The Journal of Systems and Software 208 (2024) 111885

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Performability evaluation of NoSQL-based storage systems[☆]

Carlos Araújo[a], Meuse Oliveira Jr.[b], Bruno Nogueira[c], Paulo Maciel[a], Eduardo Tavares[a,*]

[a] *Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil*
[b] *Instituto Federal de Pernambuco, Recife, Brazil*
[c] *Universidade Federal de Alagoas, Recife, Brazil*

ARTICLE INFO

Keywords:
Big data
Availability
Cloud storage
NoSQL
Stochastic petri nets

ABSTRACT

NoSQL database management systems (DBMS) have been commonly adopted in cloud storage applications, as they usually provide better performance and availability than relational DBMSs. Eventual consistency is a remarkable feature of many NoSQL database systems, in which not all redundant nodes have the newest data, but, eventually, such data will be present in all nodes. Distinct consistency levels can be utilized, but they may affect user experience and service level agreements. The assessment of consistency concerning the probability of accessing the newest data is not common, and the jointly evaluation of performance and availability is usually neglected. This work proposes a method based on stochastic Petri nets (SPN) for evaluating the consistency levels of storage systems based on NoSQL DBMS adopting quorum technique. The models take into account distinct consistency levels and redundant nodes for estimating system availability, throughput and the probability of accessing the newest data. Experimental results demonstrate the practical feasibility of our approach.

1. Introduction

NoSQL database management systems (DBMS) have been widely adopted due to the requirement of Big Data applications for dealing with huge volume of data. Indeed, NoSQL is an alternative to relational DBMS, as they allow flexible data model, horizontal scalability and absence of ACID transactions (Meier and Kaufmann, 2019). Such features allow better performance, and, consequently, prominent service providers (e.g., Amazon Tomar et al., 2019) have adopted NoSQL systems. Performance is also improved due to the adoption of eventual consistency in many NoSQL DBMSs. More specifically, a temporary inconsistency between the redundant servers (nodes) is accepted, but, eventually, all servers will be updated (Gomes et al., 2017). Different levels of consistency can be adopted to increase or decrease the probability of accessing the newest data.

Many applications rely on system accesses with low latency for meeting user expectations. Amazon achieved a 1% drop in sales due to an addition of 100 ms in response time, and Google recorded a 20% decrease in traffic (Bailis et al., 2014) due to an increment of 500 ms in its search engine response. In financial market, reducing milliseconds in trading delay can increase earnings by about 100 million per year (Tian et al., 2015). A low consistency level may considerably increase performance (i.e., low latency) and availability, as fewer servers need to be immediately updated. However, the assurance of obtaining the newest data for an operation (e.g., read) is considerably diminished. Such issues are also related to CAP theorem (Gilbert and Lynch, 2012), which states that, in a distributed system, it is not possible to simultaneously guarantee consistency, availability and partition tolerance (CAP).

In this context, performability models are an important tool for evaluating design decisions before implementing the real system (Maciel, 2023). Performability models contemplate the joint assessment of performance and dependability (Maciel, 2023; Trivedi and Bobbio, 2017), allowing the assessment of different consistency levels in the target system. However, most works overlook this assessment for NoSQL DBMS, and the probability of accessing the newest data is not commonly taken into account.

This paper proposes a performability method based on stochastic Petri nets (SPN) (Maciel, 2023) for evaluating distinct consistency levels in storage systems based on NoSQL DBMS adopting quorum technique. The method allows the assessment of operation latency, availability and the probability of accessing the newest data. Two SPN models are adopted. The first model assesses consistency levels

☆ Editor: Juan Carlos Duenas.
* Corresponding author.
E-mail addresses: cga@cin.ufpe.br (C. Araújo), meusejunior@recife.ifpe.edu.br (M. Oliveira Jr.), bruno@ic.ufal.br (B. Nogueira), prmm@cin.ufpe.br (P. Maciel), eagt@cin.ufpe.br (E. Tavares).
[1] http://cassandra.apache.org/.
[2] https://riak.com/products/riak-kv/.

https://doi.org/10.1016/j.jss.2023.111885
Received 16 March 2023; Received in revised form 9 October 2023; Accepted 24 October 2023
Available online 3 November 2023
0164-1212/© 2023 Elsevier Inc. All rights reserved.

on system latency and availability. The second SPN model evaluates data access, taking into account consistency level and the amount of redundant nodes (i.e., cluster size). These models are validated using Cassandra[1] and Riak KV DBMS,[2] and experimental results demonstrate the influence of consistency level on system behavior. A case study is also presented, which quantifies consistency and availability from the perspective of CAP theorem for system design.

The rest of this paper is organized as follows. Section 2 provides an overview of important concepts and Section 3 introduces related work. Section 4 presents the proposed models and Section 5 describes experimental results. Section 6 concludes this work.

2. Background

This section presents fundamental concepts to better understand this paper.

Consistency in NoSQL DBMS. Consistency in DBMS may be defined as the mechanism for storing and accessing data in all database instances (Mohamed et al., 2014). If the data retrieved from DBMS replicas is not the newest, the data is inconsistent. Thus, the DBMS needs to confirm that a write or read operation was executed on a subset of redundant servers, such that an accessed data is up-to-date. In addition to the influence on performance, availability may also be affected by the consistency policy, because, in a fault-tolerant DBMS, redundant nodes are usually present.

For greater performance and availability, NoSQL DBMSs usually adopt eventual consistency. More specifically, the redundant servers (replicas) may not immediately have the newest data, but, eventually, such data will be present in all nodes (Perkins et al., 2018). To mitigate this inconsistent interval, NoSQL DBMSs allow the adoption of different consistency levels, which define the minimum number of replicas that confirm an operation, which is tuned according to application requirements.

Availability may also be improved, since a DBMS requires the number of operational servers be equal to the consistency level. With a lower value, the database system needs less servers to enforce the consistency.

Fig. 1 presents a DBMS composed of 3 redundant servers and 3 different consistency levels. A server is responsible for receiving an operation request and, depending on the consistency level, it may also be responsible for coordinating the communication with other redundant nodes. This server is named as coordinator and any node can assume this function (which is dependent on the system load balancing) (Huang et al., 2017).

Assuming consistency level ONE, the confirmation of an operation (e.g., write) requires only a single server. For level TWO, two nodes must confirm the operation and, in the case of a read operation, the newest data of both servers are considered.

Similarly, level $THREE$ requires the confirmation of all nodes and, for a read operation, the newest data is guaranteed to be always accessible (strong consistency) (Harrison, 2015).

The condition for strong consistency is given by the following inequation:

$$(W + R > N) \tag{1}$$

N is the number of redundant servers; W is the number of servers with the newest data due to a write operation; and R is the number of adopted servers for a read operation.

Quorum-based consistency. Representative NoSQL DBMSs, such as Cassandra, Riak, Amazon DynamoDB, adopt a quorum-based technique for controlling database consistency between replicas. More specifically, a DBMS confirms an operation execution (e.g., write operation) if and only if a sufficient number of servers (quorum) recognize the successful completion of the operation (Gifford, 1979). Quorum is strongly related

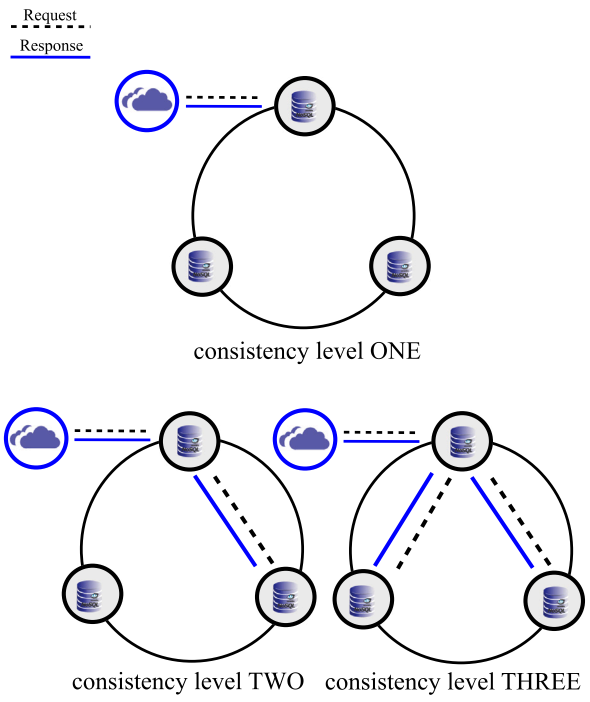

Fig. 1. Example of consistency levels (one, two and three).

to eventual consistency, since the adopted quorum is the consistency level of the operation.

Assume RF is the number of redundant servers (i.e., replication factor); R represents the read quorum; and W denotes the write quorum. The system is operational as long as the number of active servers is not less than $max(W, R)$. For instance, consider $RF = 3$, $R = 1$ and $W = 1$. To be operational, the storage system tolerates at most two simultaneous node failures.

CAP Theorem. CAP theorem defines that a distributed system can only provide two of the following features simultaneously (Gilbert and Lynch, 2012): consistency (C); availability (A); and partition tolerance (P). *Consistency* denotes all nodes have the same data version. In other words, the system will always return the newest data. *Availability* indicates every request will get a successful response. Even if some system nodes are not working, the system should still be capable to handle requests. *Partition tolerance* assumes the system must continue to function even when nodes are partially inaccessible. Partitions may occur, for instance, due to network problems or node failures.

When availability is prioritized, consistency may sacrificed (AP system). In a quorum-based NoSQL DBMS, the result may be outdated data in user requests. If the system prioritizes consistency, the application may be unavailable more often (CP system), and additional nodes may be required to meet a specified level of consistency (Brewer, 2017).

CAP theorem has a prominent demand for techniques to assess design trade-offs, taking into account previous properties (Gilbert and Lynch, 2012; Lee et al., 2021). The proposed models allow the jointly assessment of consistency, availability and partition tolerance, providing an additional tool for creating NoSQL-based storage systems.

Stochastic Petri nets - SPN. Petri nets are a family of formal models, which are suitable for representing systems with concurrency, synchronization, communication mechanism and probabilistic delays (Maciel, 2023).

A Petri net (PN) model is a bipartite directed graph, in which places (represented by circles) denote local states and transitions (depicted as rectangles) represent actions. Arcs (directed edges) connect places to transitions and vice-versa. An inhibitor arc is a arc type that depicts a small white circle in one edge, instead of an arrow, and they usually denote the unavailability of tokens in places. Tokens (small filled circles) may reside in places, which denote the state (i.e., marking) of a PN. The semantic of a PN is defined in terms of a token game, in the sense that tokens are created and consumed according to transition firings.

Particularly, this work adopts stochastic Petri nets (SPN), which is a prominent PN extension that allows the association of exponential distribution to timed transitions (represented by white rectangles), or zero delays to immediate transitions (depicted as thin black rectangles). Immediate transitions are a feature of generalized stochastic Petri nets (GSPN), but the term SPN is commonly utilized to refer to the class of stochastic nets.

Timed transitions may have different concurrency degrees: single server semantics (ss) or infinite server semantics (is). In general, is is adopted to represent parallel activities, considering that the firing rate of a transition is linearly increased according to its enabling degree. In ss, the firing rate is kept constant. For model evaluation, the state space of SPN models may be translated into continuous-time Markov chains (CTMC). The reader is referred to Maciel (2023) for more details.

3. Related work

Over the last decade, prominent works have been proposed to evaluate consistency and availability in NoSQL database management systems.

Bailis et al. (2014) propose the probabilistically bounded staleness (PBS) model, which estimates the probability to access the newest data after a write operation. Although the technique is a prominent contribution, the authors do not consider server failures. In Diogo et al. (2019), the authors evaluate the consistency approach of 5 NoSQL DBMSs: Redis, Cassandra, MongoDB, Neo4j and OrientDB. Although important information is provided regarding the consistency mechanism of each DBMS, availability is not quantitatively assessed. Burdakov et al. (2016) propose a modeling approach for assessing eventual consistency, taking into account the probability of a read operation not accessing the newest data. Experimental results show the model feasibility for evaluating consistency in replicated nodes, but availability is neglected.

Singla et al. (2018) present a technique to deal with data consistency in social networks. Experimental results indicate throughput can be improved, but they do not consider system failures. Krechowicz et al. (2021) propose a new data storage architecture to improve consistency and availability. The approach utilizes NoSQL DBMS, and the architecture is assessed using MongoDB and MemCached. Although this work presents performance results, availability and consistency are not quantified. In Klein et al. (2015), the authors carry out a performance evaluation of 3 NoSQL DBMS, more specifically, MongoDB, Cassandra and Riak. Results indicate Cassandra has the best throughput for the adopted workload, but the authors do not present availability results.

Rodrigues et al. (2019) evaluate the quality of service (QoS) of NoSQL DBMS taking into account redundancy mechanisms. The method utilizes generalized stochastic Petri nets (GSPN) for evaluating system performance (e.g. throughput) and availability. Results indicate downtime may considerably vary using distinct redundancy techniques (from 45 h to 229 hours per year). Nevertheless, nothing is stated about data consistency. Liu et al. (2015) propose a formal model for representing Cassandra DBMS, but performance and availability are not taken into account.

In Chihoub et al. (2015), the authors carry out experiments to assess the impact of consistency on energy consumption, using Cassandra as case study. Results indicate strong consistency provides the highest consumption, but energy savings can be obtained using lower consistency levels. In that work, the authors do not take into account availability. Osman and Piazzolla (2014) adopt queueing Petri nets to model Cassandra clusters, but focusing only on performance.

Unlike previous works, this paper proposes formal models to evaluate performance, availability and the probability of accessing the newest data in storage systems based on NoSQL DBMS with quorum technique. The proposed approach provides an additional tool for designers to assess distinct system configurations (e.g., CAP theorem) before implementing the real infrastructure.

4. SPN models

This section presents the conceived SPN models for evaluating NoSQL-based storage systems adopting quorum technique.

Two models are proposed. The first model, namely, performability model, takes into account the joint estimation of latency (i.e., time between successful operations) and availability (i.e., probability of a system being in operational state). This model allows the representation of redundant servers, distinct consistency levels and read or write operations. The second model, namely, data access model, evaluates the probability of accessing the newest data, taking into account the number of redundant nodes, write and read delays.

The following operators are adopted: $P\{exp\}$ estimates the probability of the inner expression (exp); $\#p$ denotes the number of tokens in place p; and $E\{\#p\}$ represents the average number of tokens in place p. Besides, the proposed models have been conceived for stationary analysis, and all timed transitions adopt exponential distribution and single-server semantics (except when indicated).

4.1. Performability model

Fig. 2 depicts the performability model, which is composed of 3 building blocks: availability, client arrival and consistency.

The availability block assumes N redundant servers, which are initially operational (place $pSrUp$). Transition $tFailure$ denotes the failure of a server, and its delay is represented by the server's mean time to failure (MTTF). This mean time considers all server components prone to fail, and the value can be obtained using, for instance, reliability block diagrams (RBD) (Maciel et al., 2011). Transition $tRestore$ represents the maintenance of a failed server (place $pSrDow$), and its delay is represented by the server's mean time to repair (MTTR). Such a value is defined by the system designer, and both $tFailure$ and $tRestore$ adopt infinite-server semantics. The model also adopts constant $CONS$, which is a positive integer denoting the consistency level. Immediate transitions $tInCoord$ and $tInQueue$ represent the failure of a read or write operation due to lack of servers to guarantee the consistency level. Inhibitor arcs perform this verification. Tokens in place $pRespFail$ indicate unsuccessful operations, and the firing of transition $tRespFail$ denotes a failure response. Arc weight ($CONS$) from $pRespFail$ to $tRespFail$ is utilized to convert the requests required by the consistency level into a single failure response.

Client arrival block represents the periodic user requests, which are indicated by the firing of timed transition $tClient$. Place $pClient$ represents the preparation of a new arrival. If there is space in the queue ($\#pQueue > 0$), a client request is accepted by the system (place $pCoord$). Place $pClClient$ denotes the state for checking such space.

In the consistency block, transition $tCoord$ represents the coordinator communication with the nodes (including itself) for executing an operation. An inhibitor arc with weight $(N-CONS)+1$ guarantees the transition will be only firable if there are enough operational servers to meet the consistency level. Whenever a server fails and the consistency level cannot be guaranteed, a failure response is returned (transition $tRespFail$). If the amount of DBMS nodes required by the consistency level performs the operation ($\#pSrvQuery = CONS$), a successful response is returned. In this case, a token is added in place $pRespSuc$, and the firing of transition $tRespSuc$ denotes such response.

The performability model is adopted to estimate availability and latency between successful operations. Availability is the readiness for the correct system functioning, and it is estimated as the probability of

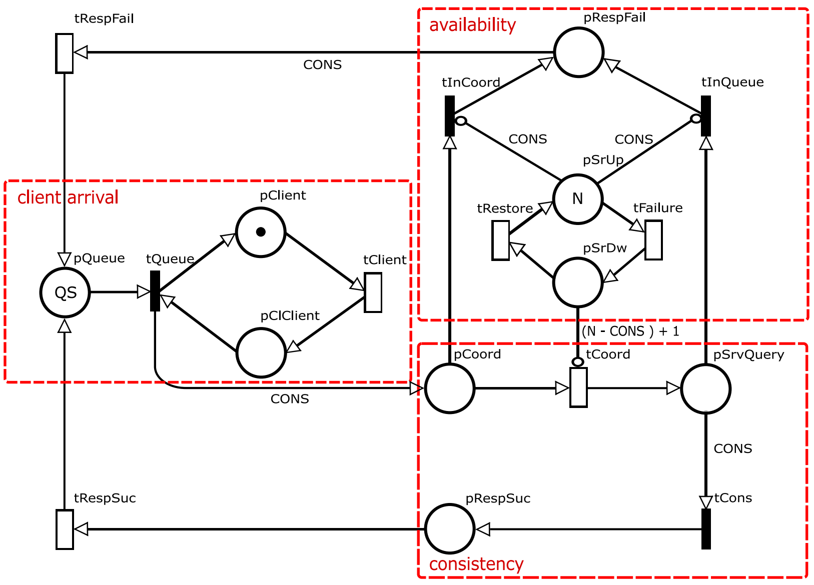

Fig. 2. Performability model.

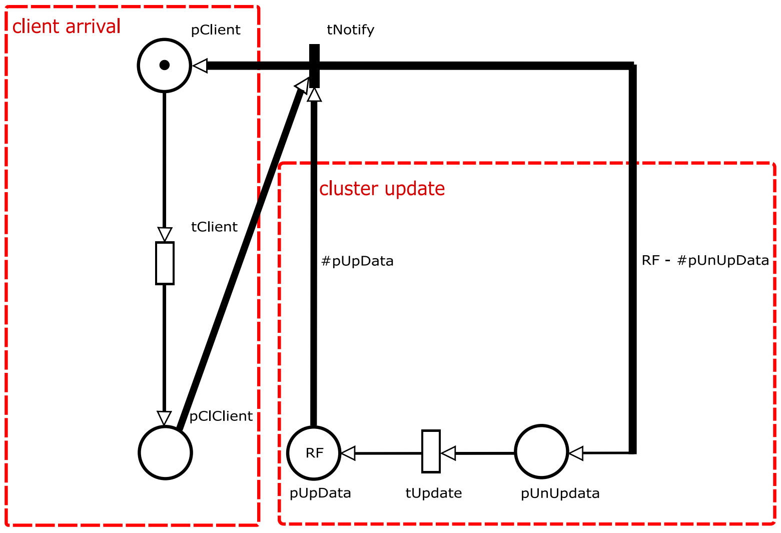

Fig. 3. Data access model.

the system being operational. Using the proposed model, availability is given by $A = P\{\#pSrUP \geq CONS\}$. Throughput is estimated as $throughput = P\{\#pRespSuc > 0\} * 1/resp$, in which $resp$ is the time associated to transition $tRespSuc$ (successful response). Thus, latency is $throughput^{-1}$.

4.2. Data access model

Fig. 3 depicts the SPN model for evaluating the probability of accessing the newest data in the database. The model assumes periodic write operations, which affects the consistency between the redundant servers (i.e., database replicas). A single write operation makes all servers temporarily inconsistents. The model adopts 2 constants: (i) RF denotes the number of database replicas; and (ii) $CONS$ represents the consistency level (quorum) of read operation. Besides, two building blocks are taken into account: client arrival and cluster update.

The client arrival block represents periodic write requests to the NoSQL cluster, and the firing of transition $tClient$ indicates the arrival of a request. The cluster update block depicts the execution of write operations in each server. Place $pUpData$ represents the nodes with the newest data, and its initial marking is equal to RF. Whenever a request is pending ($\#pClClient > 0$) and there are servers not performing a write operation ($\#pUpData > 0$), transition $tNotify$ fires. The marking of place $pUnUpData$ reflects the servers without the newest data. Besides, the actual execution of write operations is represented by timed transition $tUpdate$, which adopts infinite-server semantic

The model considers when a new data update is carried out ($\#pClClient > 0$), any ongoing update (still being replicated) becomes obsolete. Such an abstraction ensures the consistency state is always obtained with the newest data in all replicas. Besides, this abstraction enforces the quorum technique assumed in this work. Arc weight ($RF - \#pUnUpData$) from $tNotify$ to $pUnUpData$ guarantees this behavior, as data in all replicas will be out of date until the new update is carried out in RF replicas.

The probability for a single server to contain the newest data is $P_s = E\{\#pUpData\}/RF$, which is the same as the probability for a consistency level equal to 1 ($CONS = 1$). For $1 < CONS \leq RF$, the probability of accessing the newest data (P_{nd}) is the probability of accessing at least an updated server in the quorum: $P_{nd} = 1 - (1 - P_s)^{CONS}$.

5. Experimental results

This section presents experimental results to demonstrate the practical feasibility of the proposed modeling technique. Initially, model validation is presented and, next, assessments are carried out using a design of experiments (Montgomery, 2017) (DoE) approach. Finally, a case study is presented to demonstrate the design of a subsystem storage using CAP theorem. Besides, this work has adopted TimeNET tool (Zimmermann, 2017) for evaluating the proposed models.

5.1. Validation

The main idea is a comparison between the values obtained with baseline models and data collected from a real system, assuming the same parameters. Stationary analysis using numerical technique (Maciel et al., 2011) is adopted for each conceived model, and the respective result is compared to a 95% confidence interval estimated from system data.

In this context, we have adopted an environment based on XenServer 7.1 platform, considering 3 virtual machines (VM): one for the client and two for NoSQL DBMSs. Each VM utilizes Debian 9.0 operating system (OS) and a dual core CPU. The client VM has 4 GB of RAM and each database server has 8 GB. Besides, the validation has adopted Cassandra 2.1 and Riak KV 2.2. Such DBMSs have been adopted, as they utilize the quorum technique (Section 2) and demonstrate the feasibility of our approach for distinct NoSQL DBMSs.

For the sake of validation, this work utilizes a replication factor with 2 machines and consistency level 1 for Cassandra DBMS and consistency level 2 for Riak KV. Such levels represent the default configuration for these DBMSs. We have not considered server failures for performance and read operation has been taken into account.

Regarding performance, both Cassandra and Riak are considered. For availability and data access model, only Cassandra is taken into account, as Riak provides similar behavior.

5.1.1. Performability model - performance

Yahoo! Cloud Serving Benchmark (YCSB) (Cooper, 2022) has been utilized, and the following parameter values have been adopted: (i) size of a single record (*data size*) - 1 kB; (ii) probability distribution for the workload (*request distribution*) - uniform; (iii) number of records in the database (*record count*) - 200 000; (iv) amount of operations for a YCSB execution (*operation count*) - 30 000; and (v) always accesses all fields of a record (*read all fields*) - $true$. The time between requests (transition $tClient$) was set to 12.4 ms on Cassandra and 11.8 ms on Riak KV, and 30 samples were collected on each DBMS server. For Cassandra, the delays estimated for transitions $tCoord$ and $tRespSuc$ are 10.24 ms and 2.6 ms, respectively. For Riak, the mean delays are 3.49 ms ($tCoord$) and 1.8 ms ($tRespSuc$).

Concerning Cassandra, the latency between successful responses was estimated as 18.454 ms using the proposed model, and this value is contained in the 95% confidence interval [18.391, 19.315] obtained with measurements. Regarding Riak KV, the latency was estimated as 13.928 ms, and this value is also contained in the 95% confidence interval [13.912, 14.232] collected from the real system. Therefore, there is no statistical evidence to indicate the models do not represent the behavior of the adopted systems.

5.1.2. Performability model - availability

We considered the validation of availability block depicted in Fig. 2 using a fault injection technique, taking into account one Cassandra DBMS instance. 0.82 h is assumed for MTTR and 360 h is adopted for MTTF, and they were estimated using the values from Melo et al. (2017). Besides, this work assumed the period of one year, and, for the sake of feasibility, we performed an acceleration factor of 100. In this case, the experiment was carried out for almost 97 h, considering the reduction of MTTF to 3.6 h and MTTR to 0.0082 h.

This work adopted a script, which generated a failure followed by a repair. The failure is represented by stopping the network interface for communicating with the DBMS, and the maintenance restores the interface operation. Firstly, a time to failure is generated using the exponential distribution. A delay occurs, and the network interface is shutdown. Next, a time to repair is obtained using also the exponential distribution, a delay is carried out, and, then, the network interface is restored. The script was kept in a loop, and any change in the state of the DBMS was recorded.

We utilized the technique described in Keesee (1965) for calculating a 95% confidence interval for system availability. The model estimated 0.997727 for availability, which is contained in the confidence interval $[0.997319, 0.998306]$. Thus, there is no statistical evidence to refute the equivalence between the proposed model and the real system.

5.1.3. Data access model

For this model, YCSB is not feasible, since this tool does not have features for evaluating access to the newest data. Thus, we developed a script based on two threads: (i) one thread generates periodic write operations to each server; and (ii) a second thread simultaneously carries out read operations to assess the probability of accessing the newest data.

The time between write operations (transition $tClient$) was set to 100 ms and 30 samples were collected from the servers. Using these values, the estimated mean delay for transition $tUpdate$ is 2.639 ms. The model was then evaluated, and the probability of accessing the newest data is 0.974. As the model estimate is contained in the obtained 95% confidence interval [0.974, 0.976], there is no statistical evidence to reject the equivalence between the model and the real system.

5.2. Experiments

Three experiments were carried out to evaluate latency, availability and probability of accessing the newest data. These experiments are based on a design of experiments (DoE) (Montgomery, 2017) approach with a l^k factorial design.

The following sections present results using rank of effects. Effect is the change in response due to a change in the factor level. All ranks are presented in descending order, taking into account the absolute values of all effects. Besides, this work considers main effects and second-order interactions, since higher order interactions are usually negligible (Montgomery, 2017).

5.2.1. Latency

This experiment considers 4 factors ($k = 4$) with 2 levels ($l = 2$) for each operation: (i) consistency level ($cons$) - 1, 3; (ii) successful response time ($resp$) - 1.3, 2.6; (iii) coordinator communication (cc) - 5.1, 10.2; and (iv) replication factor (rf) - 3, 5. For these experiments, MTTF and MTTR of each server have been kept constant, and their values are 2880 h and 1 h, respectively. We also kept constant the time between client requests: 12.4 ms.

The levels are the same as the values utilized in the validation with Cassandra DBMS and a 50% reduction is taken to account to denote a better system capacity. $cons$ and rf assess the influence of consistency level and number of servers on system performance. The levels for $cons$ allow the evaluation of eventual and strong consistency. For RF, 3

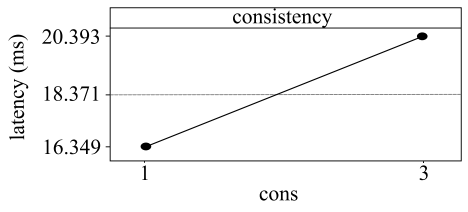

Fig. 4. Effect plot - latency.

Table 1
Rank of main and interaction effects (latency).

Factor/interaction	|Effect| (ms)
cc	5.231
$cons$	4.044
$cc * cons$	1.848
$resp$	0.795
$cons * resp$	0.106
$cc * resp$	0.090

Table 2
Results for read latency.

cons	resp (ms)	cc (ms)	rf	Latency (ms)
1	1.3	5.1	5	14.352
1	1.3	5.1	3	14.352
1	2.6	5.1	5	14.963
1	2.6	5.1	3	14.963
3	1.3	5.1	5	16.450
3	1.3	5.1	3	16.459
3	2.6	5.1	5	17.248
3	2.6	5.1	3	17.257
1	1.3	10.2	5	17.657
1	1.3	10.2	3	17.657
1	2.6	10.2	5	18.424
1	2.6	10.2	3	18.424
3	1.3	10.2	5	23.424
3	1.3	10.2	3	23.436
3	2.6	10.2	5	24.427
3	2.6	10.2	3	24.440

replicas meet the highest level for $cons$, and 5 replicas permit a better assessment of effects regarding more available resources.

Table 1 presents the rank for main and interaction (e.g., $cons * cc$) effects for latency between successful operations. Coordinator communication (cc) has the greatest effect. Consistency level ($cons$) also has an important contribution to performance, as the amount of servers required for an operation has a prominent influence on system performance. Due to the importance of cc and $cons$, their interaction ($cc * cons$) has the largest effect among all interactions.

Replication factor (rf) and its interactions do not play a significant role in determining system performance. However, redundancy is important for system availability, which also depends on the required consistency level (see next experiment).

Table 2 presents the successful latency for each DoE treatment, considering an ascending order. The lowest values are associated with consistency 1, since less communication between nodes is required for executing an operation. For a better visualization, Fig. 4 depicts an effect plot indicating the change of performance for each consistency level.

Reduced coordinator communication and response times improve latency and, for the adopted levels, a system designer should assess the procurement of better machine components and network capacity.

5.2.2. Availability

The second experiment takes into account the following factors ($k = 4$) and levels ($l = 2$): (i) consistency level ($cons$) - 1, 3; (ii) replication

Table 3
Rank of main and interaction effects (downtime).

Factor/interaction	|Effect| (h)
$cons$	2.565
rf	−2.565
$cons * rf$	−2.565
$restore$	0.855
$cons * restore$	0.855
$failure$	−0.855
$cons * failure$	−0.855
$rf * failure$	0.855
$rf * restore$	−0.855
$restore * failure$	−0.285

Table 4
Results for availability.

$cons$	rf	$restore$ (h)	$failure$ (h)	Availability
1	5	0.5	5760	>0.999999
1	5	0.5	2880	>0.999999
1	5	1	5760	>0.999999
1	5	1	2880	>0.999999
1	3	0.5	5760	>0.999999
1	3	0.5	2880	>0.999999
1	3	1	5760	>0.999999
3	5	0.5	5760	>0.999999
1	3	1	2880	>0.999999
3	5	1	5760	>0.999999
3	5	0.5	2880	>0.999999
3	5	1	2880	>0.999999
3	3	0.5	5760	0.999739
3	3	0.5	2880	0.999479
3	3	1	5760	0.999479
3	3	1	2880	0.998959

factor (rf) - 3, 5; (iii) server MTTR ($repair$) - 0.5 h, 1 h; (iv) server MTTF ($failure$) - 2880 h, 5760 h. $failure$ and $repair$ are related to the server and their values may vary due to better hardware components or skilled maintenance crew. Due to the absence of studies that evaluate availability of NoSQL DBMSs, we adopted values associated with virtual machines (Melo et al., 2017) and increased by 100% for the highest level.

Table 3 presents the rank of effects using downtime (DT). The latter is utilized for presenting the rank of effects, since downtime provides a better visualization for small values related to availability variation. Assuming one year as the time span, downtime (in hours) is $DT = (1 - A) * 8760$.

The factors $cons$, rf and their interaction are responsible for most availability variation, more than 2 h a year, considering the adopted levels. Although 2 h seem small compared to 8760 h, it may generate fines due to violations of quality of service agreements or loss of users. MTTF ($failure$) and MTTR ($restore$) also contribute to system availability. Nevertheless, the interaction of MTTF and MTTR does not have a significant influence on availability, considering the adopted factors and levels.

Table 4 provides the results of each treatment. Availability is only considerably reduced when strong consistency ($cons = rf$) is taken into account, in which no redundant machine is accessible for fast replacement of a failed node. Besides, Fig. 5 depicts an effect plot indicating the change of availability for the consistency levels, in which downtime may vary from less than one minute per year (level 1) to 2.540 h (level 3).

5.2.3. Probability of accessing the newest data

For this experiment, this work adopts four factors ($k = 4$) with 2 levels ($l = 2$): (i) consistency level ($cons$) - 1, 3; (ii) replication factor (rf) - 3, 5; (iii) time between client requests ($arrival$) - 100 ms, 200 ms; and (iv) write delay ($update$) - 2.5 ms, 5 ms. The lowest level for RF is 3, since, with a single server, the probability of accessing the newest data

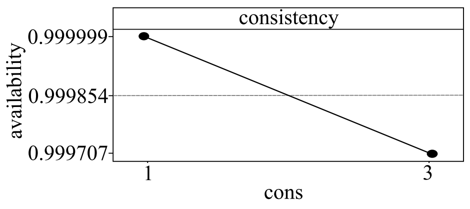

Fig. 5. Effect plot - availability.

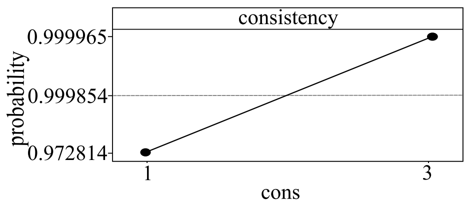

Fig. 6. Effect plot - data access.

Table 5
Rank of main and interaction effects (probability).

Factor/interaction	|Effect|
cons	0.02715
update	−0.00884
arrival	0.00884
cons ∗ update	−0.00879
cons ∗ arrival	0.00879
arrival ∗ update	−0.00281

Table 6
Results for probability.

cons	*rf*	*arrival* (ms)	*update* (ms)	Probability
1	3	100	5	0.952380
1	5	100	5	0.952380
1	3	100	2.5	0.975609
1	5	100	2.5	0.975609
1	3	200	5	0.975609
1	5	200	5	0.975609
1	3	200	2.5	0.987654
1	5	200	2.5	0.987654
3	3	100	5	0.999892
3	5	100	5	0.999892
3	3	100	2.5	0.999985
3	5	100	2.5	0.999985
3	3	200	5	0.999985
3	5	200	5	0.999985
3	3	200	2.5	0.999998
3	5	200	2.5	0.999998

is always 1 (despite its impact on availability). The values for *arrival* represent distinct client demand, and *update* may vary due to better hardware capacity or technology. Both have a 100% increment for the highest level.

Table 5 depicts the rank. *cons* has the greatest influence on probability, in the sense a higher level increases the chance of getting the most recent data. For a better visualization, Fig. 6 shows an effect graph indicating the impact of probability due to the adopted levels for *cons*.

arrival and *update* have the same impact on data access. A higher inter-arrival time increases the probability, but a slower write operation decreases the probability of accessing the newest data. As *cons* is a remarkable factor, *cons ∗ arrival* and *cons ∗ update* are the most important interactions. *rf* is not present in the ranks, since the respective influence is negligible. However, *rf* is implicitly associated with the consistency level *cons*.

Table 6 shows the probability for each DoE treatment. Higher values are associated with consistency level 3, since more servers are adopted for read operation.

Eventual consistency is an important feature of NoSQL DBMS and it has a major impact on availability, performance and probability of accessing the newest data. A suitable consistency level is a design decision. The proposed models can aid designers to evaluate consistency levels on distinct hardware infrastructures.

5.3. Case study

This section presents a case study for assessing distinct designs for a data storage system, taking into account quorum-based NoSQL DBMS and CAP theorem. Similar to previous experiments, *rf* is the replication factor, and *cons* denotes the consistency level for the read operation.

Table 7
Results for case study.

cons	*rf*	Arrival (ms)	C	A	P
1	3	100	2439.02	>0.99999999	2
2	3	100	59.49	0.99999985	1
3	3	100	1.45	0.99896018	0
1	5	100	2439.02	>0.99999999	4
2	5	100	59.49	>0.99999999	3
3	5	100	1.45	>0.99999999	2
1	3	500	497.51	>0.99999999	2
2	3	500	2.48	0.99999985	1
3	3	500	0.01	0.99896018	0
1	5	500	497.51	>0.99999999	4
2	5	500	2.48	>0.99999999	3
3	5	500	0.01	>0.99999999	2

System consistency (C) is represented as the number of inconsistent data per 100,000 data accesses, which is estimated using the probability of accessing the newest data (P_{nd} - Section 4.2) multiplied by 100,000. Availability (A) is obtained using the performability model (Section 4.1), and partition tolerance (P) is the maximum number of nodes that may become unavailable: *rf* - *cons*.

The values for tRestore (MTTR) and tFailure (MTTF) are 1 h and 2880 h, respectively. The levels for replication factor are 3 and 5, and, for the consistency level, the values are 1, 2 and 3. The periodic write requests (*arrival*) associated with transition *tClient* (Section 4.2) are 100 ms and 500 ms. These values are taken into account to demonstrate the influence on system consistency (C) considering distinct write workloads. The actual execution of write operations (*tUpdate*) is kept at 2.5 ms. Table 7 presents the results.

System consistency (C) demonstrates the smallest accesses to inconsistent data are related to consistency level 3. More specifically, only 0.01 and 1.4 inconsistent data accesses per 100,000 read requests are obtained. However, consistency level 2 also obtains prominent results, assuming a longer interval between write operations (500 ms): 2.6 accesses to obsolete data. For level 1, data inconsistency may vary between 517 to 2534 accesses.

Availability (A) is significantly degraded at consistency level 3, which generates more than 9 h per year of downtime ($DT = (1 - A) * 8760$). For $cons = 2$, the value drops to approximately 5 min. For consistency level 1, the system will be unavailable for less than 1 s per year.

Partition tolerance (P) may admit up to 4 node failures. All treatments with P greater or equal to 2 have less than 1 s per year of downtime. Nevertheless, the consistency is significantly impacted whenever write requests increase (100 ms). Results indicate a relationship between P and A. However, P and C depend on other system parameters, such as frequency of write operations.

6. Conclusion

NoSQL DBMSs adopt eventual consistency to mitigate performance issues to manipulate large amount of data. DBMS consistency may also affect other metrics, such as availability, and, thus, defining a suitable consistency level may not be an easy task. Furthermore, CAP theorem provides additional challenges concerning design trade-offs between consistency, availability and partition tolerance.

This paper presented analytical models based on SPN for evaluating storage systems adopting quorum-based NoSQL DBMS. The models estimate system availability, latency and the probability of accessing the newest data. Experimental results demonstrate consistency is a prominent factor, which cannot be neglected in the design of storage systems. As future work, we are devising techniques to assess the impact of consistency on energy consumption.

CRediT authorship contribution statement

Carlos Araújo: Conceptualization, Formal analysis, Validation, Data curation, Writing – original draft. **Meuse Oliveira Jr.:** Conceptualization, Investigation, Validation, Writing – review & editing. **Bruno Nogueira:** Investigation, Validation, Data curation, Writing – review & editing. **Paulo Maciel:** Investigation, Validation, Formal analysis, Writing – review & editing. **Eduardo Tavares:** Conceptualization, Formal analysis, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work has been supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq under grant 302997/2021-0.

References

Bailis, P., Venkataraman, S., Franklin, M., Hellerstein, J., Stoica, I., 2014. Quantifying eventual consistency with PBS. VLDB J. 23 (2), 279–302.
Brewer, E., 2017. Spanner, truetime and the cap theorem.
Burdakov, A., Grigorev, U., Ploutenko, A., Ttsviashchenko, E., 2016. Estimation models for NoSQL database consistency characteristics. In: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP).
Chihoub, H.E., Ibrahim, S., Li, Y., Antoniu, G., Pérez, M., Bougé, L., 2015. Exploring energy-consistency trade-offs in cassandra cloud storage system. In: 2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD).
Cooper, B., 2022. Yahoo! cloud serving benchmark. https://github.com/brianfrankcooper/YCSB, Accessed: 2022-09-09.
Diogo, M., Cabral, B., Bernardino, J., 2019. Consistency models of NoSQL databases. Future Internet 11 (2), 43.
Gifford, D.K., 1979. Weighted voting for replicated data. In: Proceedings of the Seventh ACM Symposium on Operating Systems Principles. ACM, pp. 150–162.
Gilbert, S., Lynch, N., 2012. Perspectives on the CAP Theorem. Computer 45 (2), 30–36.
Gomes, V., Kleppmann, M., Mulligan, D., Beresford, A., 2017. Verifying strong eventual consistency in distributed systems. Proc. ACM Program. Lang. 1 (OOPSLA), 109:1–109:28.
Harrison, G., 2015. Consistency models. In: Next Generation Databases: NoSQL, NewSQL, and Big Data. A Press, pp. 127–144.
Huang, X., Wang, J., Yu, P.S., Bai, J., Zhang, J., 2017. An experimental study on tuning the consistency of NoSQL systems. Concurr. Comput.: Pract. Exper. 29 (12).
Keesee, W., 1965. A Method of Determining a Confidence Interval for Availability. Technical Report, Naval Missile Center Point.
Klein, J., Gorton, I., Ernst, N., Donohoe, P., Pham, K., Matser, C., 2015. Performance evaluation of NoSQL databases: A case study. In: Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems.
Krechowicz, A., Deniziak, S., Łukawski, G., 2021. Highly scalable distributed architecture for NoSQL datastore supporting strong consistency. IEEE Access 9, 69027–69043.
Lee, E.A., Bateni, S., Lin, S., Lohstroh, M., Menard, C., 2021. Quantifying and generalizing the CAP theorem. arXiv preprint arXiv:2109.07771.
Liu, S., Nguyen, S., Ganhotra, J., Rahman, M., Gupta, I., Meseguer, J., 2015. Quantitative analysis of consistency in NoSQL key-value stores. In: Quantitative Evaluation of Systems. Springer, pp. 228–243.
Maciel, P., 2023. Performance, Reliability, and Availability Evaluation of Computational Systems, first ed. CRC Press.
Maciel, P., Trivedi, K., Matias, R., Kim, D., 2011. Dependability modeling, pp. 53–97. chapter 3.
Meier, A., Kaufmann, M., 2019. Nosql databases. In: SQL & NoSQL Databases. Springer, pp. 201–218.
Melo, C., Matos, R., Dantas, J., Maciel, P., 2017. Capacity-oriented availability model for resources estimation on private cloud infrastructure. In: 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC).
Mohamed, M.A., Altrafi, O.G., Ismail, M.O., 2014. Relational vs. nosql databases: A survey. Int. J. Comput. Inform. Technol. 3 (03), 598–601.
Montgomery, D.C., 2017. Design and Analysis of Experiments, ninth ed. John wiley & sons.
Osman, R., Piazzolla, P., 2014. Modelling replication in NoSQL datastores. In: Norman, G., Sanders, W. (Eds.), Quantitative Evaluation of Systems: 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings. Springer International Publishing, Cham, pp. 194–209. http://dx.doi.org/10.1007/978-3-319-10696-0_16.
Perkins, L., Redmond, E., Wilson, J., 2018. Seven Databases in Seven Weeks: A Guide To Modern Databases and the NoSQL Movement. Pragmatic Bookshelf.
Rodrigues, M., Vasconcelos, B., Gomes, C., Tavares, E., 2019. Evaluation of NoSQL DBMS in private cloud environment: An approach based on stochastic modeling. In: 2019 IEEE International Systems Conference (SysCon). pp. 1–7.
Singla, P., Singh, S., Gopinath, K., Sarangi, S., 2018. Probabilistic sequential consistency in social networks. In: 2018 IEEE 25th International Conference on High Performance Computing (HiPC). pp. 102–111.
Tian, X., Han, R., Wang, L., Lu, G., Zhan, J., 2015. Latency critical big data computing in finance. J. Finance Data Sci. 1 (1), 33–41.
Tomar, D., Bhati, J., Tomar, P., Kaur, G., 2019. Migration of healthcare relational database to NoSQL cloud database for healthcare analytics and management. In: Healthcare Data Analytics and Management. Elsevier, pp. 59–87.
Trivedi, K., Bobbio, A., 2017. Reliability and Availability Engineering: Modeling, Analysis, and Applications, first ed. Cambridge University Press.
Zimmermann, A., 2017. Modelling and performance evaluation with TimeNET 4.4. In: International Conference on Quantitative Evaluation of Systems. Springer, pp. 300–303.