Transactional YCSB: Benchmarking ACID-Compliant NoSQL Systems with Multi-Operation Transactions

Benedikt Maria Beckermann [1,2]

Abstract: NoSQL systems are popular because of their flexible data models and focus on availability, scalability, and fault tolerance. They often have loosened ACID guarantees to achieve these goals. To also support use cases that rely on ACID transactions, some existing NoSQL systems introduced ACID compliance after their release, and new NoSQL systems are created to support ACID compliance from the ground up. A benchmark that supports transactions is required to compare the performance of these ACID-compliant NoSQL systems. The Yahoo! Cloud Serving Benchmark (YCSB) is the most used benchmark for NoSQL systems but does not support transactions. YCSB+T, an extension of YCSB, introduces support for transactions into the YCSB, but only for transactions consisting of a single operation. A further extension of YCSB is required to support the performance evaluation of workloads containing transactions that consist of multiple operations.
This paper introduces *Transactional YCSB*, an extension of the benchmarking framework YCSB that enables the evaluation of ACID-compliant NoSQL systems for workloads consisting of multi-operation transactions. Further, the paper evaluates the ACID-compliant NoSQL systems FoundationDB, MongoDB, and OrientDB using the developed YCSB extension.

Keywords: Benchmark, NoSQL, Transaction, YCSB, FoundationDB, MongoDB, OrientDB

1 Introduction

NoSQL systems gained popularity in the past decades because they support more flexible data models and focus on availability, scalability, and fault-tolerance [ÖV20]. As a trade-off, NoSQL systems often loosen their consistency guarantees. For use cases where strong consistency guarantees are required, some NoSQL systems introduced full support for ACID transactions into their systems, such as the document store MongoDB [KC18]. Other NoSQL systems are built to support transactions from the ground up, such as the key-value store FoundationDB [Zh22]. Many of these ACID-compliant NoSQL systems provide benchmark results themselves, such as FoundationDB [Zh22], but these results are not necessarily comparable to other systems or workloads. Thus, a general benchmark for ACID-compliant NoSQL systems that support a range of transactional workloads is needed. Adapting existing and well-established OLTP benchmarks, such as TCP-C, to work on some NoSQL systems, such as document stores, is possible, but significant modifications are required to produce representative performances [Ka19]. While the *Yahoo! Cloud Serving Benchmark* (YCSB) [Co10] is one of the most used benchmarks for NoSQL systems [Cl21], it does not support transactions. The extension YCSB+T [De14] introduces transaction support and enables

[1] Schloss Dagstuhl - Leibniz Center for Informatics, dblp computer science bibliography, Trier, Germany, benedikt.beckermann@dagstuhl.de, https://orcid.org/0009-0008-3920-6109
[2] Universität Trier, Germany, benedikt.beckermann@dagstuhl.de, https://orcid.org/0009-0008-3920-6109

measuring the overhead of running individual operations as transactions and detecting data inconsistencies. It does not support workloads consisting of multi-operation transactions.

This paper introduces *Transactional YCSB*, an extension of YCSB, which introduces the missing support for transactions containing multiple operations. It enables the performance evaluation of transactional NoSQL systems for a new class of transactional workloads. Further, the newly introduced benchmark is used to evaluate three NoSQL systems, representing three different types of NoSQL systems: FoundationDB, which represents key-value stores; MongoDB, which represents document stores; and OrientDB, which represents graph DBMS.

The article is structured as follows: Section 2 presents an overview of transactional NoSQL systems and relevant benchmarks. Section 3 introduces the YCSB extension *Transactional YCSB*. Section 4 evaluates the performance of the three transactional NoSQL systems FoundationDB, MongoDB, and OrientDB using Transactional YCSB. Section 5 concludes the article.

2 Background and Related Work

This section provides background on transactional NoSQL systems and existing benchmarks for NoSQL systems.

2.1 Transactional NoSQL Systems

NoSQL systems are developed based on the requirements of web and cloud services. Other than traditional relational DBMS, NoSQL systems support different data models and focus on scalability, availability, and fault tolerance [ÖV20]. While some NoSQL systems loosen their consistency requirements to achieve these goals [ÖV20], other systems introduced optional ACID-compliance after their first release, such as MongoDB [KC18], or are built with support for ACID transactions from the beginning, such as FoundationDB [Zh22]. More thorough surveys of NoSQL systems are conducted by Gessert et al. [Ge17] and Davoudian et al. [DCL18]. Following, three different types of NoSQL systems are presented together with systems of that type supporting ACID transactions.

Key-Value Stores. Key-value stores represent data as pairs of keys and values [ÖV20]. They are schemaless to enable high flexibility and scalability. FoundationDB is such a key-value store [Zh22]. To achieve scalability, the system decouples the distributed storage engine from the transaction management system and allows the independent scaling of these sub-systems across several nodes. FoundationDB supports multi-key strictly serializable transactions by combining optimistic and multi-version concurrency control. Durability is ensured via replicated logs.

Document Stores. Document stores map keys to documents [ÖV20]. These documents are of self-describing types, such as XML, JSON, or YAML, and can contain nested substructures. Documents are typically grouped into collections, but documents of the same collection can have different schemas. MongoDB is such a document store [ÖV20] and uses a JSON-based data model and stores documents in binary-encoded JSON. MongoDB supports scaling across multiple nodes via sharding and primary-copy replication. Support for ACID transactions on multiple documents is provided using snapshot isolation. By ensuring a write action is propagated to a majority of replicating nodes, MongoDB ensures durability [Ku22].

Graph DBMS. Graph database management systems store data as a graph using a flexible schema to define vertex and edge types with individual properties [ÖV20]. The data of a Graph DBMS can be accessed typically through specific APIs or declarative query languages. OrientDB is a multi-model database management system that supports graph and semi-structured data management [Ri21]. Data is stored in documents and grouped in classes similar to collections in document stores. These documents can store properties and link to each other to form a graph. OrientDB can be scaled across multiple nodes using replication. The system supports ACID transactions using Fast Paxos [La06; Zh15] for distributed commits and ensures read isolation on commit.

2.2 Benchmarking Transactional NoSQL Systems

To evaluate the performance of NoSQL systems, benchmarks are needed. While it is possible to adapt the existing and well-established benchmarks for traditional relational databases, such as TCP-C, they require significant reworks to produce representative performances [Ka19]. Instead, new benchmarks are introduced with a focus on NoSQL systems. One of the most used benchmarks for NoSQL systems [Cl21; Fr14] is the Yahoo! Cloud Serving Benchmark [Co10], introduced in 2010.

The *Yahoo! Cloud Serving Benchmark* (YCSB) [Co10] is an extensible open-source benchmarking framework aimed at cloud data serving platforms. Instead of SQL, the YCSB uses a simple interface called `DB` containing the methods `read`, `insert`, `update`, `delete`, and `scan` to communicate with the platforms. ACID transactions are not natively supported. The framework contains several workload classes that define the general logic of benchmarks and workload files that clearly define individual benchmarks, including parameters of which workload class to use, how many operations to perform, what kind of key distribution to use, and more. The YCSB supports multiple key selection distributions, such as `uniform` distribution, where each key has the same likelihood of being selected, or `zipfian` distribution, where some keys are much more likely to be selected than others. The main workload class is called `CoreWorkload` and represents a client performing multiple CRUD operations, one at a time. The ratio of the different kinds of operations, the number and length of fields per record, and other properties can be set at runtime. To support the benchmarking of a cloud data serving platform, a binding is required that implements the

`DB` interface for the given platform. The current version[3] of the YCSB includes over 40 such bindings. The benchmark is executed using one or more client threads that perform the defined workload. The measurements taken by the client threads are reported to a statistic system that combines the measurements and reports the results once the benchmark has terminated successfully.

Because of the high prevalence and extensibility of YCSB, several extensions for the YCSB exist. The extension YCSB++ [Pa11] introduces multi-tester coordination and multi-phase workloads. Claesen et al. [Cl21] extend the YCSB by introducing a new distribution used for key selection that improves the benchmarking of hotspot behavior. In YCSB+T [De14], an optional transaction support for the `DB` interface is introduced. Additionally, YCSB+T provides support for the benchmarking of transactional overhead and consistency. The YCSB+T extension does not support transactions containing several operations. In a survey from 2022 about benchmarks for distributed transactional databases by Qu et al. [Qu22], YCSB+T is the only benchmark explicitly targeting transactional NoSQL systems. In recent years, transactional benchmarks for specific types of NoSQL systems have been introduced, such as a benchmark by Waudby et al. [Wa20] which validates ACID compliance in graph DBMS.

3 Making YCSB Transactional

This section describes the architecture of Transactional YCSB and the new transactional workload class. Then, the updated bindings for the evaluated systems are presented. The code of Transactional YCSB can be found on GitHub.[4]

3.1 Architecture of Transactional YCSB

The architecture of Transactional YCSB is presented in the following section by describing the implemented changes from YCSB. A simplified view of the adapted architecture is shown in Figure 1.

To add transaction support to the YCSB, the `DB` interface is extended by the methods `start`, `commit` and `rollback`, which respectively initiate, commit and rollback the active transaction. This interface is inspired by YCSB+T [De14]. Other than YCSB+T, these methods do not have an empty default implementation. Thus, database bindings for Transactional YCSB need to support them explicitly.

The class `DBWrapper` is a wrapper class for any implementation of the `DB` interface. The wrapper class forwards the database operations to the wrapped `DB` implementation, measures

[3] https://github.com/brianfrankcooper/YCSB (accessed 13.07.2024)
[4] https://github.com/greenBene/transactional-ycsb, (accessed 18.01.2025)

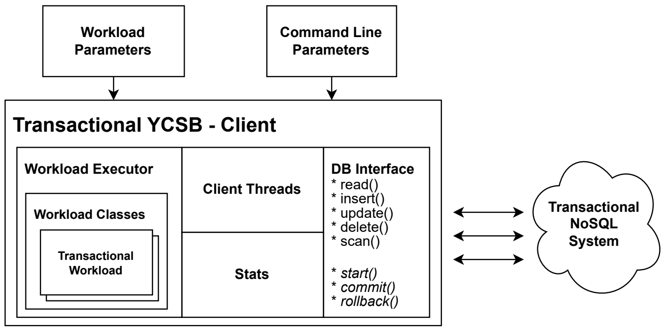


Fig. 1: The Architecture of Transactional YCSB.

the time in nanoseconds that the operation takes to execute, and stores the measured time together with information about the success of the executed command. To measure the performance of the newly supported `start`, `commit`, and `rollback` methods, the class `DBWrapper` is extended by implementations for these three methods.

The class `CoreWorkload` is updated to support transactions: A new transaction is started before executing a database operation. If the start of the new transaction is successful, the requested database operation is executed. The transaction is then committed if the database executes the operation successfully. Otherwise, the transaction is aborted. The other existing workload classes (`RestWorkload` and `TimeSeriesWorkload`) are updated accordingly.

Transactional YSCB aims to measure the performance of transactional NoSQL systems under a load of transactions consisting of multiple operations. Thus, a new workload class called `TransactionWorkload` is implemented as an extension of the `CoreWorkload` class. `TransactionWorkload` executes several transactions, each consisting of multiple `read`, `update`, `insert`, and `scan` operations. The number of transactions and the number of `read`, `update`, `insert`, and `scan` operations can be set via properties. The properties are defined via a command line argument or a workload file. Since the new workload class is an extension of the existing `CoreWorkload` class, it also supports adjusting the number and length of fields per record, the distribution of the selection of records to operate on, and more. A property file called `workloadt` is provided with an example workload setup where 300,000 transactions are executed that each first perform 9 `read` operations followed by one `update` operation. A uniform distribution is used to select the records to be read or updated.

3.2 Updated Bindings

YCSB, and thus Transactional YCSB, requires an implementation of the `DB` interface for systems to run the benchmark on. The implementations are called bindings. For the three systems evaluated by this paper, FoundationDB, MongoDB, and OrientDB, a binding exists in the current YCSB implementation. The bindings are updated to support the new transaction operation methods of the `DB` interface. The underlying client libraries used to communicate with the servers are updated to a current version for all three systems.

To support transactions in the FoundationDB binding, a new `Transaction` object is created each time a transaction is started using the `start` method. Subsequent database operations on the FoundationDB servers are performed using this transaction object. To commit or rollback an active transaction, the corresponding `commit` or `cancel` methods are called on the transaction object.

The support for transactions in the MongoDB binding is added similarly to the support for transactions in the FoundationDB binding. Here, a `ClientSession` object manages the active transaction. A call to the `start` method starts a new transaction within the session object. The database operations are executed using the session object and, thus, within the active transaction. To commit or rollback a transaction, the corresponding methods are called on the `ClientSession` object.

While the YCSB contains a binding for OrientDB, the binding is outdated and incompatible with current OrientDB client libraries. The paper introducing the redesigned OrientDB [Ri21] also presents an updated YCSB binding for OrientDB, which is not merged into the YCSB main branch at the time of writing.[5] This updated binding is used as a base to add support for transactions. A session object is used to support transactions in the OrientDB binding, similar to the MongoDB binding. This session object manages active transactions and is used to start a new transaction, execute database operations within the transaction, and commit or abort the transaction when finished.

4 Evaluation

The following section evaluates three well-established ACID-compliant NoSQL systems' performance. Each of the chosen systems represents a different type of NoSQL System: FoundationDB represents key-value stores, MongoDB represents document stores, and OrientDB represents Graph DBMS. Transactional YCSB is used to perform experiments that consist of read-heavy multi-operation transactions and differing distributions of accessed records.

4.1 Experiments

Many real-world use-cases for distributed NoSQL systems require read-heavy transactions consisting of many `read` and a few `update` operations [Co10; Ri21; Zh22]. To evaluate the performance of the systems in such use cases, four experiments are defined that emulate different read-heavy workloads. These experiments' workloads differ in length and

[5] https://github.com/brianfrankcooper/YCSB/pull/1468, (accessed 13.07.2024)

distribution of accessed records. The experiments are shown in Table 1. The experiments E1 and E2 consist of shorter transactions with 9 `read` operations and 1 `update` operation. The experiments E3 and E4 consist of longer transactions with 90 `read` and 10 `update` operations. To keep the overall load between the experiments comparable, there are 10 times fewer transactions in E3 and E4 than in E1 and E2 since their transactions consist of 10 times more operations. Experiments E1 and E3 use a uniform distribution of accessed keys to simulate use cases where the overlap of accessed records is small, such as systems that store user settings. Experiments E2 and E4 use a zipfian distribution of accessed keys to simulate use cases where the overlap of accessed records is high, such as social networks where a few posts are accessed noticeably more than the other posts. YCSB, and thus also Transactional YCSB, use 0.99 by default as the zipfian constant.

Experiment	Distribution	#Transactions	#Reads p. T.	# Updated p.T.
E1	uniform	300,000	9	1
E2	zipfian	300,000	9	1
E3	uniform	30,000	90	10
E4	zipfian	30.000	90	10

Tab. 1: List of performed experiments, showing their used distribution, number of transactions, number of read operations per transaction, and number of update operations per transaction.

4.2 Setup

A distributed cloud setup is used to perform the experiments. For each evaluated system, a cluster of three nodes is freshly set up. Two of the nodes are used to run the NoSQL system in a distributed setup (Azure D2ads v5 VM, 2x vCPU with up to 3.5 GHz, 8GB RAM, and 75 GB SSD). The third node executes the benchmark (Azure A2 v2 VM, 2x vCPU with up to 2.4 GHz, 4GB RAM, 20 GB SSD). The nodes are connected using a virtual network. All three systems are configured to run in a distributed setup on two nodes, replicating the data on both. To ensure the servers are set up in the same way for all experiments and to make the setup easily reproducible, the cluster is created using Terraform[6] and the used code is publicly available.[7]

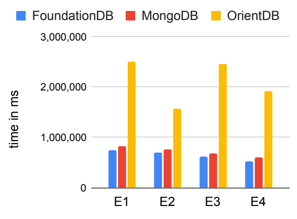
The server version used for FoundationDB is 6.3.23, the server version for MongoDB is 7.0.11, and the server version for OrientDB is 3.2.31. All three systems are tuned according to their documentation. The specific system setups are documented in a git repository.[8] All experiments are performed on a dataset of 5,000,000 records, each 1KB. The Transactional YCSB client is executed using 10 threads to simulate multiple users.

[6] https://www.terraform.io/, (accessed 18.01.2025)

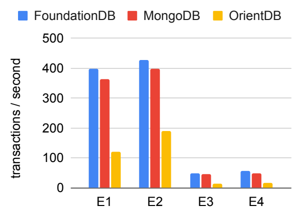
[7] https://github.com/greenBene/transactional-ycsb-benchmark/tree/main/terraform-cluster, (accessed 18.01.2025)

[8] https://github.com/greenBene/transactional-ycsb-benchmark/tree/main/database-setups, (accessed 18.01.2025)

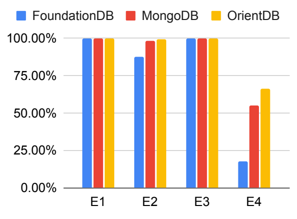
4.3 Results



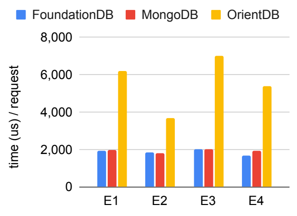
(a) Total runtime in milliseconds.



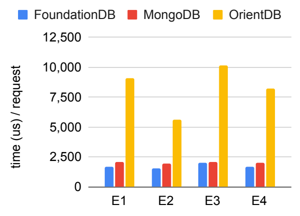
(b) Average throughput in transactions per second.



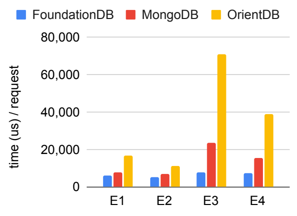
(c) Percentage of successfully committed transactions.



(d) Average latency of read operations.



(e) Average latency of update operations.



(f) Average latency of commit operations.

Fig. 2: Results of the performed experiments.

The overall performance of the three evaluated NoSQL systems running the experiments can be seen in Figures 2a-c. For all four experiments, the total runtime and the average throughput of both FoundationDB and MongoDB are comparable, but FoundationDB has a slightly higher throughput. OrientDBs throughput is in experiments E1, E3, and E4 about a third of the throughput of the others, and in experiment E2 about half.

Figure 2c shows the percentage of successfully committed transactions per experiment. All three systems achieve 100% (E1) or close to 100% (E3) in the experiments using a uniform distribution of selected records. In the experiments using a zipfian distribution, OrientDB performs best with again nearly 100% successful commits in short transactions (E2) and the highest success rate with 66.3% for longer transactions (E4). MongoDB performs similarly to OrientDB but has a slightly lower success rate when the zipfian distribution is used, with 98.5% for shorter transactions (E2) and 55% for longer transactions (E4). FoundationDB has the lowest success rate with 87.9% for shorter transactions using the zipfian distribution (E2) and by far the lowest success rate for longer transactions using the zipfian distribution (E4) with a success rate of 17.5%.

For the `read` operations (see Fig. 2d), the average latency of FoundationDB and MongoDB is similar across all four experiments. Both systems have a slightly lower latency when selecting records to read using the zipfian distribution (E2, E4) compared to the uniform distribution (E1, E3). OrientDBs latency for `read` operations is about two to three times higher than the latency of the other two systems. OrientDBs latency when selecting records to read using the zipfian distribution (E2, E4) is also lower than when using the uniform distribution (E1, E3). The throughput of the experiments with shorter transactions (E1, E2) is about 10 times higher than the throughput of the experiments with longer transactions (E3, E4) because the transactions in E3 and E4 contain 10 times as many operations.

The average latency for `update` operation (see Fig. 2e) shows a similar pattern to the latency of `read` operations. FoundationDB and MongoDB perform similarly, while OrientDB has up to 5 times higher latencies. OrientDB performs 20% to 30% better using the zipfian distribution. For FoundationDB and MongoDB, there is no major difference between uniform and zipfian distribution.

FoundationDB has the lowest average `commit` operation latency (see Fig. 2f) for all four experiments. The latency remains mostly the same between shorter (E1, E2) and longer (E3, E4) transactions. While MongoDBs latency for shorter transactions (E1, E2) is similar to the latency of FoundationDB, MongoDBs latency for longer transactions (E3, E4) is between two to three times higher than the latency of FoundationDB. MongoDBs latency for longer transactions using a zipfian distribution for record selection (E4) is a third faster than using a uniform distribution for record selection. While OrientDBs `commit` latency for shorter transactions (E1, E2) is about twice as high as the latency of FoundationDB, OrientDBs latency for longer transactions (E3, E4) is five to ten times higher than the latency of FoundationDB and two to three times higher than the latency of MongoDB.

4.4 Discussion

Based on the performed experiments, FoundationDB shows the highest throughput for read-heavy transactions, independent of record key distribution and transaction length. Because of FoundationDBs optimistic concurrency control, the system is suitable for workloads with few conflicts, such as workloads with short transactions (E1, E2) or workloads with long transactions that access records in uniform distribution (E3). However, the optimistic concurrency control also causes a low success rate for workloads with many conflicts (E4), and thus FoundationDB is unsuitable for these use cases.

OrientDB shows the lowest throughput and highest latency for all operations across all experiments. OrientDB also has the highest rate of successfully committed transactions in all experiments. This behavior could indicate that the systems conservative concurrency control contributed to the lower performance. OrientDB is also written in Java, compared to MongoDB and FoundationDB, which are written in C++. This use of a less machine-oriented programming language could cause lower performance.

MongoDB shows a similar but slightly lower throughput compared to FoundationDB in all four experiments but a higher success rate for commits, especially for transactions with a high conflict rate. Thus, MongoDB is a good choice if a workload can contain transactions with a higher conflict rate and a good, but not the highest, throughput is acceptable.

4.5 Limitations

YCSB, the benchmark the Transactional YCSB framework is built upon, is designed to be compatible with a wide variety of NoSQL systems. Thus, YCSB only supports a small subset of common operations, such as `read`, `update`, `insert`, and `scan`. More complex operations, such as graph traversal, searching through records, or similar, are not supported. NoSQL systems that enable such advanced operations at the cost of the performance of the basic operations might perform worse when evaluated using Transactional YCSB compared to other NoSQL systems focused on the common operations. Also, the created transactional workload class does not verify the consistency of the data but assumes the system's reported ACID compliance. For workloads that evaluate the consistency, see YCSB+T [De14]. Further, the system evaluation cluster is relatively small compared to many production deployments [Zh22]. The evaluated NoSQL systems might perform differently on bigger clusters. Thus, this paper's results can only be seen as an indication of the performance of the evaluated NoSQL systems. Further experiments with different setups are needed to create more generally applicable results.

5 Conclusion

This paper introduces the Transactional YCSB, a benchmark based on the YCSB that focuses on transactional workloads in NoSQL systems. It presents the architectural changes of Transactional YCSB compared to YCSB and introduces a new transactional workload. The performance of three different NoSQL systems, representing three different types of NoSQL systems running read-heavy transactional workloads, is evaluated using the Transactional YCSB.

While the discussed evaluations themselves are only a starting point for future research and cannot be used to make more general statements about the evaluated NoSQL systems, the newly introduced benchmark has proved to provide comparable insights into the performance of transactional NoSQL systems of different types that were not provided by previous benchmarks. A potential use case would be a company needing a transactional NoSQL system. The company could set the parameters of the transactional workload class to the expected load and use the benchmark to evaluate several NoSQL systems to find the system that best performs according to the company's specific needs.

Future work for Transactional YCSB can go in several directions. First, it can be evaluated if the separation of YCSB and Transactional YCSB is beneficial or if the two systems should and could be merged. Next, the implemented transactional workload can be extended to support different use cases. An optional randomized order for the operations within a transaction and the support for workloads that consist of transactions with differing operator distribution could be implemented and evaluated.

References

[Cl21] Claesen, C.; Rafique, A.; Landuyt, D. V.; Joosen, W.: A YCSB Workload for Benchmarking Hotspot Object Behaviour in NoSQL Databases. In (Nambiar, R.; Poess, M., eds.): Performance Evaluation and Benchmarking - 13th TPC Technology Conference, TPCTC 2021, Copenhagen, Denmark, August 20, 2021, Revised Selected Papers. Vol. 13169. Lecture Notes in Computer Science, Springer, pp. 1–16, 2021, doi: 10.1007/978-3-030-94437-7_1.

[Co10] Cooper, B. F.; Silberstein, A.; Tam, E.; Ramakrishnan, R.; Sears, R.: Benchmarking cloud serving systems with YCSB. In (Hellerstein, J. M.; Chaudhuri, S.; Rosenblum, M., eds.): Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010. ACM, pp. 143–154, 2010, doi: 10.1145/1807128.1807152.

[DCL18] Davoudian, A.; Chen, L.; Liu, M.: A Survey on NoSQL Stores. ACM Comput. Surv. 51 (2), 40:1–40:43, 2018, doi: 10.1145/3158661.

[De14] Dey, A.; Fekete, A. D.; Nambiar, R.; Röhm, U.: YCSB+T: Benchmarking web-scale transactional databases. In: Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE 2014, Chicago, IL, USA, March 31 - April 4, 2014. IEEE Computer Society, pp. 223–230, 2014, doi: 10.1109/ICDEW.2014.6818330.

[Fr14] Friedrich, S.; Wingerath, W.; Gessert, F.; Ritter, N.: NoSQL OLTP Benchmarking: A Survey. In (Plödereder, E.; Grunske, L.; Schneider, E.; Ull, D., eds.): 44. Jahrestagung der Gesellschaft für Informatik, Big Data - Komplexität meistern, INFORMATIK 2014, Stuttgart, Germany, September 22-26, 2014. Vol. P-232. LNI, GI, pp. 693–704, 2014, url: https://dl.gi.de/handle/20.500.12116/2970.

[Ge17] Gessert, F.; Wingerath, W.; Friedrich, S.; Ritter, N.: NoSQL database systems: a survey and decision guidance. Comput. Sci. Res. Dev. 32 (3-4), pp. 353–365, 2017, doi: 10.1007/S00450-016-0334-3.

[Ka19] Kamsky, A.: Adapting TPC-C Benchmark to Measure Performance of Multi-Document Transactions in MongoDB. Proc. VLDB Endow. 12 (12), pp. 2254–2262, 2019, doi: 10.14778/3352063.3352140.

[KC18] Keep, M.; Cabral, A.: MongoDB 4 Update: Multi-Document ACID Transactions, https://www.mongodb.com/blog/post/mongodb-multi-document-acid-transactions-general-availability, Accessed: 2024-07-16, 2018.

[Ku22] Kukic, A.: Set Global Read and Write Concerns in MongoDB 4.4, https://www.mongodb.com/developer/products/mongodb/global-read-write-concerns/, Accessed: 2024-09-24, 2022.

[La06] Lamport, L.: Fast Paxos. Distributed Comput. 19 (2), pp. 79–103, 2006, doi: 10.1007/S00446-006-0005-X.

[ÖV20] Özsu, M. T.; Valduriez, P.: Principles of Distributed Database Systems, 4th Edition. Springer, 2020, isbn: 978-3-030-26252-5.

[Pa11] Patil, S.; Polte, M.; Ren, K.; Tantisiriroj, W.; Xiao, L.; López-Hernández, J. C.; Gibson, G.; Fuchs, A.; Rinaldi, B.: YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In (Chase, J. S.; Abbadi, A. E., eds.): ACM Symposium on Cloud Computing in conjunction with SOSP 2011, SOCC '11, Cascais, Portugal, October 26-28, 2011. ACM, p. 9, 2011, doi: 10.1145/2038916.2038925.

[Qu22] Qu, L.; Wang, Q.; Chen, T.; Li, K.; Zhang, R.; Zhou, X.; Xu, Q.; Yang, Z.; Yang, C.; Qian, W.; Zhou, A.: Are current benchmarks adequate to evaluate distributed transactional databases? BenchCouncil Transactions on Benchmarks, Standards and Evaluations 2 (1), p. 100031, 2022, ISSN: 2772-4859, DOI: https://doi.org/10.1016/j.tbench.2022.100031.

[Ri21] Ritter, D.; Dell'Aquila, L.; Lomakin, A.; Tagliaferri, E.: OrientDB: A NoSQL, Open Source MMDMS. In (Pirk, H.; Heinis, T., eds.): Proceedings of the The British International Conference on Databases 2021, London, United Kingdom, March 28, 2022. Vol. 3163. CEUR Workshop Proceedings, CEUR-WS.org, pp. 10–19, 2021, URL: https://ceur-ws.org/Vol-3163/BICOD21%5C_paper%5C_3.pdf.

[Wa20] Waudby, J.; Steer, B. A.; Karimov, K.; Marton, J.; Boncz, P. A.; Szárnyas, G.: Towards Testing ACID Compliance in the LDBC Social Network Benchmark. In (Nambiar, R.; Poess, M., eds.): Performance Evaluation and Benchmarking - 12th TPC Technology Conference, TPCTC 2020, Tokyo, Japan, August 31, 2020, Revised Selected Papers. Vol. 12752. Lecture Notes in Computer Science, Springer, pp. 1–17, 2020, DOI: 10.1007/978-3-030-84924-5_1.

[Zh15] Zhao, W.: Fast Paxos Made Easy: Theory and Implementation. Int. J. Distributed Syst. Technol. 6 (1), pp. 15–33, 2015, DOI: 10.4018/IJDST.2015010102.

[Zh22] Zhou, J.; Xu, M.; Shraer, A.; Namasivayam, B.; Miller, A.; Tschannen, E.; Atherton, S.; Beamon, A. J.; Sears, R.; Leach, J.; Rosenthal, D.; Dong, X.; Wilson, W.; Collins, B.; Scherer, D.; Grieser, A.; Liu, Y.; Moore, A.; Muppana, B.; Su, X.; Yadav, V.: FoundationDB: A Distributed Key Value Store. SIGMOD Rec. 51 (1), pp. 24–31, 2022, DOI: 10.1145/3542700.3542707.