The Anatomy of Blockchain Database Systems

Dumitrel Loghin
National University of Singapore
dumitrel@comp.nus.edu.sg

Abstract

Blockchains are around for more than ten years and currently, we are witnessing the adoption of blockchain techniques in databases, and vice-versa. For example, typical blockchain data structures, such as cryptographically-linked blocks and Merkle trees, have been integrated into verifiable databases. On the other hand, database techniques, such as sharding and concurrency control, have been integrated into blockchains. In this paper, we are looking at systems that combine both blockchain and database techniques. We classify these systems into (i) permissioned blockchains, (ii) hybrid blockchain database systems, and (iii) ledger databases. We present their anatomy, including features, techniques, and design choices, by analyzing a few representative systems. In the end, we highlight their limitations and discuss future research directions.

1 Introduction

In the last few years, the line between blockchain systems and distributed databases has been blurred to a certain degree [36, 40]. We have seen the adoption of blockchain techniques in databases. For example, typical blockchain data structures, such as cryptographically-linked blocks [4] and Merkle trees [29], have been integrated into verifiable ledger databases [9, 11, 10, 14] and hybrid blockchain database systems [24]. We have also seen database techniques used in blockchains. For example, sharding is used to scale blockchains [18], while optimistic concurrency control (OCC) is used to decrease the number of aborted transactions [40, 37].

By zooming into the design and implementation of systems that combine blockchain and database techniques, we propose classifying them into three categories. Going from systems that have stronger blockchain features to systems that are closer to databases, these three categories are (i) *permissioned blockchains*, (ii) *hybrid blockchain database systems*, and (iii) *ledger databases*. From a high-level view, all these systems consist of distributed server nodes that communicate via a broadcasting service based on some consensus protocol. Each server node has a ledger (blockchain data structure) and a local database. Both the server nodes and the users (or clients) that interact with these nodes need to be authenticated. The broadcasting service is implemented either with a Crash Fault Tolerant (CFT) consensus protocol, that is closer to distributed databases, or a Byzantine Fault Tolerant (BFT) consensus that resembles typical blockchains.

In this paper, we analyze a few representative systems and present their anatomy in terms of design, techniques, features, and limitations. We shall present more details on our classification in Section 2, and analyze a few systems in Section 3. We present challenges, limitations, and future research directions in Section 4, and conclude in Section 5.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Table 2: Categories, Features, and Examples.

	Permissioned Blockchains	**Hybrid Blockchain Database Systems**	**Ledger Databases**
Administration	Decentralized	Decentralized	Centralized
Broadcasting	CFT or BFT	Typically CFT	CFT
Local Database	Tightly-coupled	Loosely-coupled	Tightly-coupled
Ledger	Replicated	Replicated	Centralized
Examples	Fabric [4] Quorum [3] Corda [27] Diem [10]	Veritas [25] BigchainDB [2] BlockchainDB [19] Blockchain Relational Database [33] ChainifyDB [39] FalconDB [35]	Amazon QLDB [9] Alibaba LedgerDB [10] Microsoft SQL Ledger [11] Spitz [14] Immudb [5] IntegriDB [49]

2 Classification

When analyzing the systems that combine both database and blockchain techniques, we can distinguish three main categories. First, we have *permissioned blockchains* (also known as private, enterprise, or consortium) that have more blockchain features than databases. Second, we have *hybrid blockchain database systems* which can be further classified into out-of-blockchain databases and out-of-database blockchains [36]. Third, we have (centralized) *ledger databases*. Table 2 presents the features of such systems and a few examples of the state-of-the-art for each category.

Permissioned blockchains, as opposed to typical permissionless or private blockchains such as Bitcoin [32] and Ethereum [14], employ authentication for the parties using the blockchain (i.e., clients and peers). They are named permissioned or private blockchains because only authenticated parties can use them. These blockchains are typically used in enterprise setups and they are operated by a consortium of organizations, hence, they are called enterprise or consortium blockchains. In such setups, an organization hosts one or more blockchain peers (or nodes). Since more than one organization is in charge of administrating and operating the blockchain, a permissioned blockchain is a decentralized system where the ledger is replicated on all the nodes (or peers). Initially, some of these permissioned blockchains considered using Byzantine Fault Tolerant (BFT) consensus protocols to replicate the ledger. For example, Hyperledger Fabric v0.6 [19, 20] used PBFT [23] and Quorum provides support for IBFT [38]. However, these BFT protocols degrade the performance of a blockchain concerning throughput and latency [20, 30]. That is why most of the current permissioned blockchains use Crash Fault Tolerant (CFT) consensus mechanisms, such as Raft [34] and Apache Kafka [1].

Hybrid Blockchain Database Systems are very similar to permissioned blockchains but they have different motivations, use cases, and database integration. These systems are motivated by the need of organizations to share a database or parts of a database. In general, this database already exists and it is loosely-coupled to the hybrid blockchain database system. For example, in a supply chain scenario, there should be a shared database with shipping options and costs. Shipping companies update this database, while the other parties just read the data. In such a case, we need a ledger to keep track of the updates in a transparent and tamper-evident way. An authentication mechanism is needed to access the ledger and the broadcasting service. Given this, most of the proposed hybrid blockchain database systems consider only CFT broadcasting services. As expected, if a BFT consensus is used instead, the performance of the system significantly degrades [24].

Ledger Databases are at the other end of the centralized-decentralized administration spectrum since they are hosted and operated by a single organization. In such a centralized model, the users need to trust that organization. To increase the trust, ledger databases use tamper-evident data structures and publish the hashes of the append-only ledger or provide proofs for current states in the database. Such systems may be distributed to increase fault tolerance and improve performance. However, they are not distributed to increase the trust as is the

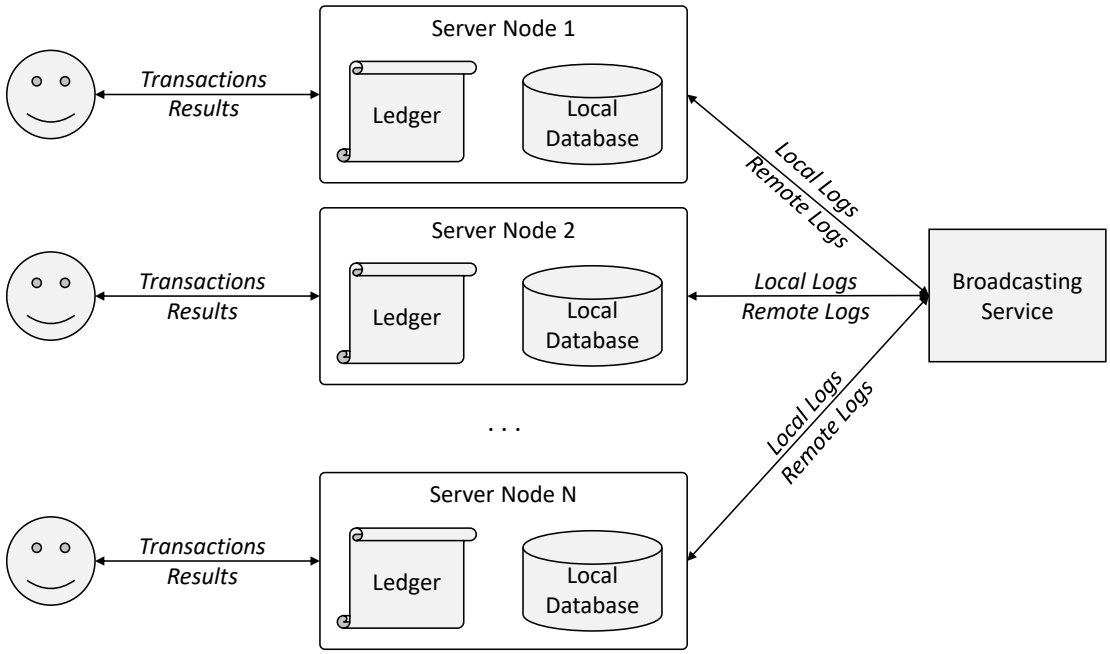


Figure 1: A Generic Hybrid Blockchain Database System

case for the other two categories. Moreover, the database and the ledger are tightly-coupled to the server nodes. While such systems require higher trust from the users, they provide higher performance and zero administration efforts compared to the other two categories.

3 Anatomy

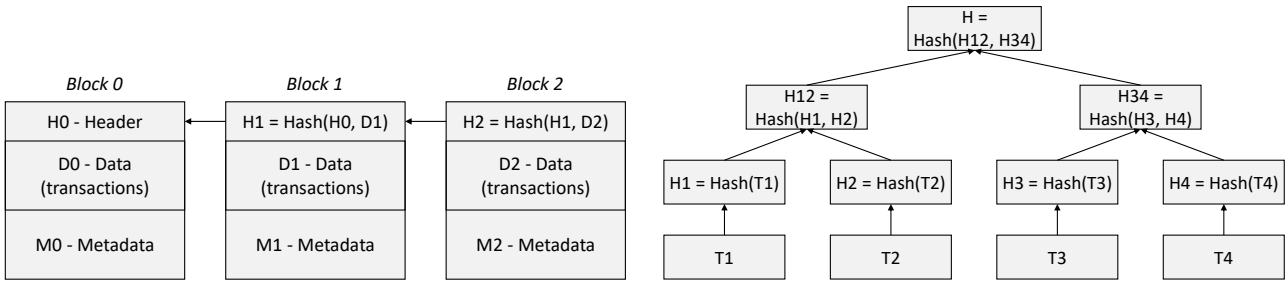
In this section, we start with the similarities among the three proposed categories, after which we present the particularities of each category together with the details of a few representative systems.

3.1 Overview

Typically, systems that combine blockchain and database features have an architecture similar to the one depicted in Figure 1. The system consists of some distributed server nodes (or peers), where each node handles user requests and coordinates with the other nodes via a broadcasting service. The users (or clients) need to be authenticated before sending requests to the nodes. A server node sends local updates (or logs) to and receives remote updates (or logs) from the broadcasting service. This broadcasting service can also be distributed across a few nodes, not necessarily the same as the server nodes. Moreover, the broadcasting service is implemented with a CFT or BFT consensus protocol. For example, the latest version of Fabric uses Raft [34], which is CFT, while Quorum supports, among others, IBFT [38].

Each server node connects to a local database and keeps a copy of the distributed ledger. Note that the local database and the ledger are different. The former keeps the latest version of the data (e.g., states, accounts, assets), while the latter keeps the entire update history using tamper-evident data structures. For example, Fabric uses LevelDB or CouchDB as its local database, which is also called World State. On the other hand, the ledger in Fabric is a linked list of blocks where the header of a block is linked to the header of the previous block using a cryptographic hash. Other systems use data structures based on Merkle trees [29] to represent the ledger.

We briefly compare these two ledger data structure, as illustrated in Figure 2. The hashed blocks data structure,



(a) Hashed Blocks (b) Merkle Tree

Figure 2: Ledger Data Structures

as shown in Figure 2a, is a linked list of blocks where a block points to its predecessor using a cryptographic pointer, except for the first block which is typically called the *genesis* block. Each block consists of data, metadata, and header sections. The data section contains all the transactions that are part of the block. The header is a digest of the block computed using a hashing function. Most of the blockchains use SHA3 or Keccak hashing algorithms. The header of all the blocks except the first one is computed as the hash of the concatenation between the hash of the previous block and the hash of the current block's data section.

A generic Merkle tree, as shown in Figure 2b, is a tree where the leaves are data representing transactions and the internal nodes are hashes. Each parent node contains the hash of the concatenation of all the hashes of its children. Hence, the root node contains the hash that represents all the underlying transactions. We note that Merkle trees can be combined with hashed blocks: the data section of a block can be organized as a Merkle tree. For example, Quorum uses this approach to store transactions in its ledger. In contrast, Fabric does not use a Merkle tree: it just hashes the transaction data as a chunk [9]. We direct the reader to [46] for an analysis of advanced Merkle tree data structures.

3.2 Permissioned Blockchains

Hyperledger **Fabric** [4] is a permissioned blockchain developed by Linux Foundation with a significant contribution from IBM. In Fabric, there are three types of nodes, namely, clients, peers, and orderers. A client sends a transaction request to a set of peers governed by an endorsement policy. For example, the *AND* policy includes all the peers in the network. That is, a client has to send the transaction to and receive endorsements from all the peers. A peer executes the transaction request in simulation mode and creates read and write sets to mark which world states are touched by the transaction. Since this is in simulation mode, the peer does not persist the changes to its local database. Then, the client sends the responses from the peers together with its transaction to the orderers. These orderers pack the transaction in a block and broadcasts the block to all the peers in the network. Currently, Fabric adopts the Raft CFT consensus among orderers. In the last phase, all the peers validate the block and persist the changes of valid transactions to the local database. Note that the peers do not need to re-execute the transaction: they only persist the write set. In the validation phase, the read set is also verified to check if any state has been modified since the transaction was simulated. In such a case, the transaction is aborted. In summary, Fabric implements an execute-order-validate (EOV or XOV) transaction lifecycle, as opposed to many other blockchains that adopt an order-execute (OX) lifecycle. Fabric supports both LevelDB (default) and CouchDB for the world states database. The ledger is stored on the filesystem as a linked list of blocks, where the block headers are linked together via hashing.

Fabric has been extensively analyzed and optimized by the database research community. Many works benchmark and analyze the performance bottlenecks of Fabric [17, 19, 20, 30, 31, 42]. In our recent work [31], we show that a Fabric network with up to 10 peers can reach around 1,000 transactions-per-second (TPS). Other works improve the rate of aborted transactions by relaxing the concurrency model (e.g., using optimistic

concurrency control) and by re-ordering the transactions [37, 40].

Quorum [3] is a permissioned blockchain that draws its source code from Ethereum (implemented in the Go programming language). Naturally, Quorum supports Solidity smart contracts, but it replaces the energy-inefficient Proof-of-Work consensus with a few alternatives, out of which Raft is the default. Besides Raft, Quorum also supports IBFT (Istanbul BFT), QBFT (Quorum BFT), and Clique Proof-of-Authority (POA). IBFT is inspired by PBFT [23], while QBFT is an optimized version of IBFT which is also interoperable with Hyperledger Besu, an Ethereum client developed by the Hyperledger Foundation. As opposed to Fabric, Quorum has only peers and clients and adopts the traditional order-execute (OX) transaction lifecycle. That is, a transaction is first grouped into a block and then executed by each peer in the network. Similar to Fabric, Quorum uses LevelDB as its local database, but it adopts Merkle Patricia Trie for the ledger. In our recent work [31], we show that Quorum with Raft exhibits a throughput of 250 TPS, which is relatively low for a permissioned blockchain.

Corda [27] is advertised as a distributed ledger technology (DLT) for enterprises. For that reason, it is built on Java and Kotlin so it can be better integrated with existing Java enterprise systems. Besides nodes, a Corda network has notaries which are responsible for validating transactions in terms of uniqueness and validity. In essence, uniqueness prevents double-spending, while validity means that the transaction passes the input-output tests and it has all the required signatures. Notaries use a consensus protocol which is Raft-based in the default version of Corda. This default version uses H2, a relational database management system written in Java, for the local database. The ledger uses a custom version of Merkle trees to hide transaction details from the entities that are not involved in the transaction. A recent publication shows that the performance of Corda is very low, at 15 TPS [26]. Even when a single notary is used to minimize the impact of consensus, the performance is low due to a synchronous (blocking) transaction processing mechanism [26].

Diem [10] is a permissioned blockchain that was developed by a consortium of companies led by Facebook. It was previously known as the *Libra* blockchain [10]. The entire project has been discontinued in 2022. However, Diem implements some powerful features which are worth mentioning. For example, it uses LibraBFT [12], a BFT consensus based on Hotstuff [45] which further improves PBFT [23]. For the ledger, Diem uses Jellyfish Merkle tree [23] which is a sparse Merkle tree inspired by the Merkle Patricia Trie used in Ethereum. RocksDB [21], a fast key-value store derived from LevelDB and developed by Facebook, is used as the underlying database. A recent study shows that Diem achieves around 600 TPS on 4 nodes [47], which is a decent performance for a BFT-based blockchain.

3.3 Hybrid Blockchain Database Systems

Veritas [25] is an out-of-blockchain database that consists of a shared database (or table) and a blockchain ledger for keeping auditable and verifiable updates done on the shared database. Each node is operated by an organization. A node uploads its local update logs and downloads remote update logs to and from a broadcasting service. Veritas employs a concurrency control mechanism based on timestamps. The timestamp of a transaction represents the sequence number of that transaction in the log. A transaction is first verified locally by the node receiving it. If it passes the verification (e.g., multi-version concurrency control – MVCC), it is included in the logs and sent to the broadcasting service. Once the other nodes agree to the updates, they send acknowledgments, and once every node receives the acknowledgments, it persists the updates to the local database and appends them to the ledger. Note that this mechanism incurs $O(N^2)$ communication complexity [24].

The original design of Veritas uses Redis [15], an in-memory NoSQL database, and Apache Kafka [1], a CFT broadcasting service. The re-implementation of Veritas in [24] achieves around 30,000 TPS, making it the fastest system among all those analyzed in this paper.

BlockchainDB [19] is an out-of-blockchain database with prominent blockchain features: it is a shared database built over a blockchain. It is the only hybrid blockchain database that uses sharding to partition the shared database. Firstly, the blockchain represents the storage layer of a BlockchainDB node. By default, BlockchainDB uses Ethereum, but other blockchains can be used as well via a plugin interface. With Ethereum, the ledger

structure is based on Merkle Patricia Trie. Secondly, a node has a database layer with a simple key-value interface. Thirdly, there is a shard manager that helps the database layer to identify the shard where a specific key is stored. Due to the use of such a slow blockchain, like Ethereum with Proof-of-Work (PoW) or Proof-of-Authority (PoA), BlockchainDB exhibits a throughput of around 50 TPS [24].

FalconDB [35] is another out-of-blockchain database that starts from a blockchain and provides a shared database to the clients. Different from other systems, FalconDB provides a relational database interface to the clients. In FalconDB, both the clients and the peers need to keep a digest of the data. The difference is that clients only keep the blockchain headers to save storage space. However, these headers are sufficient for checking the correctness of the data queried from the peers. FalconDB uses IntegriDB [49], a verifiable SQL database, to store the ledger, Tendermint for consensus, and MySQL as the local database. The throughput of the system with a YCSB write-heavy workload (50% reads and 50% writes) is around 3,000 TPS [35]. Note that a similar YCSB workload is used to evaluate Veritas, BigchainDB, and BlockchainDB [24].

Blockchain Relational Database (**BRD**) [33] has a similar design to Veritas, but it starts from a PostgreSQL [7] relational database. In this sense, BRD is an out-of-database blockchain. Also, different from Veritas, the broadcasting service orders blocks of transactions (updates) and does not serialize the transactions in a block. To speedup transaction execution, BRD implements concurrent execution with Serializable Snapshot Isolation (SSI). Note that BRD uses PostgreSQL [7] as its local database, which supports Serializable Snapshot Isolation. BRD also uses Apache Kafka as the broadcasting service. Different from Veritas, BRD keeps the ledger in the same relational database, namely PostgreSQL. According to the BRD paper [33], the system achieves a throughput of 2,500 TPS with a key-value workload.

BigchainDB [2] is another out-of-database blockchain. It starts from MongoDB [6], a NoSQL database, used as the local database. By using MongoDB, the main data abstraction in BigchainDB is an asset, represented in JSON format. Otherwise, the transaction lifecycle is similar to the one in Veritas. A transaction is verified locally by a node, then a request is sent to the broadcasting service. BigchainDB uses a BFT consensus middleware as the broadcasting service, namely Tendermint [13]. Once the majority of the nodes agree on the transaction, it is committed in the local database. BigchainDB relies on Tendermint to keep the ledger in the form of a Merkle tree. Our evaluation of the open-source BigchainDB code shows a maximum performance of around 200 TPS with the YCSB workloads.

ChainifyDB [39] is an out-of-database blockchain that starts from a relational database which can be either PostgreSQL or MySQL. Apache Kafka is used for broadcasting the transactions which are SQL statements. The ledger uses a custom representation based on LedgerBlocks. A LedgerBlock contains all the transactions that are part of a block, where a transaction is in its SQL form. Next, the LedgerBlock contains a list of bits representing the successful transactions, a SHA256 hash digest over the data changed by the transactions, and a hash value of the previous LedgerBlock that was added to the ledger. This representation is similar to the one used by Fabric. ChainifyDB achieves a throughput of around 1,000 TPS on three nodes using the SmallBank workload when all three nodes need to reach consensus. When only two out of three nodes need to reach consensus, the throughput increases to around 5,000 TPS [39].

3.4 Ledger Databases

Amazon Quantum Ledger Database **QLDB** [9] is a verifiable database developed by Amazon and provided as a cloud service. QLDB follows the structure depicted in Figure 1 by integrating a relational database and a ledger in its server node. The database keeps the current states and the history of those states, while the ledger is an append-only journal that keeps track of all the changes done to the database in an immutable way. While it is not clear what is the underlying database, the ledger in QLDB is implemented based on Merkle trees. Our preliminary evaluation of QLDB shows a throughput of 10,000 TPS, which is relatively low for a centralized system. However, we note that an update in QLDB changes both the database and the ledger, and these two changes are done sequentially.

Table 3: Summary of Systems, Features, and Performance

System	Broadcasting Service	Ledger Structure	Local Database	Throughput [TPS]
Fabric [4]	Raft (CFT)	Linked Blocks	LevelDB	1,000 [31]
Quorum [3]	Raft (CFT)	Merkle Patricia Trie	LevelDB	250 [31]
Corda [27]	Raft (CFT)	Merkle Tree	H2	*10* [26]
Diem [10]	LibraBFT (BFT)	Jellyfish Merkle Tree	RocksDB	*600* [47]
Veritas [25]	Kafka (CFT)	Sparse Merkle Tree	Redis	30,000 [24]
BlockchainDB [19]	PoW/PoA (BFT)	Merkle Patricia Trie	Ethereum(LevelDB)	50 [24]
FalconDB [35]	Tendermint (BFT)	Merkle Tree(IntegriDB)	MySQL	*3,000* [35]
BRD [33]	Kafka (CFT)	Relational	PostgreSQL	*2,500* [33]
BigchainDB [2]	Tendermint (BFT)	Merkle Tree(Tendermint)	MongoDB	200 [24]
ChainifyDB [39]	Kafka (CFT)	LedgerBlock	PostgreSQL/MySQL	*1,000* [39]
QLDB [9]	N/A	Merkle Tree	N/A	10,000
LedgerDB [10]	Master-Workers (CFT)	Merkle Tree	L-Stream	20,000
SQL Ledger [11]	N/A	Merkle Tree	SQL Server	*70,000* [11]
Spitz [14]	2PC + timestamp (CFT)	Merkle Tree	ForkBase	*70,000* [14]

LedgerDB [10] is a verifiable database developed by Alibaba and provided as a cloud service. LedgerDB updates the ledger, which is based on a Merkle tree, asynchronously. Specifically, the transactions are batched and the Merkle tree is updated with the batched transactions. Hence, this approach is called batch accumulated Merkle tree (bAMT). LedgerDB supports multiple underlying storage engines, but L-Stream, a custom storage developed by Alibaba, is the default one. L-Stream is an append-only filesystem created specifically for LedgerDB. In terms of distributed architecture, the server nodes in LedgerDB are coordinated by a master that ensures CFT and workload balancing. Our preliminary evaluation of LedgerDB shows a throughput of 20,000 TPS, two times higher compared to QLDB.

SQL Ledger [11] is a ledger database developed by Microsoft and offered as a service on its Azure cloud. It has a similar architecture to QLDB and LedgerDB, but it uses Microsoft's SQL Server as the underlying storage engine. SQL Ledger keeps a ledger data structure based on Merkle trees and two tables, namely, the Ledger Table and the History Table. The Ledger Table reflects the latest record for a given key, while the History Table records the previous version of that record. It is not clear what type of consensus is used to coordinate among multiple nodes in SQL Ledger. Moreover, the reported evaluation was done on a single server with 72 cores [11]. In this evaluation, SQL Ledger achieves a throughput of 70,000 TPS with TPC-C workloads. It is expected to see lower SQL Ledger performance in a distributed setting.

Spitz [14] is a verifiable database that uses ForkBase [43] at the storage level. The authors identify the source of low performance in the other systems as being the existence of separate sub-systems for the ledger and database. Hence, Spitz relies on ForkBase for both the ledger and database. Spitz consists of multiple transaction processing nodes and a common ForkBase backend. The processing nodes coordinate via a two-phase commit (2PC) protocol. A global timestamp service is used to ensure the order of the transactions. Hence, Spitz is a CFT system. The ledger is implemented in ForkBase with the help of a data structure inspired by Merkle trees. Spitz is evaluated on a key-value store application and it achieves up to 70,000 TPS on write operations with 10,000 records. The performance degrades to about 10,000 TPS with more than one million records [14].

3.5 Summary

We summarize the features and the performance of the systems analyzed in this paper in Table 3. In this table, the values in italic are taken from their respective papers, while the other values are based on our measurements [24, 31]. Note that for ledger databases, the *Broadcasting Service* feature is not accurate. However, we list the mechanisms used by the systems for coordination among distributed nodes under this feature. In the

next section, we present the limitations of current systems and some challenges in designing and implementing systems that combine blockchain and database features.

4 Challenges and Limitations

When analyzing the systems that combine blockchain and database features, we observe the lack of open-source code for most of the hybrid blockchain databases and ledger databases. Hence, it is difficult to understand the exact implementation and to assess the performance of these systems. In our previous work [24], we re-implemented Veritas and BlockchainDB in a modular way that allows us to replace some of the components, such as the consensus mechanism and the local database. However, more needs to be done to achieve an open-source, flexible and modular hybrid blockchain database system where the consensus and the underlying database can be replaced in a plug and play manner.

At the same time, such systems should offer both key-value and relational interfaces to the users. We note that most of the existing systems offer simple key-value interfaces, with the exception of FalconDB, ChainifyDB, QLDB, and SQL Ledger. It remains to be analyzed what is the impact of having a flexible user interface on performance. For example, what is the impact of having a relational interface when the underlying database is NoSQL? For such designs, the server node needs to be flexible enough and yet exhibit good performance.

Regardless of a BFT or CFT consensus, sharding could be used to improve the scalability and performance of such hybrid systems. So far, only BlockchainDB considers sharding, but its use of blockchain as the underlying storage hinders its performance. In one of our previous works [18], we used sharding to scale Fabric v0.6 in a Byzantine environment. We have shown that Fabric can scale to around 1,000 nodes distributed world-wide, while achieving a performance of around 4,000 TPS. However, sharding comes with the downside of managing cross-shard transactions which have a negative impact on performance.

Last but not least, the effect of newer BFT consensus protocols or optimizations should be evaluated. The existing systems either use a version of PBFT [23] or Tendermint [13]. New consensus frameworks, such as HotStuff [45], Basil [41], and Leopard [28] among others, are claiming much higher throughput compared to PBFT. It remains to be analyzed if such systems can improve the performance of blockchains or hybrid systems. For example, Hotstuff claims more than 100,000 operations per second, while our evaluation of Veritas with Apache Kafka exhibits 30,000 TPS. If we replace Kafka with Hotstuff, can we achieve at least the same performance of 30,000 TPS?

5 Conclusions

In this paper, we analyzed systems that combine both blockchain and database techniques. We classify these systems into three categories, namely, (i) permissioned blockchains, (ii) hybrid blockchain database systems, and (iii) ledger databases. While sharing a similar architecture, each category and each system in a category has its own particularities. We then analyzed a few representative systems, such as Fabric [4], Quorum [3], Veritas [25], QLDB [9], and LedgerDB [10], among others. The exact performance of these systems is hard to evaluate due to the lack of open-source code. On the other hand, existing implementations are not flexible and modular enough. By designing and implementing a modular system where the user interface, consensus, and local storage are plug and play, we could answer more of the existing questions. For example, can we replace a CFT broadcasting framework with a newer BFT consensus framework while experiencing no performance loss? Such questions remain to be answered in the future.

Acknowledgements

This research/project is supported by the National Research Foundation, Singapore under its Emerging Areas Research Projects (EARP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. We would like to thank Prof. Beng Chin Ooi, Prof. Tien Tuan Anh Dinh, Dr. Pingcheng Ruan, Tianwen Wang, and Cong Yue for their help with this project.

References

[1] Apache Kafka, https://kafka.apache.org/, 2017.

[2] BigchainDB 2.0 The Blockchain Database, Technical report, 2018.

[3] GoQuorum, https://github.com/ConsenSys/quorum, 2021.

[4] Hyperledger Fabric, https://www.hyperledger.org/use/fabric, 2021.

[5] immudb, https://codenotary.io/technologies/immudb/, 2021.

[6] MongoDB, https://www.mongodb.com/, 2021.

[7] PostgreSQL, https://www.postgresql.org/, 2021.

[8] Amazon Quantum Ledger Database (QLDB), https://aws.amazon.com/qldb/, 2022.

[9] Hyperledger Fabric Ledger, https://archive.ph/edzMi, 2022.

[10] Z. Amsden, et al., The Diem Blockchain, https://archive.ph/1xfcy, 2021.

[11] P. Antonopoulos, R. Kaushik, H. Kodavalla, S. Rosales Aceves, R. Wong, J. Anderson, J. Szymaszek, *SQL Ledger: Cryptographically Verifiable Data in Azure SQL Database*, page 2437–2449, 2021.

[12] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, A. Sonnino, State Machine Replication in the Libra Blockchain, https://archive.ph/Uxlb3, 2019.

[13] E. Buchman, *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*, PhD thesis, The University of Guelph, 2016.

[14] V. Buterin, A Next-Generation Smart Contract and Decentralized Application Platform, http://archive.fo/Sb4qa, 2013.

[15] J. Carlson, *Redis in Action*, Manning Shelter Island, 2013.

[16] M. Castro, B. Liskov, Practical Byzantine Fault Tolerance and Proactive Recovery, *ACM Transactions on Computer Systems (TOCS)*, 2002.

[17] J. A. Chacko, R. Mayer, H.-A. Jacobsen, *Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric*, page 221–234, 2021.

[18] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, B. C. Ooi, Towards Scaling Blockchain Systems via Sharding, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 123–140, 2019.

[19] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, J. Wang, Untangling Blockchain: A Data Processing View of Blockchain Systems, *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1366–1385, 2018.

[20] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, K.-L. Tan, BLOCKBENCH: A Framework for Analyzing Private Blockchains, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 1085–1100, 2017.

[21] S. Dong, A. Kryczka, Y. Jin, M. Stumm, RocksDB: Evolution of Development Priorities in a Key-Value Store Serving Large-Scale Applications, *ACM Trans. Storage*, 17(4), oct 2021.

[22] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, R. Ramamurthy, Blockchaindb: A Shared Database On Blockchains, *Proc. VLDB Endow.*, 12(11):1597–1609, 2019.

[23] Z. Gao, Y. Hu, Q. Wu, Jellyfish Merkle Tree, `https://archive.ph/s7pPF`, 2019.

[24] Z. Ge, D. Loghin, B. C. Ooi, P. Ruan, T. Wang, Hybrid Blockchain Database Systems: Design and Performance, *Proc. VLDB Endow.*, 15(5):1092–1104, 2022.

[25] J. Gehrke, L. Allen, P. Antonopoulos, A. Arasu, J. Hammer, J. Hunter, R. Kaushik, D. Kossmann, R. Ramamurthy, S. T. V. Setty, J. Szymaszek, A. van Renen, J. Lee, R. Venkatesan, Veritas: Shared Verifiable Databases and Tables in the Cloud, *Proc. of 9th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2019.

[26] R. Han, G. Shapiro, V. Gramoli, X. Xu, On the Performance of Distributed Ledgers for Internet of Things, *Internet of Things*, 10:100087, 2020, Special Issue of the Elsevier IoT Journal on Blockchain Applications in IoT Environments.

[27] M. Hearn, R. G. Brown, Corda: A Distributed Ledger, `https://bit.ly/3iLajrI`, 2019.

[28] K. Hu, K. Guo, Q. Tang, Z. Zhang, H. Cheng, Z. Zhao, Leopard: Towards High Throughput-Preserving BFT for Large-scale Systems, 2021.

[29] L. Liu, M. T. Özsu, editors, *Merkle Trees*, pages 1714–1715, Springer US, 2009.

[30] D. Loghin, G. Chen, T. T. A. Dinh, B. C. Ooi, Y. M. Teo, Blockchain Goes Green? An Analysis of Blockchain on Low-Power Nodes, 2019.

[31] D. Loghin, T. T. A. Dinh, A. Maw, C. Gang, Y. M. Teo, B. C. Ooi, Blockchain Goes Green? Part II: Characterizing the Performance and Cost of Blockchains on the Cloud and at the Edge, 2022.

[32] S. Nakamoto, Bitcoin: A Peer-to-peer Electronic Cash System, http://archive.fo/CIl1Y, 2008.

[33] S. Nathan, C. Govindarajan, A. Saraf, M. Sethi, P. Jayachandran, Blockchain Meets Database: Design And Implementation Of A Blockchain Relational Database, *Proc. VLDB Endow.*, 12(11):1539–1552, 2019.

[34] D. Ongaro, J. Ousterhout, In Search of an Understandable Consensus Algorithm, *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, 2014. USENIX Association.

[35] Y. Peng, M. Du, F. Li, R. Cheng, D. Song, Falcondb: Blockchain-Based Collaborative Database, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 637–652, 2020.

[36] P. Ruan, T. T. A. Dinh, D. Loghin, M. Zhang, G. Chen, Q. Lin, B. C. Ooi, Blockchains vs. Distributed Databases: Dichotomy and Fusion, *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 1–14, 2021.

[37] P. Ruan, D. Loghin, Q.-T. Ta, M. Zhang, G. Chen, B. C. Ooi, A Transactional Perspective on Execute-Order-Validate Blockchains, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 543–557, 2020.

[38] R. Saltini, D. Hyland-Wood, Correctness Analysis of IBFT, 2019.

[39] F. M. Schuhknecht, A. Sharma, J. Dittrich, D. Agrawal, ChainifyDB: How to get rid of your Blockchain and use your DBMS instead, *Proc. of 11th Conference on Innovative Data Systems Research (CIDR)*, 2021.

[40] A. Sharma, F. M. Schuhknecht, D. Agrawal, J. Dittrich, Blurring the Lines between Blockchains and Database Systems: The Case of Hyperledger Fabric, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 105–122, 2019.

[41] F. Suri-Payer, M. Burke, Z. Wang, Y. Zhang, L. Alvisi, N. Crooks, Basil: Breaking up BFT with ACID (Transactions), *Proc. of ACM SIGOPS 28th Symposium on Operating Systems Principles*, page 1–17, 2021.

[42] P. Thakkar, S. Nathan, B. Viswanathan, Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform, *Proc. of IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 264–276, 2018.

[43] S. Wang, T. T. A. Dinh, Q. Lin, Z. Xie, M. Zhang, Q. Cai, G. Chen, B. C. Ooi, P. Ruan, Forkbase: An Efficient Storage Engine for Blockchain and Forkable Applications, *Proc. VLDB Endow.*, 11(10):1137–1150, 2018.

[44] X. Yang, Y. Zhang, S. Wang, B. Yu, F. Li, Y. Li, W. Yan, LedgerDB: A Centralized Ledger Database for Universal Audit and Verification, *Proc. VLDB Endow.*, 13(12):3138–3151, 2020.

[45] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, I. Abraham, HotStuff: BFT Consensus with Linearity and Responsiveness, *Proc. of 2019 ACM Symposium on Principles of Distributed Computing*, page 347–356, 2019.

[46] C. Yue, Z. Xie, M. Zhang, G. Chen, B. C. Ooi, S. Wang, X. Xiao, Analysis of Indexing Structures for Immutable Data, *Proc. of ACM SIGMOD International Conference on Management of Data*, page 925–935, 2020.

[47] J. Zhang, J. Gao, Z. Wu, W. Yan, Q. Wu, Q. Li, Z. Chen, Performance Analysis of the Libra Blockchain: An Experimental Study, 2019.

[48] M. Zhang, Z. Xie, C. Yue, Z. Zhong, Spitz: A Verifiable Database System, *Proc. VLDB Endow.*, 13(12):3449–3460, 2020.

[49] Y. Zhang, J. Katz, C. Papamanthou, IntegriDB: Verifiable SQL for Outsourced Databases, *Proc. of 22nd ACM SIGSAC Conference on Computer and Communications Security*, page 1480–1491, 2015.