Shendi University Journal of Applied Science, Issue (10) , June 2023 (7): p 85-98

AN EVALUATION OF CONSISTENCY MODELS IN NOSQL DATABASES

Mohamed Siddig Hassan[1] and Mohamed Bakri Bashir [2]

[1] Facutly of Computer Sciences. Shendi University, Shendi, Sudan
msh@ush.edu.sd

[2] Faculty of Computer Science, AlTaif University, Saudi Arabia
mhmdbakri@ush.edu.sd

ABSTRACT

This study focuses on the various consistency models are used in ***NoSQL*** databases. ***NoSQL*** databases are designed to handle large volumes of unstructured or semi-structured data, and they often use distributed architectures to achieve high scalability and availability. However, maintaining consistency across this type of database can be challenging due to the distributed nature of the database and the concurrent access of multiple users or applications. This study explores the different types of consistency models used in ***NoSQL*** databases. The study examines the strengths and weaknesses of each model and how they ensure data consistency and integrity in distributed databases. The findings of this study can help database administrators and developers choose the appropriate consistency model for their ***NoSQL*** database based on their specific requirements and use cases.

KEYWORDS: *NoSQL, NoSQL-Database, DBMS, Consistency , eventual consistency*

1. INTRODUCTION

1.1 BACKGROUND

In this days, the data was growing rapidly with time, for example, the amount of data volumes was generated by Twitter and Facebook users in each day was estimated by 12 and 500 TB [1] [2]. The Relation Databases can't handle this amount of data [3]. Therefore, an enterprise companies tend to find new types of databases that meet the requirements for handling and processing this massive of data that generated every day. ***NoSQL*** (Not-Only-SQL) is a new database to handle the massive data by supporting cluster architecture has recently become very popular. Google and Amazon are considered pioneers in producing these types of databases and their products are today considered one of the most distinguished products based on the concept of ***NoSQL***.

This type of database is based on distributed database system models, and ***CAP*** Theorem Presented by Eric Brewer for management database systems ***DBMs***. Consistency, Availability and partition tolerance submitted by **Eric Brewer** since 2000. ***Consistency*** refers to the fact that, at any one time, all copies of the data in the system seem identical to an outsider. Availability describes how the system as a whole keeps running even when a node fails. It is necessary for

partition-tolerance that the system function even in the face of random message loss. Similar to network flaws [4].

In related database management systems, ACID is preferred to. ***Atomicity***: The state in which all of the transaction's operations will either succeed or fail. ***Consistency***: The absence of inconsistent data in any transaction or its effects. A transaction will act as though it is the sole action taking place when it is ***isolated***. ***Durability***: The inability for an operation to be undone after it has been carried out [4]. Most of References are divided the ***NoSQL*** databases models into: (1) Key-value ***NoSQL*** model, (2) Column-Family ***NoSQL*** model, (3) Document ***NoSQL*** model, and (4) Graph ***NoSQL*** model.

NoSQL DBMSs also demand eventual consistency, which implies that initially, not all redundant nodes will have the most up-to-date data, but eventually, all servers will have the same data, in order to increase performance. ***NoSQL*** DBMSs support multi-level consistency when various applications have varying needs for consistency. N must always be greater than or equal to C, where C denotes the consistency level that must be met for a read/write operation and N is the number of replicated nodes. [5].

There were seven sections in this article. Introduction is found in section one. The related work is presented in section 2, and the consistency in ***NoSQL*** DBMS is covered in section 3. The approach is introduced in section 4, and the findings and discussions are presented in part 5. There were 5 sections in this essay. Introduction is found in section one. The related work is presented in section 2, and the consistency in ***NoSQL*** DBMS is covered in section 3. The approach is introduced in section 4, and the findings and discussions are presented in part 5 and 6.

1.2 MOTIVATION

NoSQL Database are weak support for the *ACID transactional* guarantees and strong data consistency features, because this challenge the developers have solve this problem within code of the applications and may cause the difficulties in the application development life cycle and also reduce the efficiency of the production development.

1.3 CONTRIBUTION

In this study we present the consistency models in ***NoSQL*** databases. The researcher contributions are stated, (1) To present a consistency models in ***NoSQL*** Specially in Key-value data model. (2) Proposed a best model of consistency in may applied in ***NoSQL*** data store. (3) Fill the gap of the study in the consistency models

2. RELATED WORKS:

Various representative studies have frequently evaluated eventual consistency in ***NoSQL*** databases, frequently taking the performance impact into account. To assess how read and write operations, database replication, and eventual consistency of ***NoSQL*** DBMSs are impacted, the authors of [2] recommend adopting a probabilistic technique. recommend adopting a probabilistic technique. Although the authors give model validation with a small relative error, performance and availability issues are not taken into account in the works. Mathematical models to specify and assess ultimate consistency on data storage systems have been provided by the study of Attiya et al. [6] Performance, however, is not taken into account. Using DynamoDB to explore the impact of operation delay, such as write operation, on eventual consistency was done in the work by Bailis et al. [7]. Assessing the implications of operation delay, such as a write operation, on eventual consistency using DynamoDB. They also provide a method for calculating consistency that accounts for the number of database replicas. In evaluating the performance of three ***NoSQL*** DBMSs (MongoDB, Cassandra, and Riak), Klein et al. took into account the number of clients. Results from experiments suggest that strong consistency may result in a 25% decrease in system performance. [8]. According to Huang et al., queue length should be used as a metric for consistency. DBMS Cassandra has been utilized in tests. [9]. In [10] The researchers conducted several studies to find out how database consistency impacts energy usage. Results indicate that energy use is highly influenced by effort. Liu et al. attempted to determine how long it would take to update data in databases with eventual consistency. In the study [11] the researchers offer a probabilistic approach. Osman et al.'s offer a Petri Net Model for evaluating the Cassandra DBMS's performance while taking into consideration different redundancy strategies and cluster sizes. The model delivers values that are close to those of the actual system, according to the results, but they do not address system availability. [12].

3. NoSQL Consistency

An operation sequence that usually complies with the ACID properties is referred to as a *transaction*. If a transaction is successful, it is said to commit; if not, it is called to abort [13].

A single valid state for all database instances can be characterized as *consistency* in database management systems ***DBMS***. A *database management system consistency* can be defined as a single acceptable state for all database instances as long as the data remain the same across all redundant database servers. [14, 15]. Because a DBMS must guarantee that the returned data is the most recent for readings and must confirm that the write operation has been successfully

performed on each requested server, this has an influence on performance. Since there are more database replicas present in distributed systems, availability is also impacted by consistency policy in a manner similar to how accessibility is. In order to boost efficiency and availability, NoSQL databases use eventual consistency, which permits temporary inconsistency (i.e., not all redundant servers will immediately have the most recent data) and permits a database replica to return its available data (which may not be the newest). There will finally be consistency across all redundant servers [16]. ***NoSQL*** DBMSs permit the adoption of different consistency levels (i.e., the bare minimum of redundant servers holding the most recent data), which are adjusted in accordance with an application's needs [17]. This contributes to closing the inconsistency. Fewer servers need to be upgraded because of fault tolerance and increased availability. Another ***NoSQL*** trait is strong consistency, which always returns the most recent data.

According to the most recent studies, consistency models can be categorized into a variety of categories, including strong consistency, weak consistency, eventual consistency, causal consistency, read-your-writes consistency, session consistency, monotonic reads consistency, and monotonic writes consistency.

1) ***Weak-Consistency Model***:

   This model, as the name indicates, reduces consistency. It specifies that a read operation does not guarantee the return of the most recently stored value. It also does not ensure the sequence of events [18]. The time interval between a write operation and the point at which each read operation provides the updated data is referred to as the inconsistency window [19]. Because there is no need to include more than one replica or node in a client request, this paradigm results in a highly scalable system.

2) ***Eventual Consistency Model***:

   A consistency model that ensures if there is no additional updates on a given item, all the reads to that item will eventually return the same value [19]. Replicas frequently arrive with the same data state. Read operations might not always return the most recent version while this procedure is in progress. The connection lags between replicas and their sources, system load, and the number of replicates involved will affect the inconsistency interval. [18]. This method is half-way between a strong-consistency model and a weak-consistency model. Many ***NoSQL*** databases provide Eventual Consistency as a feature. The world's most popular companies that use Cassandra can provide availability and network partitioning to such a degree that it does not hinder functionality. Facebook, the company that originally developed Cassandra, is one of them.

3) ***Strong Consistency Model***:

The identical value will be returned by any read from any replica thanks to a robust consistency model. All clients will utilize the identical data entry and data, and each transaction must appear to be committed instantly. The write action must commit before a read operation may access the updated version of an instance. Every storage system instance accepts a particular global sequence of events. [19] , [20].

4) ***Casual Consistency Model***:

Any operations that recognize the update on an element are required to take the modified value into account. The eventual consistency model will be used in the event that another process does not acknowledge the write operation [18]. Although less dependable than sequential consistency, causal consistency is more dependable than eventual consistency. When the Eventual Consistency model is reinforced to be Causal Consistency, the system's availability and network partitioning properties are decreased. [18].

5) ***Read-Your-Writes Consistency Model***:

With the help of the read-your-writes consistency model, it is made sure that a replica is at least current enough to include changes made by a single transaction. Transactions are applied sequentially, therefore by guaranteeing that a replica has a particular commit applied to it, we can make sure that all transaction commits that took place prior to the given transaction have already been committed to the replica. If a process updates an object, that process will always take into account the modified value. Other processes will eventually read the modified value. Therefore, read-your-writes consistency is achieved when the system guarantees that every attempt to read a record that has been modified will return the updated value.

6) ***Session Consistency Model***:

A process will follow a read-your-writes consistency model for the length of a session if it makes a request to the storage system while it is operating within that session. All reads are current with the session's writes using session consistency, although writes from other sessions may need to wait. Although everything arrives in the correct order from prior sessions, the data is not always guaranteed to be up to date. This offers excellent consistency at half the cost of good performance and availability.

7) ***Monotonic Read Consistency Model***:

Every time a process reads a value, it returns that value or one that is more recent [15]. It implies that the same item is read by the same process consistently and in the same order. However, this does not guarantee that read operations between processes on the same object will be ordered monotonically. Because of this, monotonic readings ensure that a process that reads r1, r2, and r2 cannot experience a state that is earlier than the writing

represented in r1; reads, by nature, cannot travel backward. Monotonic readings do not apply to operations carried out by different processes; they only apply to those carried out by the same process. There are full monotonic readings available: Even during a network split, all nodes can advance [21].

8) ***Monotonic Write Consistency Model***:

Before any more write operations by the same process on the same object, a process-initiated write action on that particular object must be completed [19]. In other words, the same process writes to the same object consistently in the same order. However, this does not guarantee that write operations between processes on the same object will be ordered monotonically. The effect of this is that monotonic writes guarantee that if a process writes w1, then w2, then all processes will observe w1 before w2. Monotonic writes do not apply to operations carried out by different processes; they only apply to those carried out by the same process. All monotonic writes are available: Even during a network split, all nodes can advance [22].

9) Time-Line Consistency Model:

Yahoo created this consistency model especially for YAHOO PNUTS in order to solve the inefficiencies of serializable transactions of the big data and its relation with their geo-replication. Furthermore, it seeks to reduce the shortcomings of eventual consistency [23]. ***NoSQL*** databases are support eventual consistency instead of strong consistency. They do not support database transactions which ensure strong data consistency [24].

Each type of ***NoSQL*** models support many level of Consistency for example the eventual consistency supported may levels of consistency For the confirmation of an activity at consistency level **ONE**, just one node or server is required (such as a write or read). For level 2 operations, **TWO** nodes are needed, and while reading, the most current data from both servers is taken into consideration. The **QUORUM** policy [25], which requires that the least integer bigger than 50% of the database nodes be used to determine consistency, is compatible with a level like this. Like the **ALL** policy, Level Three asks confirmation from each node. [26]. The latest information is constantly accessible thanks to reading (high consistency) [27].

Table (1): *NoSQL* Consistency Models

Consistency Model	Grantees	
Weak Consistency Model	A read operation will not really support serialization and doesn't guarantee that it will provide the value that was most recently saved in memory.	
Session Consistency Model	Consistency with read-your-writes is only guaranteed during a session.	
Read-Your-Writes Consistency Model	An operation always receives the most recent update on read operations.	
Monotonic Reads Model	Every time return the same value as the last reading, or one that is more recent.	
Monotonic Writes Consistency Model	Prior to performing any more writes, a write operation must always complete.	
Casual Consistency	Order of actions overall with a causal connection	
Strong Consistency	Serializability	A set of operations is composed of concurrent computations of a group of serialization units.
	Linearizability	Every operation is immediately seen in the overall, sequential order of events, or it is handled as a single operation.
Eventual Consistency	Eventually, the state of the updates will be consistent across all replica nodes.	
Time-line Consistency	The actions are performed on the same record by all replica nodes in the same "correct proportion".	

Table (2) Consistency Model in *NoSQL* Databases

***NoSQL* Database**	**Data Model**	**Consistency Model**	**Applications/Services**	**API**
Amazon Dynamo	Key-Value	Eventual Consistency	E-Commerce Platforms like Amazon Stores (**AWS A**mazon **W**eb **S**ervices)	Multiple Consistency Level
Cassandra	Column-Family	Eventual Consistency	Facebook, Netfelx, inbox search, eBay, Sound Cloud, Rack Space Cloud	Multiple Consistency Level (ONE, TWO, ALL, QURAM)
Raven DB	Document	Eventual Consistency	Toyota	Multiple Consistency Level
MongoDB	Document	Eventual Consistency	SAP AG Software Entreprise, MTV, Vodafone, AMAR BANK	CRUD API
Raik	Key-Value	Eventual Consistency	Yammer Social Network, Github	Multiple Consistency Level
Yahoo PNUTS!	Multi-Model	Timeline Consistency	Yahoo Mail	Multiple Consistency Level
Apache HBase	Column Family	Strong Consistency	Facebook messenger, using Hadoop for large set of application	JSON API
Microsoft Azure	BOLB Tables	Strong Consistency	Office 365, OUTLOOK, Bing	RESTfull API
Redis	Key-Value	Strong Consistency	Flicker, Instagram	JSON API
Google Spanner	MultiModel	Strong Consistency	Google F1	SQL-Like

Many applications demand either a rigorously strong type of consistency or just static eventual consistency. However, consistency requirements are not evident for another type of applications since they are dependent on data access behavior dynamical, client demands, and the results of reading inconsistent data such as ecommerce platforms because these kinds of applications, the fast accessibility and availability are critical. Strong consistency techniques may therefore be unaffordable. Although they are preferred for some applications, great levels of uniformity are not always required. In situations like these, undesirable results are caused by either immobile eventual or strong sorts of consistency. When storage systems are dispersed geographically, strong consistency guarantees

become unaffordable due to high network latencies. As a result, applications requiring high availability and performance are best served by weaker consistency semantics, such as eventual consistency.

4. Methodology

The one of the contribution is to select the best type of the consistency if we setup the eventual consistency. And determine how the cost of replicated NoSQL data storage varies depending on the consistency level being used. As a result, we add to our earlier research by figuring out how much it will cost to use ScyllaDB.

4.1 ScyllaDB

ScyllaDB is a distributed *NoSQL* wide-column database for data-intensive applications that require high performance and low latency, its sharded cluster, replica set or standalone, It is an open source *NoSQL* database and support cloud [28].

The number of replicas (in a cluster) that must acknowledge a read or write operation before the coordinator node may judge the operation was successful is determined by a Consistency Level (CL). That means the CL maybe is the important factor for the NoSQL database that used the eventual consistency.

Table (3) describe Scylla Consistency levels [28]

Consistency Level	With Replicas Must Response	Consistency	Availability
ANY (Write Only)	The closest replica, according to the snitch. After an indicated handoff, write succeeds if all replica nodes are down. Assures never-failing writes while offering low latency.	Lowest (Write)	Highest (Write)
ONE	The Snitch's assessment of the closest replica. The requirements for consistency are not overly strict.	Lowest (READ)	Highest (READ)
TWO	The closest two replicas as determined by the Snitch.		
THREE	The closest three replicas as determined by the Snitch		
QUORUM	A simple majority of all replicas across all datacenters. This CL allows for some level of failure		
ALL	All replicas in cluster	Highest	Lowest
LOCAL_QUORUM	Confined to the same datacenter as the coordinator.	Low in multi-data centers	
EACH_QUORUM (WRITE ONLY)	A simple majority in each datacenter.	Same across datacenter	
LOCAL_ONE	Same as ONE, but confined to the local datacenter.		
SERIAL	Returns results with the most recent data. Including uncommitted in-flight LWTs. Writes are not supported, but read transactions are supported.	Linearizable	
LOCAL_SERIAL	Same as SERIAL, but confined to a local datacenter. Writes are	Linearizable for the local DC	

Consistency Level	With Replicas Must Response	Consistency	Availability
	not supported, but read transactions are supported.		

4.2 Experimental Setup

We deploy a single replica set in Amazon Elastic Compute Cloud (AWS) to conduct the tests (EC2). With 30 nodes on the USA (us-east-1) site and 5 nodes in the same geographical region and different availability zones, we deployed ScyllaDB on two data zones. Each node has the following specifications:

- **250 GB NVMe SSD,**
- **32 GB of Memory,**
- **8-cores INTEL CORE.**
- **Standard architecture of 1000 Gbit/s dark fibers**
- **OS: Linux Ubuntu 18.4**

With ScyllaDB, we used a replication factor of three copies, with two of them allocated to Zones 1 and 5.

4.3 Workloads

For testing the eventual consistency and casual consistency we had initialize a workload which would perform multiple sequential operation for a single session rather than independent point quires, we used a workloads based on social media of twitter. Which involve each client doing mix of read and writes (read tweets, and write tweets) or just a serializable of read (insights, status checks) in the session. A performance will done with enabling the transactions, and verifying the setting for the consistency, read performance, write performance, as well as the number of threads.

4.4 Benchmarks

We need a benchmark tool that makes use of the features of various workloads in order to run the experiment and assess the consistency levels; in this example, we use the Yahoo Cloud Service Benchmark (YCSB) 1.12.0. YCSB can be utilized with a variety of programs, including Additionally, YCSB displays genuine cloud features like scale-out, elasticity, and high availability, and we use it to execute Workload A, a workload with a high read-to-update ratio (60:40). After replication, our workload in both environments consists of 10 million operations on 5 million rows for a total of 50.84 GB of data.

5. Results and Discussion

Many applications that demand low latency writes can't wait for the response of replication of each write and use write all for doing writes for all nodes that's mean some of writes may be not written and rolled back.

Figure (1) Shows that Comparisons between two types of eventual consistency had applied to ScyllaDB, and its throughputs. In this figure the results show that the applied of the configuring the ALL of eventual Consistency is the best in threads and throughputs rather than Quorum.

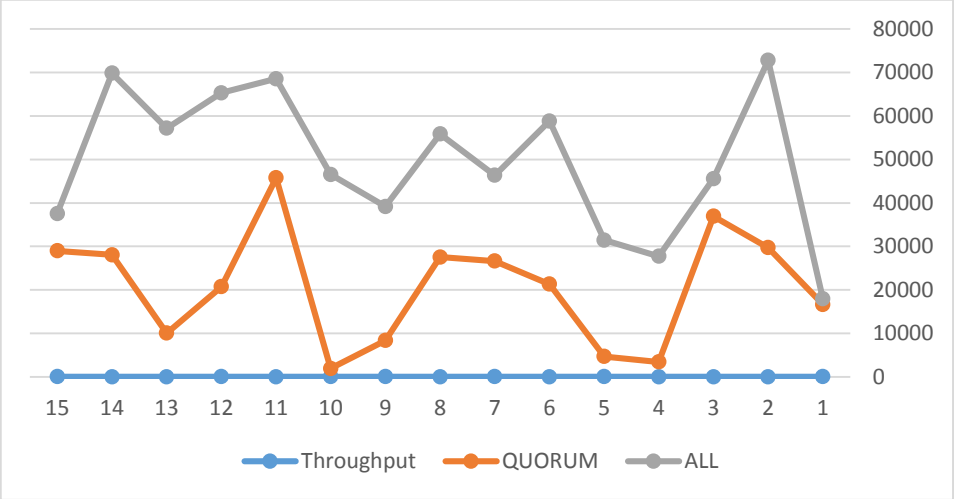

Figure (1): Describe the Throughputs Vs. two types of eventual consistency

Figure (2): Describe the Read/Write Operation Throughputs

On the other hand, the second figure shows that the reading and writing factor with the response time. It is clear from this that the best performance in balancing the load between the reading and writing process is when the consistency settings are applied in configuration of QUORUM consistency in the replica set.

6. Conclusion

This Study was aimed to draw a key for the types of consistency of NoSQL databases. Because NoSQL was used widely at this time and select one of the best models of consistency the eventual consistency and test it with one of the *NoSQL data model* is the column model.

7. Future works

In this study we test and implementation of the eventual consistency for the column NoSQL database. We recommend that study the other types of consistency with other types of *NoSQL* data model. And study the effect of each types of consistency over the truncations throughputs.

References

[1] M. G. (. S. Sakr (ed.), Large Scale and Big Data:Processing and Management, Boca Raton,: Auerbach Publications, 2014.

[2] U. G. A. P. E. T. Aleksey Burdakov, "Estimation Models for NoSQL Database Consistency Characteristics," in *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2016.

[3] U. G. A. P. A.V. Burdakov, "Comparison of table join execution time for parallel DBMS and MapReduce," in *Software Engineering / 811: Parallel and Distributed Computing and Networks / 816: Artificial Intelligence and Applications Proceedings*, Innsbruck, Austria, 2014.

[4] R. H. S. Jablonski, "NoSQL Evaluation A Use Case Oriented Survey," in *internation conference on cloud and services computing*, 2011.

[5] E. B. E. T. a. M. N. d. O. J. C. Gomes, "Performability Model for Assessing NoSQL DBMS Consistency," in *IEEE International Systems Conference (SysCon)*, Orlando, FL, USA, 2019.

[6] F. E. a. A. M. H. Attiya, "Limitations of highly-available eventually-consistent data stores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 141-155, 2017.

[7] S. V. M. J. F. J. M. H. a. I. S. P. Bailis, "Probabilistically bounded staleness for practical partial quorums," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 776-787, 2012.

[8] J. W. P. S. Y. J. B. a. J. Z. X. Huang, "An experimental study on tuning the consistency of nosql systems," *oncurrency and Computation: Practice and Experience*, vol. 29, no. 12, pp. 29-41, 2017.

[9] I. G. N. E. P. D. K. P. a. C. M. J. Klein, "Performance evaluation of nosql databases: A case study," in *in Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, New York, NY, USA, 2015.

[10] S. I. Y. L. G. A. M. S. P. a. L. B. H. E. Chihoub, "Exploring energy-consistency trade-offs in cassandra cloud storage system," *International Symposium on Computer Architecture and High Performance Computing* , pp. 146-153, 2015.

[11] S. N. J. G. M. R. R. I. G. a. J. M. S. Liu, Quantitative Analysis of Consistency in NoSQL Key-Value Stores, Cham: Springer International Publishing, 2015, pp. 228-243.

[12] R. Osman and P. Piazzolla, "Modelling Replication in NoSQL Datastores," *Cham: Springer International Publishing*, pp. 194-209, 2014.

[13] C. F. d. V. R. P. J. O. Coelho FACL, "pH1: A Transactional Middleware for NoSQL," *IEEE 33rd International Symposium on Reliable Distributed Systems Available*, 2014.

[14] O. G. A. a. M. O. I. M. A. Mohamed, "Relational vs. nosql databases: A survey," *International Journal of Computer and Information Technology*, vol. 3, no. 3, pp. 598-601, 2014.

[15] J. R. G. Paz, "Introduction to azure cosmos db," in *Microsoft Azure Cosmos DB Revealed: A Multi-Model Database Designed for the Cloud*, Berkeley, CA: Apress, 2018, pp. 1-23.

[16] E. R. a. J. W. L. Perkins, Seven databases in seven weeks: a guide to modern databases and the NoSQL movement, Pragmatic Bookshelf,, 2018.

[17] R. O. a. W. J. K. G. Haughian, "Benchmarking replication in cassandra and mongodb nosql datastores," in *Database and Expert Systems Applications*, Cham: Springer International Publishing,, 2016, pp. 152-166.

[18] W. (. ,. 5. 4.-4. Vogels, "Eventually consistent: Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.," *Communications of the ACM*, vol. 52, no. 1, pp. 40-44, 2009.

[19] P. B. a. A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *Queue*, vol. 11, pp. 03-20, 2013.

[20] P. a. M. V. Viotti, "Consistency in non-transactional distributed storage systems.," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, pp. 1-34, 2016.

[21] Jepsen., "Monotonic Reads.," 2022. [Online]. Available: https://jepsen.io/consistency/models/monotonic-reads. [Accessed 01 01 2023].

[22] J. M. Writes. [Online]. Available: https://jepsen.io/consistency/models/monotonic-writes. [Accessed 02 01 2023].

[23] H. E. Chihoub, "Managing Consistency for Big Data Applications on Clouds: Tradeoffs and Self-Adaptiveness," *THÈSE / ENS CACHAN - BRETAGNE*, 2013.

[24] A. O. M. Y. J. T. María Teresa González-Aparicio, "Transaction processing in consistency-aware user's applications deployed on NoSQL databases," *Human Centric Computer Information System*, vol. 7, no. 7, pp. 2-18, 2017.

[25] S. L. R. C. a. O. H. S. P. Kumar, "Consistencylatency trade-off of the libre protocol: A detailed study," *Advances in Knowledge Discovery and Management*, vol. 7, pp. 83-108, 2018.

[26] L. L. a. S. S. E. Casalicchio, "Energy-aware autoscaling algorithms for cassandra virtual data centers," *Cluster Computing*, vol. 20, no. 3, pp. 2065-2082, 2017.

[27] G. Harrison, "Consistency models," in *Next Generation Databases: NoSQL, NewSQL and Big Data*, Berkeley, CA: Apress, 2015, pp. 127-144.

[28] ScyllaDB, "ScyllaDB Documentation - Consistency," 19 01 2023. [Online]. Available: https://docs.scylladb.com/stable/cql/consistency.html. [Accessed 19 02 2023].