from query result of a collection rather than HDFS (Hadoop Distributed File System), furthermore reduce output can be appended to an existing collection rather than an empty HDFS directory [27]. Map/Reduce works in the following way:

- Client defines a map function, a reduce function and query to input data and collection to store the output and sends the request to the MongoS server.
- MongoS distributes the request to all the members of each shard who then execute the query and pass the output to map function. Map function execute the code and give out key value pairs.
- The primary shard server will receive the key value pairs and execute the user-defined reduce function, returning the value to be written to the output collection as defined by the client.

The presence of multiple shard servers that can respond to Map/Reduce functions making availability one of the important traits of MongoDB.

DynamoDB

DynamoDB also known as Amazon DynamoDB is a NOSQL database service proprietary to Amazon.com as part of the Amazon Web Services and released in 2012. DynamoDB has similar data model to Dynamo and touts itself as “built on the principles of Dynamo”. Even though DynamoDB says it is built on the principles of Dynamo, it is widely known that Dynamo and DynamoDB have differences such as the Dynamo paper explains a simple key value store whereas DynamoDB supports secondary indexes, range keys, complex data types and conditional writes among others. Dynamo was published by Amazon as a paper but not as implementation. Dynamo is the name given to set of techniques that together form a highly available key-value distributed data store or storage system.

DynamoDB is cloud based and customers pay for what they use. DynamoDB is highly scalable with low latency and high throughput because of the use of key/value stores that are designed with simpler and less constrained data models than RDBMS [29]. Amazon DynamoDB releases user from operational overhead and takes that on itself freeing developers to focus on learning the DynamoDB API using a programming language of their choice. Amazon Web Services (AWS) provides SDKs (Software Development Kit) to develop applications using DynamoDB. Some of the SDKs available through AWS are .NET, Java, PHP, JavaScript, Python, Ruby, iOS and Android.

Architecture of DynamoDB

DynamoDB data model’s key components include tables, items and attributes where the database is actually a collection of tables. A table is a collection of items and an item is a collection of attributes. Contrary to relational databases, DynamoDB is schema less except for primary key meaning there is no predefined schema such as table name, column name and data types. An item in DynamoDB can have any number of attributes, however, item size is limited to 400 KB. Item size is calculated as the sum of lengths of its attribute name and values where values could be binary or UTF-8 lengths [30]. An attribute in an item is a name-value pair that can be single-valued or multi-valued set.

An example of items in a table is shown below where different product items can be placed in the table where Id is the primary key. In the example below, the table ProductCatalog has three items, one book and two bicycles. Item 101 has authors as a multi-valued attribute and items 201 and 202 have color as multi-valued attribute. Null or empty string are not allowed as attribute values.

```
{
Id = 101

   ProductName = "Book 101 Title"

   ISBN = "111-1111111111"

   Authors = [ "Author 1", "Author 2" ]

   Price = -2

   Dimensions = "8.5 x 11.0 x 0.5"

   PageCount = 500

   InPublication = 1

   ProductCategory = "Book"
```

```
}

{

Id = 201

   ProductName = "18-Bicycle 201"

   Description = "201 description"

   BicycleType = "Road"

   Brand = "Brand-Company A"

   Price = 100

   Gender = "M"

   Color = [ "Red", "Black" ]

   ProductCategory = "Bike"

}

{

Id = 202

   ProductName = "21-Bicycle 202"

   Description = "202 description"

   BicycleType = "Road"

   Brand = "Brand-Company A"

   Price = 200

   Gender = "M"

   Color = [ "Green", "Black" ]
```

```
            ProductCategory = "Bike"

        }
```

Figure 15

Example of Items table in DynamoDB [30]

Primary Key is one of the important specification of DynamoDB in addition to table. Two types of keys are supported by DynamoDB, which are:

Hash Type Primary Key – This type of key is made up of just one attribute known as hash attribute. An unordered index is built by DynamoDB based on the attribute.

Hash and Range Type Primary Key – Here, a primary key is made up of two attributes among which the first attribute is a hash attribute and the second is a range attribute. Just like the hash type primary key, an unordered index is built based on the primary key attribute and in addition, a sorted range index based on the range attribute is also built.

Table Name	**Primary Key Type**	**Hash Attribute Name**	**Range Attribute Name**
Forum (Name, ...)	Hash	Name	-
Thread (ForumName, Subject, ...)	Hash and Range	ForumName	Subject
Reply (Id, ReplyDateTime, ...)	Hash and Range	Id	ReplyDateTime

Figure 16

Key types examples in DynamoDB [30]

In the above example, Forum Table has Name as the only one primary key with hash attribute. Thread table has ForumName as the hash attribute name and Subject as range attribute name. Similarly, Reply table has Id as hash attribute name and ReplyDateTime as Range attribute name.

Secondary Indexes – DynamoDB also allows creation of secondary indexes when working with hash and range key. A secondary index can be helpful in querying data in a table using an alternate key in addition to the use of primary key. There are two types of secondary indexes – Local and Global. A local secondary index has the same hash key but a different range key. A global secondary index has a hash and range keys that are different from those on the table.

Data Types in DynamoDB – DynamoDB supports three different data types as listed below:

Scalar: Number, String, Binary, Null and Boolean.

Multi-valued: Number set, String set and Binary set.

Document: List and Map.

Operations in DynamoDB – In order to use DynamoDB and to make the most out of it, DynamoDB provides a set of different operations to work on tables and items that are listed below:

Table operations – Operations such as Create, Update and Delete can be used on a table in DynamoDB. Furthermore, operations such as UpdateTable, DescribeTable and ListTables can also be used to increase/decrease a table's throughput, retrieve table information and list tables respectively in order.

Item operations – Item operations allow to add, update and delete items from a table. Other item operations such as UpdateItem, GetItem and BatchGetItem also allow to update/add/delete attributes, retrieve a single item and multiple items respectively.

Scan and Query – Scan operation can be carried out in DynamoDB on a table or secondary index. Since running a Scan on large tables and secondary indexes could consume enormous resources, Query operation is suggested whenever appropriate. Query operation allows to query a table using hash attribute and/or range filter. Tables and secondary indexes can be queried only if the primary key is of hash and range type.

Data read and consistency – Multiple copies of each item is maintained in DynamoDB to ensure durability, so a successful operation means that it will eventually be consistent and won't mean that data is propagated to all the copies right away. This behavior in DynamoDB shows that it follows BASE model and values availability over consistency.

Neo4j

Neo4j is a graph orientated database developed by Neo Technology Inc. and initially released in 2007 with a version 1.0 being released in February 2010. Different editions of the databases are licensed under GNU General Public License, AGPL (Affero General Public License) and Commercial. Neo4j is known to be an ACID compliant transactional database with native graph and processing.

Neo4j is implemented in Java programming language, however it is accessible from softwares written in other languages using Cypher Query Language (CQL). CQL is a graph query language for Neo4j allowing expressive querying and updating of the graph store.

Neo4j is composed of two elements – a node and a relationship. Each node represents an entity and each relationship signifies association between two nodes. Since relationships take priority in Neo4j, foreign keys and other processing solutions such as MapReduce are redundant [31].

Architecture of Neo4j

Neo4j relies mostly on nodes and relationships, however there are many more terms that need to be understood in order to understand the architecture of Neo4j graph database.

Nodes – Nodes in graph databases are often used to entities but depending on domain, they may be used for other purposes as well. Nodes can contain properties. A property is defined when a node has zero or more named values. Besides properties and relationships, nodes can also be labeled with labels.

Relationships – Graph databases like Neo4j utilize relationships to connect nodes and also allow for finding related data. Similar to nodes, relationships have properties.

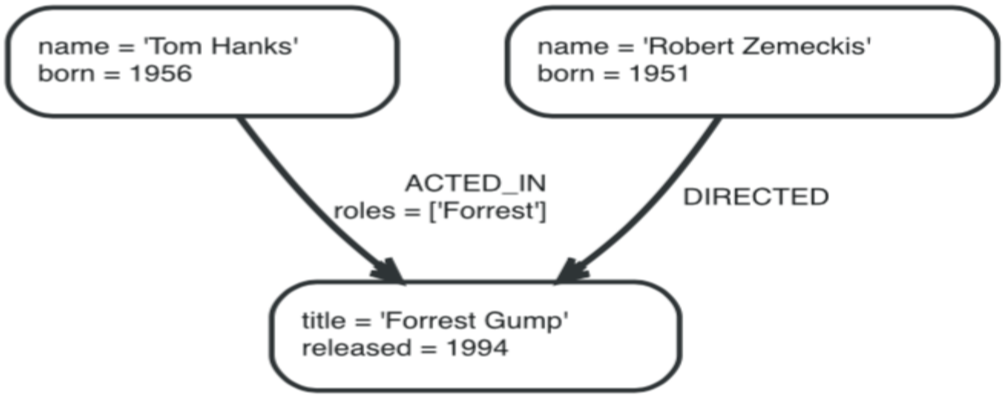


Figure 17

Relationship in Neo4j [31]

Properties – Properties are present in both nodes and relationships and could contain values such as Numeric, String, Boolean and collection of any other type of value.

Labels – Labels help assign roles or types to nodes. All nodes labeled with the same label belongs to the same set. Grouping nodes into sets eliminates the need to search the whole graph making queries more efficient. A node can have one or more labels. Label names can be any non-empty Unicode string.

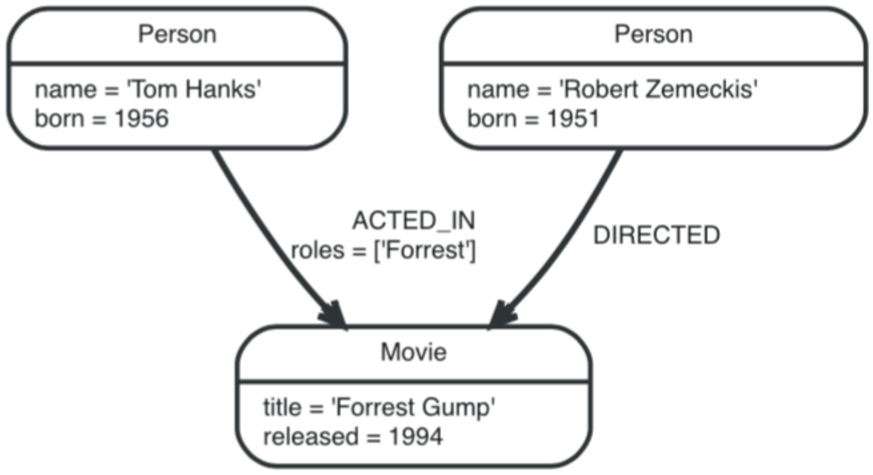


Figure 18

Labels in Neo4j [31]

Traversal – A traversal is a way of navigating through graph to find paths starting from one node to related nodes to find answers to queries.

Paths – A path is one or more nodes that connects relationships mostly retrieved as a query or traversal.

Schema – Schema is optional with Neo4j however introduction of schema allows for performance and modeling benefits. Schema is available only on a master machine of a Neo4j cluster.

Indexes – Neo4j creates and updates indexes once the properties to index have been determined. Indexes give performance boost compared to not having them.

Constraints – Neo4j has the ability to use constraints to keep data clean and any changes that break the rules are denied. As of this writing, Neo4j only supports unique constraints.

ACID/BASE Comparison of NOSQL databases

The four database systems – Cassandra, MongoDB, DynamoDB and Neo4j are discussed below. The database types are described by explaining all the four ACID properties and how they pertain to each database type.

Cassandra

Atomicity – Atomicity is supported at the row-level or partition level which means that inserting or updating columns in a row is treated as one write operation. However, in high availability configurations and fast write performance situations, atomicity is ignored. Multiple row updates into one all or nothing operation is not supported. A write success to one replica and failure on other replicas doesn't immediately trigger a rollback. Cassandra utilizes timestamps to determine a recent update and uses that for a client request.

Consistency – Cassandra seems to offer differing consistency based on the type of consistency utilized. Consistency could either be in CAP terms or in ACID terms. Most of the times, consistency in Cassandra is thought to have eventual consistency.

> Tunable consistency – Cassandra allows to tune consistency and availability. Data across all the nodes in a distributed database cluster can be consistent per CAP theorem. Another thing to note here is the fact that how tuning availability and consistency will gives way to partition tolerance [32]. Since Cassandra does not lock nor have any dependencies in updating multiple rows or tables, a user is able to choose on a per operation basis.
>
> Linearizable consistency – This is used in cases where tunable consistency is not enough in a distributed, master-less database with quorum reads and writes. Linearizable

consistency gives a serial isolation level for lightweight transactions conforming to ACID terms. An example of an application that registers new accounts needs to ensure that only one user can register for one particular account. Linearizable consistency takes on the challenge by checking for the existence of the account before performing insertion into a non-concurrent map [33].

Isolation – Cassandra supports full row-level isolation which means that writes to a row are isolated to the client that is performing a write and are invisible to others until the first write is complete. This property of Cassandra is different from earlier Cassandra versions where partial updates in a row were visible to another user.

Durability – Durability is represented well in Cassandra as all writes to a replica are recorded both in memory and in commit log. Commit log help in recovering any lost writes due to a server crash or failure. Local durability and replication of data on other nodes strengthens durability.

Table 1

ACID properties in Cassandra

Atomicity	Supported at row-level. Ignored on high availability situations.
Consistency	Eventual Consistency. Focused on Partition Tolerance.
Isolation	Row-level isolation.
Durability	Recorded in memory and in commit log. Replication on nodes.

MongoDB

Atomicity – MongoDB provides atomicity at the single document level where writes aren't applied partially to an inserted or updated document. The operation can be called atomic because it either fails or succeeds for that document in its entirety. Modifying subdocuments inside a document is still atomic. Atomicity is unsupported to operations that span multiple documents or collections. There are ways to model operations so that they atomic however doing that would cause too much overhead and negate benefits of using a NOSQL database.

Consistency – Consistency is strong in primary MongoDB server even in replica set configurations. However, secondary nodes maybe out of date and MongoDB can only guarantee eventual consistency with respect to the primary MongoDB server [34]. By default, MongoDB prohibits reads from secondary servers because of the chances of inconsistent data, however this can be changed knowing that changing the default could cause inconsistent data reads. As per CAP theorem Consistency and Availability both cannot be achieved together as in this case.

Isolation – MongoDB, as per guarantees provided by ACID ensures complete isolation when a document is updated and any errors will cause the operation to roll back so that the client receives a consistent view of the document. There are patterns such as "*update if current*" and operators such as *$isolation* available in MongoDB that provide a way to achieve isolation.

Durability - MongoDB provides flexibility to developers in terms of MongoDB's write concerns where a developer can configure operations to commit to the application only after they have been flushed to the journal file on the disk. Parameters such as *syncdelay* and *journalCommitInterval* allow for that configuration options. Per MongoDB, this is the same model used by traditional databases to provide durability guarantees. There are situations where a developer can specify that a write be considered complete only after writing to N number of secondary. If durability is primary goal, then MongoDB allows to do that however performance might suffer because of writing to so many replicas.

Table 2

ACID properties in MongoDB

Atomicity	Available at single document level.
Consistency	Strong only in primary server. Eventual consistency.
Isolation	Complete Isolation.
Durability	Durable. Can be further configured.

DynamoDB

Atmoicity – Amazon's DynamoDB relies on BASE approach so which is why atomicity is not actually present. Atomicity can be achieved on a single item but this property is lost when it involves multiple items. There are ways to get around and have atomicity in DynamoDB such as using Atomic writes that involve applying a set of commands each applied to a DynamoDB item which when complete assures that either all or none of the commands are executed. The techniques have been developed by DynamoDB developers and can be used as extension of the Amazon Web Services for Java.

Consistency – DynamoDB supports both eventually consistent reads and strongly consistent reads. In case of eventually consistent reads, a read request immediately after a write operation might not show the latest change. Operations such as *GetItem, BatchGetItem, Scan or Query* might not reflect the latest data. In case of strongly consistent reads request, DynamoDB returns most up to date data which reflects updates related to write operations that were successful. By default, most operations are eventually consistent and strongly consistent requests can be obtained by specifying optional parameters

Isolation – By default, DynamoDB does not provide any isolation guarantees however there are ways to implement isolation levels as discovered by developers. One way is to achieve isolation is to read only committed changes which can be accomplished by taking advantage of the fact that the algorithm saves old item image away before it applies changes. This approach is not a full proof approach; however, it does avoid the weakest consistency read style. Another way that provides stronger isolation is by using locks where read transaction could be coded exactly like a write and rollback at the end of the transaction. Writes are expensive than reads which means there is an overhead here.

Durability – Amazon DynamoDB synchronously replicates data across three facilities within an Amazon Web Services region which makes it possible for a high uptime and durability. This could change with the service level agreement that is in place since DynamoDB is operated as a service on Amazon Web Services.

Table 3

ACID properties in DynamoDB

Atomicity	Present only on single item.
Consistency	Eventually consistent.

Isolation	Isolation absent by default.
Durability	Durability option present.

Neo4j

Atomicity – Atomicity is present in Neo4j as the database is left unchanged if any part of the transaction fails. Neo4j is setup so as all database operations that access the graph, indexes or the schema must be performed either as a transaction or they fail.

Consistency – Consistency is achieved with Neo4j since a transaction will only leave the database in a consistent state even to different master slave configurations. Consistency can be achieved if chosen to do so by manually applying locks, however this could affect performance.

Isolation – Isolation is also achieved as a modified data cannot be accessed by other operations during a transaction. Neo4j specifically provides a default isolation level which is READ_COMMITTED that means that only data that have been committed are accessible.

Durability – Neo4j provides durability by making sure the database can recover the results of a committed transaction. In most cases, Neo4j achieves durability by the use of update log which acknowledges write once flushed to disk.

Table 4

ACID properties in Neo4j

Atomicity	Atomicity is present.
Consistency	Consistency is mostly present.

Isolation	Isolation provided by default.
Durability	Full durability.

Data Analysis

All four database systems discussed provide varying level of ACID compliance, some more than the other. Some follow the BASE approach while other follow the CAP theorem. Here below, ACID properties are compared side by side of each database.

Table 5

ACID properties on NOSQL databases side by side

	Atomicity	Consistency	Isolation	Durability
Cassandra	Row-level	Eventual	Row-level	Durable
MongoDB	Single document	Eventual	Isolation present	Durable
DynamoDB	Single item	Eventual	Isolation not present	Option available
Neo4j	Atomicity present	Consistent	Provided	Durable

Among the four databases, Neo4j seems to be the most ACID compliant satisfying all four properties to some degree if not all. When it comes to atomicity, all the others i.e. Cassandra, MongoDB and DynamoDB are present only at the lowest level of data structure. All three again are eventually consistent suggesting one of the important properties of the BASE approach focusing more on availability rather than consistency. After Neo4j, MongoDB seems to provide better isolation whereas Cassandra only provides at the lowest level and DynamoDB doesn't provide any isolation at all. All four database systems have some or full degree of durability.

BASE properties in the different NOSQL systems can be compared as shown below with examples on how they are implemented. As can be seen, this is derived from the materials that were discussed above. All four databases show varying degree of availability and consistency. Soft State of NOSQL databases are not discussed as all NoSQL databases have soft state because of the consistency property that eventually replicates to other nodes.

Table 6

BASE properties on NOSQL databases side by side

	Basically Available	Eventually Consistent
Cassandra	Highly available	Eventual consistency
	Example: Distributed Storage. Multi data center. Linear scalability.	Example: Tunable for partition tolerance. Linearized for strong consistency.
MongoDB	Consistency over availability	Strong consistency
	Example: Uses replica sets. Distributed across datacenters.	Example: Consistent data only on primary MongoDB server. Reads allowed only from primary.

DynamoDB	Highly available (HA)	Eventual and strong consistency
	Example: data replicated across three datacenters in availability zones.	Example: By default, eventual consistency, hoever, strong consistency can be achieved.
Neo4j	HA configuration available	Consistent
	Example: Several slave databases can be configured to replicate master database.	Example: Data leaves master/slave only in a consistent state.

As can be seen the above table, all four databases show either eventual or strong consistency, however, strong consistency seems to be rare and depends on special configuration to achieve it.

Summary

The four database types discussed above showed how each could be similar to another and could be so much different from one another. One database system is more popular while the other is not as much.

This chapter also went into lengths in discovering BASE properties along with their relevancy in ACID world. The chapter analyzed each of the properties and compared each database against the other. This showed how each handled the different properties differently showing more data integrity in one than the other.

Chapter 5: Results, Conclusion and Recommendations

This chapter discusses the findings of this starred paper as a whole and conclusion drawn from the various results that were discovered in earlier chapter and also summarize the overall methodology in obtaining those results. The chapter presents the study questions and also describes how each question was answered by this study and provide reasoning as to how some questions were inadequately answered. Deriving from questions that need future research, recommendations are provided.

Results

The methodology used for this study was mostly based on review of materials such as white papers, manuals, technology blogs, scholarly articles and other publications among others. Each database type was described as much as they could be provided enough sources were available. Databases architecture were also described to better understand how they stack up against relational databases. Query languages were also discussed in the case that they were present so that the similarities and/or differences could be compared to that of structured query language commonly used in relational databases.

Each database type intended for this study were also analyzed to how they fared against ACID properties. As it is known, almost all databases type that were discussed in this starred paper are BASE compliant but the paper wanted to compare how they compared to ACID. It is a well-known fact that ACID compliant databases are more focused on data integrity. This approach of comparing ACID and BASE models gives us a good baseline to compare to when it comes to data integrity.

The following section summarizes the research questions that were intended to be answered by this starred paper and provides a description of how they were answered throughout the research paper.

How does BASE model used in NOSQL databases compare to ACID?

The research paper explained how ACID and BASE are different. ACID properties are different from BASE properties. As explained in the paper, ACID properties are the backbone of relational databases. ACID properties value data integrity to the fullest whereas BASE properties value data availability as their backbone. Both ACID and BASE have their own uses, neither can replace the other. Relational databases that follow ACID are used mostly in places where transaction concurrency is important such as financial institutions and retail places. These uses value transaction concurrency to the upmost and where transaction volumes are not enormous. BASE on the other hand, along with CAP theorem in some cases, value availability as the most important property in NOSQL databases. In databases that follow BASE properties, consistency and concurrency is important but only after availability is fulfilled.

How do these differences affect consistency, performance or scalability among others?

As this research paper suggested, BASE does suffer from consistency issues as not all the nodes in a cluster are updated at the same time giving rise to inconsistent data to users accessing a data that was modified and not updated concurrently. Consistency is often reached in NOSQL databases in due time, however that could still lead to concurrency problem.

NOSQL databases do take performance seriously as they are meant to run on multiple machines unlike RDBMS which usually are meant to run on single machine. Running on multiple machines makes it such that performance is better as data is readily available.

Scalability is one of the important properties of NOSQL databases and separate themselves from RDBMS distinctly. RDBMS, because of their data structure and schema, make them difficult candidate for scalability. In addition, write operations are not cost effective and horizontal scaling almost unavailable. NOSQL databases are easy to scale, nodes can be added to cluster with ease and in very cost effective manner. In NOSQL databases, they handle data that grow exponentially, horizontal scaling as well as vertical scaling is important. Vertical scaling means upgrading equipment while horizontal scaling means being able to add nodes.

Do these differences, if any, compromise data integrity more than ACID would?

Obviously, as the research found, when there is presence of inconsistent data, there is an extreme possibility that data integrity could be compromised. In NOSQL, because nodes in a cluster do not get updated data at the same time, data accuracy is a concern.

In RDBMS, data integrity is handled well and no two write operations on a data can occur at the same time. There are other different integrity mechanisms such as entity integrity, domain integrity, referential integrity and other user defined integrity in place in RDBMS that value data quality.

On the other hand, most of the NOSQL databases do not have built in data integrity solutions and rely heavily on applications to enforce data integrity which is a hard sell.

Practical Implication

Now that the different NOSQL databases have been discussed, it is essential to know where these databases are used. In other words, the business or practical implication of NOSQL databases should be known. NOSQL databases are mostly used in web applications where there are enormous amounts of data and where data accuracy is of not much significance. NOSQL databases are not preferred when it comes to complex transactions that cannot afford to lose data such as an inventory system, in that case, Relational database is much preferred. NOSQL databases are ideal where scalability is important that support live addition and removal of machines, load balancing and fault tolerance. Some use cases are mentioned below for each NOSQL database.

Cassandra: Cassandra and similar column family stores are used mostly in content management where contents are stored along with tags, categories, etc. It is used in places where expiring column feature can be used such as building video/desktop/mobile games. Some of the industry stalwarts that use Cassandra are Accenture, eBay, Coursera, Comcast, Hulu among many others.

MongoDB: MongoDB and similar document store databases are used mostly in cases where small continuous reads and writes are required. These are also popular where there are wide varieties of data types and access patterns. MongoDB is popular when it comes to scalability because of its sharding feature. MongoDB is used in industry by The Weather Channel, MetLife, Bosch, Expedia among many.

DynamoDB: Amazon, DynamoDB and similar key-value stores are used primarily in fast in-memory access cases, small read write cases and easier upgradability cases. Programming fluidity similar to MongoDB is another benefit for DynamoDB where it is used to develop APIs in different languages. Some of the industry pioneers that use DynamoDB are EA Sports, Nordstorm, Shazam, New York Times among others.

Neo4j: Neo4j and other similar graph orientated databases are popular in social networking platforms and used also in cases where fast navigation is required between entities. They are also used in situations where dynamic properties are needed such building relationships. Walmart and eBay use Neo4j for customer recommendations. Some other Neo4j customers in the industry include Cisco, Lufthansa, TomTom among others.

Conclusion

This research paper's focus was distinguishing BASE from ACID. The research paper described ACID properties and how they relate to relational databases. ACID properties were mentioned in this paper as a benchmark for data integrity. This paved way for the discussion about how BASE compared against ACID when it came to data integrity.

Accordingly, BASE properties were discussed specifically in how they were applied in four different NOSQL database systems from each column orientated, document orientated, key value orientated and graph orientated namely Cassandra, MongoDB, DynamoDB and Neo4j respectively. The different NOSQL databases were also discussed to reveal their architecture and peeped into their database query language if available.

In conclusion, this research paper's aim was to provide one single document that could look into NOSQL database types, distinguish from relational databases and also provide a view into how these NOSQL database types stood against ACID properties. This paper should make it easy for a reader to understand NOSQL database systems. The content of this report was presented in its entirety.

Future work

The scope of the paper was limited to presenting differences, distinction and comparison in a scholarly manner. However, more extensive study can be done on this subject matter given that there are no time and resource limitations. Some suggestions for future work that can be done on this subject matter are listed as follows:

- In a lab environment, setup all the different NOSQL database systems and setup clusters and nodes.
- Conduct performance testing.
- Conduct horizontal scalability testing by adding nodes.
- Conduct consistency testing.

These types of testing mentioned above will demand a lot of resources and time, which is beyond the scope of this paper. However, doing so would provide important data that could further provide additional insight into NOSQL systems.

References

[1] Zollmann, J. (2012, August 20). NoSQL Databases. Retrieved from Software Engineering Research Group: http://www.webcitation.org/6hA9zoqRd

[2] Mapanga, I., & Kadebu, P. (2013). Database Management Systems: A NoSQL Analysis. International Journal of Modern Communication Technologies & Research (IJMCTR), 1(7), 12-18.

[3] Bartholomew, D. (2010). SQL vs. NoSQL. Linux Journal, (195), 54-59.

[4] Seltzer, M. (2008). Beyond Relational Databases. Communications of The ACM, 51(7), 52-58. doi:10.1145/1364782.1364797

[5] Hammes , D., Medero , H., & Mitchell , H. (2014). Comparison of NoSQL and SQL Databases in the Cloud . *Proceedings of the Southern Association for Information Systems Conference* (pp. 1-8). Macon, GA: Southern Association for Information Systems Conference.

[6] Brewer, E. (2012). CAP Twelve Years Later: How the “Rules” Have Changed. Computer, 45(2), 23-29.

[7] Davis, M. A. (2012, April 9). NoSQL Equals NoSecurity. InformationWeek, 29. Retrieved from St. Cloud State University Library Gale Databases: http://www.webcitation.org/6hABCqOha

[8] Barry, D. (n.d.). ACID Properties. Retrieved November 1, 2014, from Service Architecture: http://www.webcitation.org/6hABV4EXq

[9] Mihalcea, V. (2014, January 14). A Beginner's guide to ACID and database transactions. Retrieved November 10, 2014, from Java Code Geeks: http://www.webcitation.org/6hABdYCvy

[10] Pritchett, D. (2008). Base: An Acid Alternative. Queue, 48-55.

[11] Altrafi, O., Mohamed, M., & Ismail, M. (2014). Relational vs. NoSQL Databases: A Survey. International Journal of Computer and Information Technology, 03(03), 598-601.

[12] Kopp, M. (2011, October 5). NoSQL or RDBMS? – Are we asking the right questions? Retrieved November 11, 2014, from about-performance: http://www.webcitation.org/6hABvlSHG

[13] Englund, B., & Södergren, P. (2011). Investigating NoSQL from a SQL Perspective. KTH Computer Science and Communication.

[14] A Comparison of NoSQL Database Management Systems And Models. (2014, February 21). Retrieved from Digital Ocean: http://www.webcitation.org/6hACY1MFW

[15] Ho, R. (2010, October 10). BigTable Model with Cassandra and HBase. Retrieved from Database Zone: http://www.webcitation.org/6hACnZO92

[16] Comparing Document Orientated and Relational Data (n.d.). Retrived from Couchbase: http://www.webcitation.org/6hADEspjd

[17] Reeve, A. (2013, November 25). Big Data Architectures - NoSQL Use Cases for Key Value Databases. Retrieved from EMC2 InFocus: http://www.webcitation.org/6hADVIDWr

[18] Data Structures/Hash Tables (n.d.). Retrieved from Wikibooks: http://www.webcitation.org/6hAECwYgV

[19] Olson, K. (n.d.). What are the main differences between the four types of NoSql databases? Retrieved from Quora: http://www.webcitation.org/6hAEOkiBj

[20] Graph database (n.d.). Retrieved from WhatIs: http://www.webcitation.org/6hAEVLZ1N

[21] Apache Cassandra™ 2.0. (2015, August 13). Retrieved from http://www.webcitation.org/6hAEi0UxB

[22] Architecture Gossip. (2011, November 13). Retrieved from Cassandra Wiki: http://www.webcitation.org/6hAEsZaxd

[23] Agundez, R. (2016, Feburary 08). Explore Apache Cassandra. Retrieved from Qualogy: http://www.webcitation.org/6hAFO6ZZv

[24] Getting Started with Cassandra (2011, July 11). Retrieved from Blogspot: http://www.webcitation.org/6hAF5aZbF

[25] Meng, A. (2014, November 23). Cassandra Query Language (CQL) vs SQL. Retrieved from http://www.webcitation.org/6hAFiekDV

[26] Tovi, I. (n.d.). MongoDB - Document Orientated Database. Retrieved from The Academic College of Tel Aviv: http://www.webcitation.org/6hAFwTNBW

[27] Ho, R. (2012, April 06). An Articulate Introduction to MongoDB. Retrieved from Database Zone: http://www.webcitation.org/6hAGNIYew

[28] Ye, W. (2013, October 3). Setup MongoDB with Sharding Infrastructure. Retrieved from Wayneye: http://www.webcitation.org/6hAGcliqd

[29] Doble, W. (2014, September 1). Comparing the Use of Amazon DynamoDB and Apache HBase for NoSQL. Retrieved from Amazon Web Services Docs: http://www.webcitation.org/6hAGvHUT7

[30] Amazon DynamoDB - Developer Guide. (2012, August 10). Retrieved from Amazon Web Services Docs: http://www.webcitation.org/6hAH31OjL

[31] Why Graph Databases? - Neo4j Graph Database. (2015). Retrieved from Neo4j: http://www.webcitation.org/6hAHWGLqa

[32] Tudorica, B. G., & Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. 2011 RoEduNet International Conference 10th Edition: Networking in Education and Research.

[33] Consistency. (2016, January 13). Retrieved from Datastax: http://www.webcitation.org/6hAHn1rq3

[34] Han, J., E, H., Le, G., & Du, J. (2011). Survey on NoSQL database. 2011 6th International Conference on Pervasive Computing and Applications.