A Survey on the Usage of Consensus Algorithms in Key-Value Databases

Chengyu Chen
University of Michigan
Ann Arbor, USA
ccccy@umich.edu

Yatian Liu
University of Michigan
Ann Arbor, USA
dougliu@umich.edu

Abstract

Recent rapid developments of distributed and non-relational (or NoSQL) databases not only reflects much attention drawn by their high scalability, low latency, and flexibility, but also diversifies the implementations of database management systems. Key-value database (or key-value store) is a widely-used type of NoSQL database. Data consistency is one fundamental requirement for both traditional databases and key-value stores, and the flexible application scenarios of key-value stores lead to different choices of consistency levels to satisfy other performance metrics. This survey first summarizes recent progress in consensus algorithms, which is often used to implement reliable state machine replication and strong data consistency, and then investigate how commonly deployed key-value databases implement strong consistency with consensus algorithms as one option. The results of the survey show that the latest advances in consensus algorithms have already been applied in some popular key-value stores, other SMR protocols like primary-backup are also used for strong consistency, and many key-value stores choose to offer a variety of consistency levels instead of providing only eventual consistency or strong consistency.

Keywords: survey, distributed system, key-value store, consensus, consistency, replication, reliability

1 Introduction

While relational databases still hold a considerable amount of shares on the internet, non-relational databases are increasingly popular in the industry, including Cassandra [22], Dynamo DB [10], etcd [12], etc. Compared to the traditional rows-and-columns schema, non-relational databases often drift away from those reliable, powerful yet complex design choices and instead embraces the system simplicity, horizontal scalability, and structural flexibility. Representative types of such databases consist of key-value store, document store, graph store, and column store, each of them prioritizing different functionalities. Among them, key-value store is the simplest model and both the document store and graph store model are based key-value store, and it is therefore available for most NoSQL databases.

Ensuring consistency of replicated data is a basic problem for any distributed data storage systems, and key-value databases are no exceptions. Since key-value databases are used for more flexible applications that do not always require strong consistency such as linearizability, key-value database designs choose different levels of consistency to satisfy different performance requirements. For instance, the Amazon Dynamo database [10] provides very high availability while only offers eventual consistency of replicated data, and the Redis database [35] also values availability and flexibility over consistency, while the etcd database [12] uses the Raft consensus algorithm [28] for replication to ensure strong consistency. We would like to study and compare the different choices of popular distributed key-value database implementations to learn when strong consistency is important in key-value databases and how strong consistency is implemented in key-value stores. One common way to provide strong consistency is state machine replication (SMR), and consensus algorithms can implement reliable SMR.

We conduct this survey to: 1. present a systematic review on the popular academic paper about consensus algorithms between 2010 and 2020; 2. review and analyze how commonly deployed key-value databases utilize consensus algorithms for implementing strong consistency. After preliminary filtering, 10 popular distributed key-value databases are surveyed on their implementation of strong consistency. Then, some interesting common traits are observed and analyzed, including the apparent favor for simple SMR protocols over general consensus algorithms for data consistency, the wide adoption of the more recent Raft consensus protocol in place of the classic Paxos protocol, and the diversity of supported consistency levels as well as the corresponding implementations.

2 Related Works

Several surveys have been performed as attempts to evaluate NoSQL databases. Hecht et al., motivated by the rise of many non-relational databases around 2011, evaluated many NoSQL databases from various angles: data model, query possibilities, etc [19]. Han et al. summarized prominent characteristics among popular highly scalable and highly concurrent NoSQL databases [15]. Furthermore, Gessertl et al.

EECS 591, Fall 2021, Ann Arbor, MI, USA

in 2016 provided a top-down view of the diverse NoSQL databases in order to characterize the decision factors buried under their heterogeneity [14].

All of the works mentioned above managed to provide excellent insight in this field. However, these works mostly focuses on characteristics of general NoSQL databases, and we consider it suitable to survey distributed key-value stores, the most ubiquitous variants, as an delegation to gain more specific knowledge of the current directions of the industries. In particular, the involvement of consensus algorithms for implementing consistency was not paid much attention in previous works, but it is covered as the most important aspect in our survey.

3 Survey Methods

Our survey contains two parts, literature survey and database survey, with the focus on database survey.

For literature survey, we would like to systematically search for recent influential advances in consensus algorithms, so we choose to perform a advanced search on Scopus, sort the result by number of citations, and briefly check the content of the top 15 results to see if they propose consensus algorithms suitable for implementing strong consistency in databases. The search keyword we use is

```
TITLE-ABS(consensus) AND ALL("distributed
system") AND SUBJAREA(COMP) AND (PUBYEAR > 2009)
AND (PUBYEAR < 2021),
```

which corresponds to papers satisfying the following requirements:

- Contains “consensus” in title or abstract.
- Contains “distributed system” anywhere.
- Is in the Computer Science subject area classified by Scopus.
- Is published in the year range (2009, 2021).

Database survey is divided into two stages: preliminary survey and detailed survey. In the preliminary survey, we refer to the popularity ranking on DB-Engines [38] to find the top 20 popular databases offering key-value store capability. DB-Engines is one of the most comprehensive knowledge bases of relational and NoSQL database management systems (DBMS). It uses an open and objective method [37] to rate and rank databases. The rankings are updated once a month and results in November are used for preliminary survey. We briefly surveyed whether the top 20 databases offer strong consistency, and then choose the top 10 databases that offer it for detailed survey. In detailed survey, we focus on finding and understanding the implementation of strong consistency for each database and whether the implementation uses a consensus algorithm, and we also note down the main distinguishing features of each database.

4 Literature Survey Results

In this section, the top 15 most cited papers selected by the previous methods are briefly reviewed to see if they propose new consensus algorithms that can be applied to key-value stores and general DBMS.

[5, 11, 27, 42, 43, 45–47] are related to consensus algorithms in blockchain, which deal with Byzantine fault tolerance and are not needed for databases. [24] describes a consensus protocol for large-scale dynamic networks with Byzantine nodes and no non-local information, which is not suitable for databases, either. [28] introduces the Raft algorithm, a consensus algorithm for asynchronous network and non-Byzantine fault tolerance based on Paxos [23]. [8, 13, 18, 21, 44] are not actually focused on consensus algorithms in distributed systems. From this result, we think [28] is the only paper that introduces a consensus algorithm (i.e. Raft) suitable for database consistency implementation.

Raft is a consensus algorithm whose function is equivalent to multi-Paxos but is more understandable and easier to implement. Same as multi-Paxos, it guarantees safety but not always liveness under an asynchronous network and manages appends to a replicated log. One major problem for Paxos is that it is difficult to understand and the original description of the algorithm requires many supplemental details and complex changes to be applied to practical systems. In order to make Raft easier to understand than Paxos, Raft’s design uses techniques including decomposition of tasks (leader election, log replication, and safety) and state space reduction (“reduces the degree of non-determinism and the ways servers can be inconsistent with each other”). User studies conducted by the authors show that Raft is easier for university students to learn than Paxos, and it already had multiple open-source implementations when the paper was published.

5 Database Survey Results

5.1 Preliminary Survey

This section presents the level of consistency provides by the top 20 key-value databases on DB-Engines as described in the Section 3. Since this is only a brief survey on a simple aspect, references to each database’s documentation is not given here. The surveyed databases can be classified into five categories:

1. etcd (rank 6), Apache Ignite (rank 10), and Aerospike (rank 11) list strong consistency as one of their main features.
2. Redis (rank 1), Amazon DynamoDB (rank 2), and Riak KV (rank 8) prioritizes availability and only provides eventual consistency. It is worth noting that Redis can optionally provide casual consistency, and the Redis Labs is currently developing a module for Redis that supports strong consistency using Raft [34].

3. Microsoft Azure Cosmos DB (rank 3), Couchbase (rank 4), Hazelcast (rank 7), Ehcache (rank 9), ArangoDB (rank 12), OrientDB (rank 13), Oracle NoSQL DB (rank 14), ScyllaDB (rank 15), InterSystems Caché (rank 17), and Infinispan (rank 19) all offer different consistency levels or data replication strategies, and strong consistency is available as an option.
4. memcached (rank 5) is distributed does not have replication for data and is not intended as a persistent storage.
5. RocksDB (rank 16), Oracle Berkeley DB (rank 18), and LevelDB (rank 20) are not distributed and can only operate locally.

Only databases in Class 1 and 3 supports strong consistency, so we then choose the top 10 from the two classes for detailed survey. The databases chosen are: Azure Cosmos DB, Couchbase, etcd, Hazelcast IMDG, Ehcache, Apache Ignite, Areospike, ArangoDB, OrientDB and finally Oracle NoSQL.

5.2 Detailed Survey

In this sub-section, we presents the consistency related features of the top 10 key-value databases on DB-Engines selected in the last sub-section that claim to provide strong consistency, and how they implement strong consistency if they disclose this information. General description and main distinguishing features of each database is also given.

5.2.1 Azure Cosmos DB. Azure Cosmos DB is a "fast NoSQL database with SLA-backed speed and availability, automatic and instant scalability, and open-source APIs for MongoDB and Cassandra" [25] developed by Microsoft. Initially released in 2017, it is very flexible, providing various NoSQL database models such as key-value store, document store, and graph database. It also has a flexible configuration and uses SLA (Service Level Agreements) to customize the performance guarantees for different clients. Backed by Microsoft's worldwide databases, it can distribute data in multiple regions across the world.

In terms of consistency, Azure Cosmos DB is also flexible and offers five different consistency levels to configure: strong, bounded staleness, session, consistent prefix, and eventual [26]. Limited by the PACELC theorem, different consistency level provides different availability and latency guarantees. Unfortunately, the database is proprietary and the documentation only describes what the five consistency levels are without talking about the detailed implementation of each level. In addition to server configuration for consistency levels, it also allows a client to override the configuration and asks the servers to provide a weaker consistency level for reads and writes in order to get better availability and latency guarantees. One thing to note is that strong consistency level for accounts with regions spanning more than 5000 miles is not supported by default due to high latency concerns.

5.2.2 Couchbase. Initially released in 2010, Couchbase is an open source, distributed, document-oriented, NoSQL database [9]. Key-value stores are also supported. One of the most prominent features that Couchbase provides is the flexibility in terms of durability, availability and latency.

Under the hood, Couchbase utilizes replications to improve both fault tolerance and abilities to fail over. With regards to implementations, however, different techniques are employed in difference scopes - intra-cluster or inter-cluster. Within a cluster, Couchbase uses primary-backup to replicate data and thus has an option to ensure strong consistency by configuring query services to wait for all replicas before returning results. However, when it comes to inter-cluster environment, where Couchbase maintains two symmetrical and unidirectional replicas, only eventual consistency is offered [9].

No consensus algorithms are involved in intra-cluster replications, and we suspect that it is because synchrony is guaranteed within a cluster and thus multiple primaries cannot be present at the same time, which eliminates the need for implementing consensus. It is a different case in inter-cluster communications because the geographically diverse data-centers imposes significant network instability and latency and therefore invalidate the synchrony assumption mentioned before. Consensus algorithms, however, are not utilized neither in this case, and it is likely because Couchbase prefers synchronization speeds over consistency.

5.2.3 etcd. etcd is "a distributed, reliable key-value store for the most critical data of a distributed system" [12]. It is a free and open-source project led by Cloud Native Computing Foundation. Its name means "distributed etc directory" where etc refers to the `/etc` directory in UNIX-based operating systems. It focuses on providing strong consistency and high performance for the core configuration files for a larger distributed system, while the maximum storage size (several gigabytes according to the documentation) is relatively limited compared to other key-value stores. It is used by the popular Kubernetes project to store its configuration data.

etcd's documentation states clearly that it uses the Raft consensus protocol for replication to implement strong consistency. In order to have a simple system design that support strong consistency, it choose to not support data sharding and only has a single consistent replication group. This means that the system does not need to store the allocation of key ranges to different replication groups and only needs the Raft protocol to achieve strong consistency, but it also limits the maximum storage size of the system since large data size without sharding will lead to poor performance.

5.2.4 Hazelcast IMDG. Hazelcast In-Memory-Data-Grid (IMDG) is an open-source distributed in-memory object store supporting a wide variety of data structures that published in year 2008 [16]. It implements both eventual consistency and strong consistency using different sets of techniques. In the

context of the famous CAP theorem, AP (ensure Availability during Partitions) systems are built using primary-backup and lazy replication (also known as optimistic replication, meaning updates will be executed locally and then be propagated to other replicas). As for CP (ensure Consistency during Partitions) systems, the Raft consensus algorithm is being employed to build a strongly consistent layer for distributed data structures [17]. Such flexibility allows users to choose between availability and consistency based on their practical usages.

5.2.5 Ehcache. Ehcache is a Java cache library with a key-value interface and its developers claim that it is "Java's most widely-used cache" [39]. A cache library is like the cache in a CPU, which can store data from slow resources like queries to SQL databases in a fast storage. It is a free and open-source project developed by Terracotta, Inc. It can run in both local mode and distributed mode, has a tiered storage structure with different speed, and supports flexible configurations for cache expiration and eviction policies [40].

When operated in distributed mode, Ehcache offers two levels of consistency: eventual consistency and strong consistency. Ehcache uses Terracotta [41] for distribution, and the data replication is therefore done by the Terracotta server clusters. From documentation, Terracotta uses a replication scheme similar to primary-backup and servers have two modes: active servers and passive servers [36]. One cluster has only one active server and several passive servers. The active server processes clients' requests and replicates received messages to all the passive servers, and when the active server is considered down a passive server will be elected to become the new active server. Although the documentation does not say it, we think Terracotta assumes network synchrony since primary-backup is used. When consistency level of Ehcache is set to strong, write operations will return only after the active server finishes replication, and when consistency level is set to eventual, write operations will return after the active server finishes processing request locally, resulting shorter latency.

5.2.6 Apache Ignite. Apache Ignite is a distributed database for high-performance computing with in-memory speed. Graduated from Apache Incubator in 2015, Ignite has gained a lot of popularity [38]. One of the many features that Ignite offers is that operations can be either *atomic* or *transactional*. Multiple writes can be performed atomically at once in transactional mode, which requires lock mechanisms.

In atomic mode, no consensus algorithm is involved as high availability and fault tolerance are achieved by the primary-backup technique. Users have the flexibility to configure the number of backups to decide the trade-off between availability and latency. What's more, the level of synchronization between primary and backups can also be adjusted so that nodes servers may or may not return write acknowledgement before the updates propagating to all the replicas and thus furthermore influence both availability, fault tolerance and latency [6].

A consensus algorithm is deployed to facilitate locks in transactional mode. To ensure consistency across all the participating nodes, Ignite utilizes Two-Phase Commit (2PC) to guarantee the safety property in such scenarios: in phase one, the coordinator node sends "prepare" messages that notify recipients to acquire locks to primary nodes and wait for their acknowledgements, and these primary nodes also send such messages to their backup nodes and wait for their acknowledgements; in second phase, the workflow is the same except that the messages sent now serve the purpose to notify data nodes to commit [4]. The transaction concludes when acknowledgement for commit messages are sent back in a reverse order in this chain of commands.

5.2.7 Aerospike. Aerospike is a distributed key-value database that is scalable and provides ACID reliability, strong consistency, and operational efficiency [2]. Initially published in 2012, it has a proprietary enterprise edition and a community edition that was previously free and open-source but now released under a commercial license. The value content for a key is flexible, ranging from a set of binary data to structured and typed documents like JSON. Strong consistency along with scalability is one key selling point of the database.

Strong consistency mode is only provided in the enterprise edition. From the architecture white book [1], Aerospike implements its own protocol, which is quite complex, for replication and failure recovery. From a high level view, an integer replication factor n is set in configuration and n nodes are randomly selected from all the nodes to store replicas of a data partition. The node selection is recorded in a roster shared by all the nodes. One of the n nodes is master. Write operations can only be processed and replicated by the master node while read operations can be processed by all the replicas. Master will wait for acknowledgement from all the other replicas for write operations. Heartbeat is used for server health monitoring, and when one replica loses connection a new node will be selected as a replica, so as long as there are remaining active nodes a partition will always have n active and consistent replicas. If master node loses connection, a new master can also be elected under certain network partition scenarios specified by the white book. From the online documentation [3], the re-election rule guarantees 100% availability with strong consistency when there is only a two-way network partition, which may be a common real-life failure mode. Aerospike can implement strong consistency using the previous protocol even in geo-replicated clusters with latency of several hundred milliseconds, but it also offers an replication option with eventual consistency called cross-data replication (XDR) with shorter latency.

5.2.8 ArangoDB. ArangoDB is a native multi-model, open-source database with flexible data models for documents,

graphs, and key-values that came to public in 2012. It features ACID transactions [7].

Its replication of data servers also uses a primary-backup based protocol and does not involve consensus algorithms, presumably because of the synchrony assumptions within clusters. To ensure consistency without consensus algorithms, ArangoDB makes sure the replication is synchronous and therefore improve consistency at the cost of increased latency [7]. But the Raft consensus algorithm is still used during replication of configuration servers. These agents with consensus across them all come with many benefits, the most important ones of which are their abilities to supervise the fail-over process consistently (acting as a last resort) and the safety property of the vital configurations [7].

5.2.9 OrientDB. OrientDB is a free and open-source multi-model NoSQL database that was released in 2010. The developers claim that it is "the first multi-model open source NoSQL DBMS" that combines graph database with document store [32], and it also provides key-value store. It has a commercial enterprise edition but the open-source edition has most of the essential features.

OrientDB has a leaderless (or multi-master) replication scheme, and every node in a replica group can process write requests [33]. It has two replication parameters `readQuorum` and `writeQuorum`, which specify the number of coherent read and write responses to wait for before sending replies to clients [31]. This idea is similar to the one in Flexible Paxos [20] which published later than the system's initial release time, but since OrientDB uses multi-master replication the order of read and write requests cannot be uniquely determined and thus the scheme used by OrientDB cannot be used for consensus. The developers claim that by setting `writeQuorum` to "all" strong consistency can be achieved, but they did not explain in detail about how concurrent write conflicts are resolved and how availability will be influenced by setting `writeQuorum` to "all".

5.2.10 Oracle NoSQL. Oracle NoSQL, as the name suggests, is a multi-model, scalable, distributed NoSQL database, designed to provide highly reliable, flexible, and available data management across a configurable set of storage nodes [30].

To ensure availability and fault tolerance, like many other NoSQL databases, Oracle NoSQL takes the approach of primary-backup replications. The benefits of such architectures are that multiple data servers can now offload the concentrated read/write stress from the master server and that single-point-of-failure can be mitigated. What's more, such schema also entails the configurablity of levels of consistency of data across data servers. The documentation of Oracle NoSQL states that the consistency can be set from one extreme — eventual consistency — where write operations at primary nodes return immediately without waiting for the updates to propagate to all replicas, to another extreme — strict consistency — where the write operations block until all the data are synchronized. These common yet useful practices allow administrators to fully utilize the system depending on their in-real-life use cases [29].

Consensus algorithms, however, still plays an important role when it comes to fault tolerance. Failures of replicas can be easily resolved as the most updated view of data are still accessible on primary nodes and the other replicas are still in coordination led by the primary node. But faulty primary nodes do make a difference in this context. These failure models can be mitigated by a reliable Paxos election to promote a replica to be a new primary [29] that resolves conflicts consistently.

6 Discussion

In this section, some interesting observations of the database surveyed are listed and analyzed.

Firstly, it is frequently observed that different consistency levels are supported, from the most relaxed "eventual consistency" to the most strict "strong consistency". More specifically, 7 out of 10 surveyed databases feature such flexibility: Azure Cosmos DB, Hazelcast, Ehcache, Apache Ignite, Aerospike, OrientDB and Oracle SQL. Some, like Azure Cosmos DB, even provides more than two levels to provide more freedom of configuration. Users, therefore, can exploit such flexibility and determine their ideal balance and trade-off between consistency and latency.

For the majority of these databases, such diversity are inherited naturally in their replication approach, i.e., primary-backup or its variants. By setting the write operations to return before changes propagate to all replicas, queries will then face potential data inconsistency (usually on replicas) but the overall latency cost is minimized. Vice versa, once it is configured that write operations block until the data is synchronized across data servers, read requests will no longer need to address inconsistency concerns at the cost of latency. But Hazelcast takes a different approach by implementing eventual consistency with AP system (primary-backup with lazy replication) and strong consistency with another CP system (Raft) in the context of CAP theorem.

Secondly, Raft is used more often than Paxos in implementing strong consistency for the database surveyed. In the 10 databases surveyed in detail, etcd and Hazelcast use Raft for data replication and ArangoDB uses Raft for configuration replication, while only Oracle NoSQL DB uses Paxos, and Paxos is only used for new leader election in primary-backup failure recovery but not the full data replication process. We think this result demonstrates that Raft is indeed easier to understand and implement than Paxos, and we are glad to see that the recent development of consensus algorithms have already been applied to industrial applications.

Thirdly, it is shown that besides consensus algorithms, primary-backup is another popular way to implement strong consistency, which was not expected before the survey. In fact, only a correct state machine replication (SMR) algorithm is needed to ensure strong consistency, and if network synchrony can be assumed primary-backup is sufficient for SMR. A majority of systems (Couchbase, Ehcache, Apache Ignite, Aerospike, ArangoDB, and Oracle NoSQL, which is 6 out of 10) in our survey use protocols based on primary-backup instead of consensus algorithms for state machine replication. Possible reasons for this are that primary-backup is significantly easier to implement than consensus algorithms and the systems using primary-backup for replication mostly only do replication within a cluster of machines in the same data center, making the synchrony assumption valid.

Fourthly, the trade-off between latency and consistency is also an important factor to consider in addition to the one between availability and consistency under network partition. ArangoDB uses Raft only for configuration replication but not data replication, and Oracle NoSQL only uses Paxos for leader election in primary-backup. We suspect that this is because data replication requires larger data transactions and using consensus algorithms will lead to greater latency penalty. In addition, Azure Cosmos DB and Couchbase do not provide strong consistency for replication between faraway clusters and the documentation mentions that this decision is due to latency concerns. Aerospike has an option to provide strong consistency for cross-cluster replication, but it specifically notes in the documentation that latency will be increased.

The last point we want to raise is that there are systems that implement strong consistency using neither consensus algorithms nor primary-backup. Aerospike's replication scheme significantly modifies primary-backup in that it will assign a new node to become a replica whenever a replica fails, and thus can tolerate more than one failure using only two copies of data. OrientDB uses multi-master replication, which usually does not provide strong consistency, but it use special configuration options to achieve strong consistency.

7 Conclusion

In this paper we surveyed the recent progress in consensus algorithms and the application of consensus algorithms in popular key-value databases. The Raft consensus algorithm is a major development of the Paxos algorithm and is shown by the survey results to be indeed more widely used than Paxos in key-value stores. Also, many key-value stores offer strong consistency instead of eventual consistency, which contradicts to the common belief that NoSQL databases are only optimized for speed and availability and care less about consistency. In addition, simpler SMR algorithms like primary-backup is also widely used to provide strong consistency. We hope the results of this survey can motivate researchers to continue working on developing new consensus algorithms since new algorithms are actively used in the industry and help system designers to make decisions on consistency options and implementation when designing new key-value stores.

References

[1] Aerospike. 2020. *Introducing Aerospike's Architecture.* White paper. Aerospike, Inc.

[2] Aerospike, Inc. 2021. *Architecture Overview.* Retrieved December 11, 2021 from https://docs.aerospike.com/docs/architecture/index.html

[3] Aerospike, Inc. 2021. *Strong Consistency mode.* Retrieved December 11, 2021 from https://docs.aerospike.com/docs/architecture/consistency.html

[4] Akmal B. Chaudhri. 2018. *Apache Ignite Transactions Architecture: 2-Phase Commit Protocol.* Retrieved December 11, 2021 from https://www.gridgain.com/resources/blog/apache-ignite-transactions-architecture-2-phase-commit-protocol

[5] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference.* 1–15.

[6] Apache Ignite. 2021. *Apache Ignite Documentation.* Retrieved December 11, 2021 from https://ignite.apache.org/docs/latest/

[7] ArangoDB. 2021. *ArangoDB v3.8.4 Documentation.* Retrieved December 11, 2021 from https://www.arangodb.com/docs/stable/index.html

[8] Yanbei Chen, Xiatian Zhu, and Shaogang Gong. 2017. Person re-identification by deep learning multi-scale representations. In *Proceedings of the IEEE international conference on computer vision workshops.* 2590–2600.

[9] Couchbase. 2020. *Couchbase Under the hood: An Architectural Overview.* White paper. Couchbase, Inc.

[10] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review* 41, 6 (2007), 205–220.

[11] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. 2018. Untangling blockchain: A data processing view of blockchain systems. *IEEE transactions on knowledge and data engineering* 30, 7 (2018), 1366–1385.

[12] etcd Authors. 2021. *etcd.* Retrieved December 11, 2021 from https://etcd.io/

[13] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. 2019. A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications* 126 (2019), 45–58.

[14] Felix Gessert, Wolfram Wingerath, Steffen Friedrich, and Norbert Ritter. 2017. NoSQL database systems: a survey and decision guidance. *Computer Science-Research and Development* 32, 3 (2017), 353–365.

[15] Jing Han, Ee Haihong, Guan Le, and Jian Du. 2011. Survey on NoSQL database. In *2011 6th international conference on pervasive computing and applications.* IEEE, 363–366.

[16] Hazelcast. 2021. *Hazelcast IMDG Reference Manual 4.2.2.* Retrieved December 11, 2021 from https://docs.hazelcast.com/imdg/4.2/

[17] Hazelcast. 2021. *Hazelcast's Replication Algorithm.* Retrieved December 11, 2021 from https://docs.hazelcast.com/imdg/4.2/consistency-and-replication/replication-algorithm

[18] Jianping He, Peng Cheng, Ling Shi, Jiming Chen, and Youxian Sun. 2013. Time synchronization in WSNs: A maximum-value-based consensus approach. *IEEE Trans. Automat. Control* 59, 3 (2013), 660–675.

[19] Robin Hecht and Stefan Jablonski. 2011. NoSQL evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing*. IEEE, 336–341.

[20] Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. 2017. Flexible Paxos: Quorum Intersection Revisited. In *20th International Conference on Principles of Distributed Systems*.

[21] Xin Jin, Ram Krishnan, and Ravi Sandhu. 2012. A unified attribute-based access control model covering DAC, MAC and RBAC. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 41–55.

[22] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40.

[23] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Transactions on Computer Systems* 16, 2 (1998), 133–169.

[24] Heath J LeBlanc, Haotian Zhang, Xenofon Koutsoukos, and Shreyas Sundaram. 2013. Resilient asymptotic consensus in robust networks. *IEEE Journal on Selected Areas in Communications* 31, 4 (2013), 766–781.

[25] Microsoft. 2021. *Azure Cosmos DB | Microsoft Docs*. Retrieved December 11, 2021 from https://docs.microsoft.com/en-us/azure/cosmos-db/

[26] Microsoft. 2021. *Consistency levels in Azure Cosmos DB | Microsoft Docs*. Retrieved December 11, 2021 from https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels

[27] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. 2017. A review on consensus algorithm of blockchain. In *2017 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2567–2572.

[28] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 305–319.

[29] Oracle. 2018. *Oracle NoSQL Database Whitepaper*. Retrieved December 11, 2021 from https://www.oracle.com/technetwork/database/nosqldb/learnmore/nosql-database-498041.pdf

[30] Oracle. 2021. *Oracle NoSQL Database Release 21.2 Concepts*. Retrieved December 11, 2021 from https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/concepts.html

[31] OrientDB Community Group. 2021. *Configuration · OrientDB Manual*. Retrieved December 11, 2021 from https://orientdb.org/docs/3.0.x/distributed/Distributed-Configuration.html

[32] OrientDB Community Group. 2021. *Home · OrientDB Manual*. Retrieved December 11, 2021 from http://www.orientdb.com/docs/last/index.html

[33] OrientDB Community Group. 2021. *Replication · OrientDB Manual*. Retrieved December 11, 2021 from https://orientdb.org/docs/3.0.x/distributed/Replication.html

[34] Radis Labs. 2021. *RedisLabs/redisraft: A Redis Module that make it possible to create a consistent Raft cluster from multiple Redis instances*. Retrieved December 11, 2021 from https://github.com/RedisLabs/redisraft

[35] Redis Ltd. 2021. *Redis*. Retrieved December 11, 2021 from https://redis.io

[36] Software AG. 2021. *Active and Passive Servers*. Retrieved December 11, 2021 from https://documentation.softwareag.com/terracotta/terracotta_10-7/webhelp/terracotta-db-webhelp/co-srv_active_and_passive_servers.html

[37] solid IT gmbh. 2021. *DB-Engines Ranking - Method*. Retrieved November 19, 2021 from https://db-engines.com/en/ranking_definition

[38] solid IT gmbh. 2021. *DB-Engines Ranking - popularity ranking of key-value stores*. Retrieved November 12, 2021 from https://db-engines.com/en/ranking/key-value+store/all

[39] Terracotta, Inc. 2021. *About*. Retrieved December 11, 2021 from https://www.ehcache.org/about/

[40] Terracotta, Inc. 2021. *Clustered Cache*. Retrieved December 11, 2021 from https://www.ehcache.org/documentation/3.9/clustered-cache.html

[41] Terracotta, Inc. 2021. *Terracotta*. Retrieved December 11, 2021 from https://www.terracotta.org/

[42] Marko Vukolić. 2015. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *International workshop on open problems in network security*. Springer, 112–125.

[43] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. 2019. A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access* 7 (2019), 22328–22370.

[44] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 325–338.

[45] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 931–948.

[46] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. 2017. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE international congress on big data (BigData congress)*. IEEE, 557–564.

[47] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. 2018. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services* 14, 4 (2018), 352–375.