Literature Review
Automatically Measuring Consistency Properties of MongoDB

Julia Wong
SID 430131851
Supervisor: Alan Fekete

May 10, 2017

1 Introduction

The exponential growth of web applications and its dependency on data has driven the widespread adoption of NoSQL databases over traditional relational databases. This is due to attractive features offered such as high availability and low latency. However, this is achieved at the cost of weaker consistency properties. This literature review involves the appraisal of consistency models and properties, existing benchmarks and metrics to measure properties of NoSQL systems and MongoDB[1], a well known NoSQL database. Whilst these areas have been presented in a variety of contexts, this paper will primarily focus on their application to distributed database systems.

2 Consistency

Consistency is a property that is critical to all databases regardless of whether they are NoSQL or relational. Literature has defined consistency in different manners based on the context of the database system used and is commonly viewed from two perspectives - *client-centric* which focuses on the clients perspective of consistency guarantees and *data-centric* which takes the perspective of the storage provider and focuses on the internal state of the database [1–3]. With the rise of distributed systems and NoSQL, we find that transactional consistency is no longer applicable and thus a variety of new models for atomic consistency have arisen.

2.1 Transactional Consistency

Relational databases require for all interactions to be executed indivisibly through the concept of a transaction. Haerder and Reuter define a terminological framework for describing transactional databases. The most notable contribution of

[1] https://www.mongodb.com

their paper is the identification of four essential properties of database transactions known as ACID. These are: atomicity, consistency, isolation and durability [4]. The consistency property aims to ensure data integrity and completeness through the requirement that the transaction must only commit legal results that leave the database in a valid state. It should be noted that the consistency property in ACID is specific to databases in the transaction-paradigm and hence is not applicable to NoSQL databases [1, 5]. NoSQL databases also don't offer the level of reliability ACID offers [6]. These limitations have motivated the development of models that redefine the consistency property for distributed systems.

2.2 Item Consistency

2.2.1 BASE

The drastic increase in web applications has seen distributed databases drop strong consistency guarantees, seen in relational databases, with weaker consistency models such as *eventual consistency* [7]. Terry defines eventual consistency from a data-centric perspective: "servers converge towards identical database copies on the absence of updates" [8], whilst Vogels provides a similar definition but with a client-centric perspective: "if no new updates are made to the object, eventually all accesses will return the latest value" [1]. In reaction to the increase in adoption of high availability systems and emergence of distributed databases with eventual consistency guarantees, Brewer created the BASE design philosophy - Basically Available, Soft State, Eventually consistent [9]. BASE can be viewed as the direct opposite of ACID due to its optimism in allowing for leniency with database consistency [10]. The BASE design philosophy has allowed for an effective representation of the movement to focus on availability at the expense of consistency .

2.2.2 CAP

Not all distributed database systems guarantee eventual consistency, however they will all however encounter similar trade offs. The CAP theorem is less specific than BASE but has a wider scope to cater for all distributed systems. CAP was first presented by Brewer [11] as a conjecture and was later proved and formalised by Gilbert and Lynch [5]. The CAP theorem states that a web service, at any given time, can only provide two of the three guarantees of: consistency, availability and partition-tolerance. CAP defines consistency as reading the most recent write. However, in a distributed system we cannot sacrifice partition-tolerance which leaves a trade-off between availability and consistency [1, 7, 12, 13]. Sacrificing consistency over availability is at the cost of user experience [7, 13], especially in latency-sensitive applications [14]. Similarly, [2] claims that "As long as no stale data is observed, the customer is satisfied". Whilst CAP is frequently mentioned in literature relating to distributed databases, it is increasingly misunderstood that CAP only imposes restrictions when the system is faced with failures [12]. Furthermore, Brewer later reflects on CAP after twelve years and admits that CAP is misleading due to the oversimplification of the relations between properties which assume that when faced with network partitions, the property chosen is perfect and the

other cannot be achieved at all. [9]. Whilst CAP is general enough to be able to effectively represent trade offs faced in distributed systems, it does not address latency, a critical property which is essential in web applications.

2.2.3 PACELC

Latency, a property closely related to partition-tolerance, is essential to consider in online applications. CAP fails to address the issue of latency and Abadi (2012) argues that the tradeoff between consistency and latency "has been more influential on DDBS design than CAP tradeoffs" [12]. This leads to the contribution of PACELC, a new theorem for distributed systems which unifies CAP with the trade offs between consistency and latency. The theorem states that if there is a network partition (P) then the system must trade off between availability (A) and consistency (C). Otherwise (E), when there is an absence of partitions, the system must trade off latency (L) and consistency (C). This new model is advantageous over CAP as it is considers scenarios where there are partitions and an absence of them which allows one to gain a greater and more complete portrayal of consistency trade offs in a distributed database.

2.3 Properties

Consistency from a client-side perspective can be broken down into a variety of fine-grained models and properties. This greatly assists customers in understanding the level of consistency a service may be providing and allow for companies to be held accountable if these levels are not met. Vogels offers insight into principles and abstractions, relating to eventual consistency, that have assisted Amazon in delivering reliable, global distributed systems [1]. Terry et al. provide insight into weak consistency guarantees that were developed for their Bayou project [8] and give various examples where each would be used. [15] We draw on both papers to define consistency at various layers and restrict our scope to *read your writes* and *monotonic reads* guarantees.

Eventual consistency is a form of weak consistency that was previously defined in section 2.2. Weak consistency is a very coarse grained level of consistency that can be viewed as the polar opposite of strong consistency. It does not guarantee that the latest read will return the latest write [1]. However, it does allow for reads and writes to be performed with significantly less synchronization than would be required with strong consistency [8]. Eventual consistency falls under the umbrella of weak consistency due to it relying on writes converging at a later point in time.

2.3.1 Read Your Writes

The read your writes consistency guarantee requires read operations to reflect the most previously updated value. This guarantee is motivated by user experience to ensure that the effects of writes are able to be read in the same session [8].

2.3.2 Monotonic Reads

The monotonic reads consistency guarantees that reads can never return older values in subsequent reads and will always return a non-decreasing set of writes.

A database using this guarantee appears to the user as one that is being increasing updated over time [8].

3 Benchmarks

Consistency properties allow for a more granular definition of consistency and can assist a service owner in defining what level of consistency their service will provide to users. However, in order to effectively measure consistency properties across various distributed systems, we must define a common benchmark to assist in fairly quantifying metrics [2, 16–18].

Both Rahman et al. [17] and Bermbach et al. [16] clearly state the need for a benchmark for eventually consistent systems and work towards defining a benchmark to quantify consistency guarantees. [16] aims for their paper to be a "call for action" to motivate others to contribute to their efforts in defining a comprehensive consistency benchmark. Whilst there have been previous efforts to empirically measure consistency, many have shortcomings which affect the accuracy of results [17]. Both papers provide detail into their vision of a common benchmark and verify their ideas by implementing a minimal working version that is run against a well known NoSQL database. Only Δ-atomicity is proposed and implemented as the metric to use to measure consistency in [17] despite identifying numerous errors with other approaches such as disruption of workload and pattern of failures. The contribution from Rahman et al. appears minimal in comparison to Bermbach et al. who identify various metrics to use, consider issues such as geo-distribution and multi-tenancy and give a thorough overview of the architecture of their proposed benchmark. [17] also take a client-centric perspective and use coarse-grained metrics. [16] is also critical of their experiment as it is highly dependent on a client workload which may be difficult to reproduce. Unlike [16], [17] fail to explain how their benchmark could be used in application development. Both papers have recommended that fault injection tools be supported in future benchmarks and [16] has identified Simian Army[2] as a potential option.

The *Yahoo! Cloud Serving Benchmark* (YCSB) is a very well known benchmark that allows for the comparison of performance between cloud services [18]. It was made open source to encourage the contribution from others. The paper claims that a key feature of YCSB is its extensibility and describe in detail how one could extend YCSB and provide a new set of workloads. Despite the benchmark only measuring performance, the long term goal of the YCSB is to provide a standard benchmark to assist in the evaluation of various systems across various criteria including consistency. The YCSB framework consists of a workload generator and a package of standad workloads to cover performance. [16] incorporates YCSB as their workload generator for their experiment to evaluate the results of their benchmark on Cassandra and MongoDB. They looked into the effects of the workload and effects of geo-distribution. The most surprising result was MongoDB's reaction to geo-distribution - the single availability zone took longer than the multiple availability zone experiment. A faulty replica and a clock synchronization issue are what Bermbach et al. suspect is causing this result. [17] also uses YCSB for their benchmarking experiment on Cassandra and

[2]https://github.com/Netflix/SimianArmy is an open source suite of tools. One of these tools is Chaos Monkey, a tool that will insert and simulate errors.

measured the staleness of data and observed results that indicated that there were some inconsistencies which result in higher staleness for certain periods of time.

3.0.1 Algorithmic and Probabilistic Approaches

Similar to [17], Anderson [13] takes a client-centric approach to consistency. However, they aim to quantify violations through an analysis of the trace of the database system. They use an algorithm to construct a precedence graph and have a set of rules for adding edges. These rules differ based on the level of consistency tested for. A depth first search is then run to determine if the graph is acyclic. The number of cycles found represents the number of violations of consistency the system has. However, [13] points out that edges may be accounted for more than once and so their method provides an upper bound. The runtime of this process is a drawback as it can only be used offline otherwise it may impact on system performance. Anderson also assumes failure-free executions which doesn't allow for a complex enough scenario to be tested. They also suggest that NTP be used to achieve sufficient synchronization in future however it should be noted that [2] warns that NTP have accuracy limitations.

Bailis et al. [19] present a probabilistic approach to provide an expectation of recency for reads of data items. However, their approach is only able to be applied to quorum replicated data stores. Their probabilistic model was verified through an experiment which compared the predicted staleness with the observed consistency. The experiment proved that eventually consistent stores are often faster than their strongly consistent counterparts. However, like [13], Bailis' approach assumes a failure-free execution and is unable to model network partitions. This approach also does not allow for the modeling of workloads whcih overlap with reads [14].

Zellag and Kemme [20] also attempt to quantify the number of inconsistencies when an application is running under a certain level of consistency. They focus on quantifying anomalies like [13] and also create a dependency graph, however their approach allows for the algorithm to run online. Having the algorithm run at the application layer, independent of the data store is the main advantage of Zellag and Kemme's approach over Anderson's approach. However, their approach assumes a causally consistent datastore which is not reflective of today's solutions which rely on eventual consistency [16].

3.0.2 Metrics

Systems that adopt eventual consistency should be aware that data staleness may occur due to the lack of guarantee that reads will immediately see the updated values. Golab et al. [14] explores methods of quantifying eventual consistency for comparisons between eventually consistent data stores and focuses on data staleness in particular. They propose that k atomicity and Δ-atomicity be used to measure version and time based staleness respectively. Both of these measures were suggested in [16] outline of metrics for a standard consistency benchmark whilst [17] only identified Δ-atomicity. [21] suggests that future research should attempt to solve the challenge of consistency verification. Zhu and Wang have achieved this by providing a formal definition and proof for client-centric consistencies [15]. Similar to [8] they focus their study on consistency

within a session which limits the scope to a single user operation.

Wada et al. [22] assess consistency from a client-centric approach by measuring data staleness with Δ-atomicity. Their approach uses a writer to repeatedly write the time into a data element and a reader that will read the contents from the data element 50 times a second and record how long this takes. This experiment was repeated every hour for a week. Results showed that SimpleDB does not support monotonic reads or read your writes consistency guarantees. Tests were also run on Amazon S3 which showed that inconsistency may only be visible in the event of a failure. [2] extends the work done by [22] by using multiple readers as using a single reader was identified as a weakness due to it it not being representative of a normal NoSQL database which is likely to have multiple users attempting to read at the same time. Having multiple readers also allows for the effects of geo distribution to be observed as well as latency. Interestingly, [2] reports contradicting results for their S3 experiment as they find that there are frequent violations of monotonic read consistency. They also observe alternating periodicities in data staleness and guess that it is due to the implementation of S3. A weakness of this approach is that it is disruptive to the workload and is not suitable for use in a production environment. [17] identified this as a weakness of many attempts to empirically study consistency.

4 MongoDB

MongoDB is a widely used NoSQL database that uses a document store approach. It often appears in literature which compares various NoSQL databases against each other, usually in terms of performance. There has been limited work to test the consistency guarantees of MongoDB or other NoSQL document stores. Kyle Kingsbury [23–25] has the most notable work in this area and has performed several analysis' of MongoDB's consistency claims using Jepsen, a distributed systems verification framework. This framework also includes the Knossos analysis tool which analyses the oplog (operation log) to verify there are no inconsistencies. MongoDB has been responsive to claims from Kingsbury and have now integrated Jepsen into their CI system [26].

The first analysis of Jepsen proved that MongoDB lost all acknowledged writes at all consistency levels when the a network error was encountered [23]. Kingsbury's second analysis, and most notable, tests MongoDB's consistency model which claims that is strictly consistent [24]. MongoDB defines strictly consistent as "A property of a distributed system requiring that all members always reflect the latest changes to the system. In a database system, this means that any system that can provide data must reflect the latest writes at all times." [27]. Kingsbury's experiment finds that when using a majority write concern and all reads using the primary preference, a network partition will conflict with MongoDB's claim of strong consistency and return a stale read. It is noted that MongoDB will also allow for alternating fresh and stale versions to be seen with successive reads. Another finding of Kingsbury in this analysis is the manifestation of dirty reads. Stale reads violates the property of read your writes, whilst the reading of alternating fresh and stale data violates the monotonic reads guarantee.

Whilst Kingsbury provides a thorough analysis of MongoDB's claim of being strictly consistent, it should also be noted that MongoDB has a second level of

consistency for its reads. MongoDB guarantees eventual consistency for reads with a preference of secondary. Currently, no work has been done to benchmark MongoDB with a secondary read preference or test if its secondary reads comply with the consistency guarantees of read your writes or monotonic reads.

5 Conclusion

In the reviewed literature, we have observed that NoSQL systems have had an increased adoption for use in web applications and have resulted in new models being created to represent trade offs in distributed systems. The need for a standard consistency benchmark for measuring eventual consistency is evident in the literature as there are currently gaps in various metrics used to quantify consistency. Having a benchmark would benefit both clients and service provider by providing a better understanding of what various consistency levels offer and when these levels are violated. In particular a benchmark to measure eventual consistency would allow us to measure MongoDB's claim of eventual consistency for reads to its secondary replica.

References

[1] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[2] D. Bermbach and S. Tai, "Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior," in *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*, p. 1, ACM, 2011.

[3] M. v. Steen and A. S. Tanenbaum, "Distributed systems: principles and paradigms," 2007.

[4] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys (CSUR)*, vol. 15, no. 4, pp. 287–317, 1983.

[5] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.

[6] N. Leavitt, "Will nosql databases live up to their promise?," *Computer*, vol. 43, no. 2, 2010.

[7] P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *Commun. ACM*, vol. 56, pp. 55–63, May 2013.

[8] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welch, "Session guarantees for weakly consistent replicated data," in *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*, pp. 140–149, IEEE, 1994.

[9] E. Brewer, "Cap twelve years later: How the" rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[10] D. Pritchett, "Base: An acid alternative," *Queue*, vol. 6, pp. 48–55, May 2008.

[11] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, 2000.

[12] D. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer*, vol. 45, pp. 37–42, Feb 2012.

[13] E. Anderson, X. Li, M. A. Shah, J. Tucek, and J. J. Wylie, "What consistency does your key-value store actually provide?," in *HotDep*, vol. 10, pp. 1–16, 2010.

[14] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, and X. S. Li, "Eventually consistent: Not what you were expecting?," *Commun. ACM*, vol. 57, pp. 38–44, Mar. 2014.

[15] Y. Zhu and J. Wang, "Client-centric consistency formalization and verification for system with large-scale distributed data storage," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1180–1188, 2010.

[16] D. Bermbach, L. Zhao, and S. Sakr, "Towards comprehensive measurement of consistency guarantees for cloud-hosted data storage services," in *Technology Conference on Performance Evaluation and Benchmarking*, pp. 32–47, Springer, 2013.

[17] M. R. Rahman, W. M. Golab, A. AuYoung, K. Keeton, and J. J. Wylie, "Toward a principled framework for benchmarking consistency.," in *HotDep*, 2012.

[18] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 143–154, ACM, 2010.

[19] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, "Probabilistically bounded staleness for practical partial quorums," *Proc. VLDB Endow.*, vol. 5, pp. 776–787, Apr. 2012.

[20] K. Zellag and B. Kemme, "How consistent is your cloud application?," in *Proceedings of the Third ACM Symposium on Cloud Computing*, p. 6, ACM, 2012.

[21] W. M. Golab, X. Li, and M. A. Shah, "Analyzing consistency properties for fun and profit.," in *PODC* (C. Gavoille and P. Fraigniaud, eds.), pp. 197–206, ACM, 2011.

[22] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data consistency properties and the trade-offs in commercial cloud storages: the consumers' perspective," in *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, (Asilomar, California USA), pp. 134–143, jan 2011.

[23] K. Kingsbury, "Jepsen: Mongodb," 05 2013.

[24] K. Kingsbury, "Jepsen: Mongodb stale reads." Website, 04 2015.

[25] K. Kingsbury, "Mongodb 3.4.0-rc3," 02 2017.

[26] J. Abrahams, "Testing linearizability with jepsen and evergreen: "call me continuously!" — the mongodb engineering journal," 2017.

[27] MongoDB, "Mongodb glossary."