ISSN No:-2456-2165

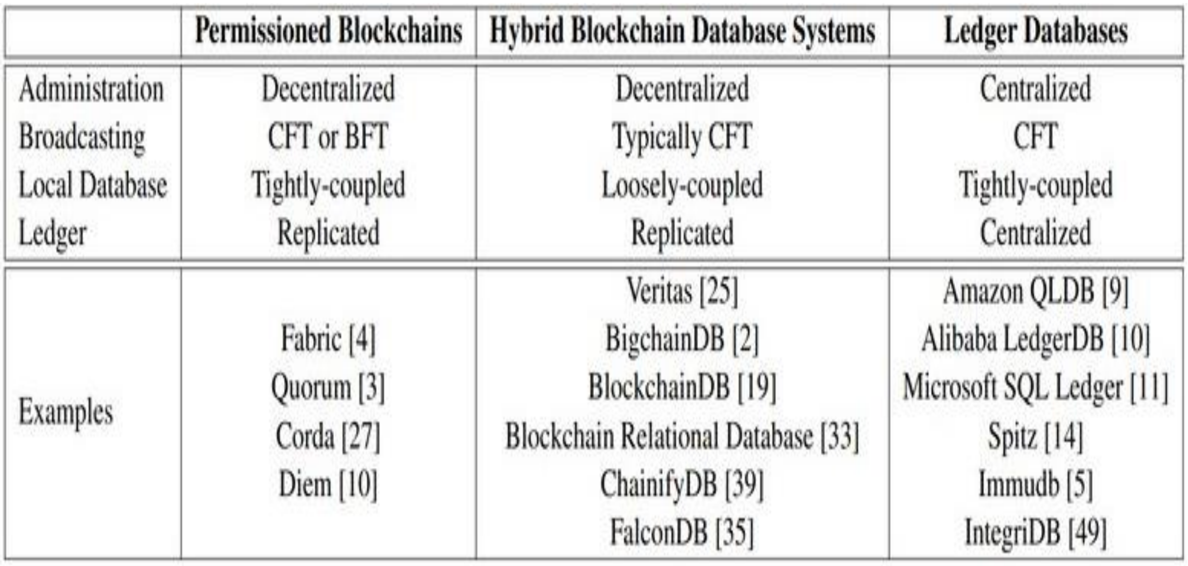
The Anatomy of Blockchain Database Systems Blockchain Database System

Aditi Morey, Akanksha Kulkarni
School of Engineering Ajeenkya DY Patil University,
Pune, India-412105

Abstract:- Blockchains are here for more than ten-fifteen years and currently, we are adopting the blockchain techniques in databases, and vice-versa. Example, A typical blockchain data structures, such as cryptographically-linked blocks and Merkle trees, have been interspersed. Into verifiable databases. On the other hand, database techniques, such as sharding and concurrency control, have been interspersed. Into blockchains. In this paper, I am looking at systems that combine both blockchain and database techniques. I classify these systems into three types and that is given below (1) Permissioned Blockchains, (2) Hybrid Blockchain Database Systems, (3) Ledger Databases. I also present their anatomy, including the features, techniques, and design choices, by analyzing a few representative systems. In the end, I highlight their challenges and discuss research directions

I. INTRODUCTIONS

In the last many years, the line between blockchain systems and distributed databases has been disappeared to a certain degree. We have seen adopting the blockchain techniques in databases. Example, blockchain data structures, such as cryptographically-linked blocks and Merkle trees , have been interspersed into verifiable ledger databases and hybrid blockchain database systems. And we have also seen the database techniques used in blockchains. For example, shading is used to scale blockchains, while the optimistic concurrency control (OCC) is used to decrease the number of aborted transactions. By focusing on the design and implementation of systems that combine blockchain and database techniques, we are classifying them into three categories. Going from the systems that have strong blockchain features to the systems that are very closer to the databases, these three categories are as follows:- (1) permissioned blockchains, (2) hybrid blockchain database systems, and (3) ledger databases. From a effective view, all these systems consist of the 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.



	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]

Table 1: Categories, Features and Examples

II. CLASSIFICATION

When analyzing the systems that mix each info and blockchain techniques, we will distinguish 3 main categories. First, we've permissioned blockchains (also referred to as non-public, enterprise, or consortium) that have a lot of blockchain options than databases. Second, we have hybrid blockchain database systems which may be more classified into out-of-blockchain databases and out-of- database blockchains. Third, we have (centralized) ledger databases. Table a pair of presents the features of such systems and many samples of the progressive for every class Permissioned blockchains, as critical typical permission less or private blockchains comparable to Bitcoin and Ethereum, use authentication for the parties using the blockchain (i.e., shoppers and peers). they're named permissioned or non-public blockchains as a result of solely echt parties will use them. These blockchains are generally utilized in enterprise setups and that they are operated by a association of organizations, hence, they are known as enterprise or consortium blockchains. In such setups, a corporation hosts one or a lot of blockchain peers (or nodes). Since quite one organization is responsible of administrating and operational the blockchain, a permissioned blockchain may be a redistributed system wherever the ledger is replicated on all the nodes (or peers). Initially, a number of these permissioned blockchains thought of victimisation Byzantine Fault Tolerant (BFT) agreementprotocols to copy the ledger. For example, Hyperledger material v0.6 used PBFT and gathering provides support for IBFT . However, these BFT protocols degrade the performance of a blockchain regarding outturn and latency. that's why most of this permissioned blockchains use Crash Fault Tolerant (CFT) consensus mechanisms, comparable to Raft and Apache Kafka. Hybrid Blockchain info Systems are terribly similar to permissioned blockchains however they need totally different motivations, use cases, and database integration. These systems are actuated by the necessity of organizations to share a database or elements of a database. In general, this info already exists and it's loosely-coupled to the hybrid blockchain database system. For example, during a provide chain scenario, there ought to be a shared database with shipping choices and costs. Shipping corporations update this database, whereas the opposite parties simply scan the data. In such a case, we want a ledger to stay track of the updates in a clear and tamper-evident way. Associate in Nursing authentication mechanism is required to access the ledger and also the broadcasting service. Given this, most of the projected hybrid blockchain database systems think about solely CFT broadcasting services. As expected, if a BFT agreement is used instead, the performance of the system considerably degrades.

Ledger Databases are at the opposite finish of the centralized-decentralized administration spectrum since they're hosted and operated by one organization. In such a centralized model, the users have to be compelled to trust that organization. to extend the trust, ledger knowledgebases use tamper- evident data structures and publish the hashes of the append-only ledger or give proofs for current states within the database. Such systems is also distributed to increase fault tolerance and improve performance. However, they are not distributed to increase the trust as is that the 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.

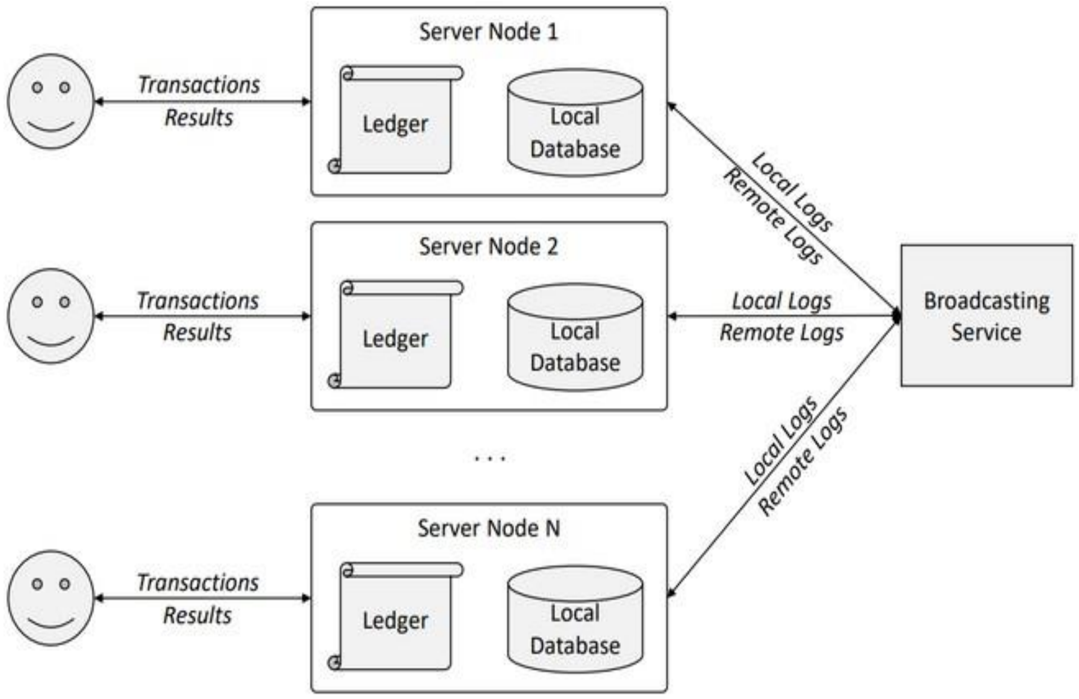


Fig. 1: A Generic Hybrid Blockchain Database System

III. TYPES OF BLOCKCHAIN

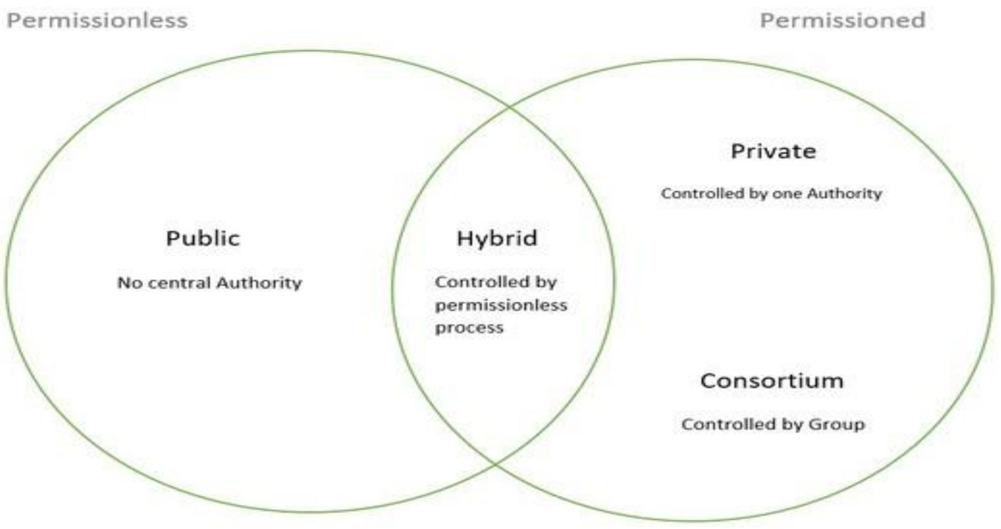


Fig. 2: The types of blockchain

IV. ANATOMY

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

➢ *Overview*

Typically, systems that combines the blockchain and database features they have the similar architecture to the one depicted in Figure 1. The system consists the same distributed server nodes, where each and every node handles user requests and also coordinates with the other nodes via a broadcasting service. The clients need to be authenticated before sending the requests to the nodes. A server node sends local updates and receives remote updates from the broadcasting service. This broadcasting service can also be distributed across a few nodes, it is not necessary that they are 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, which is CFT, while Quorum supports, among others, IBFT. 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 to represent the ledger. Figure 2 shows the types of blockchain. We briefly compare these two ledger data structure, as illustrated in Figure 3.

The hashed blocks data structure,

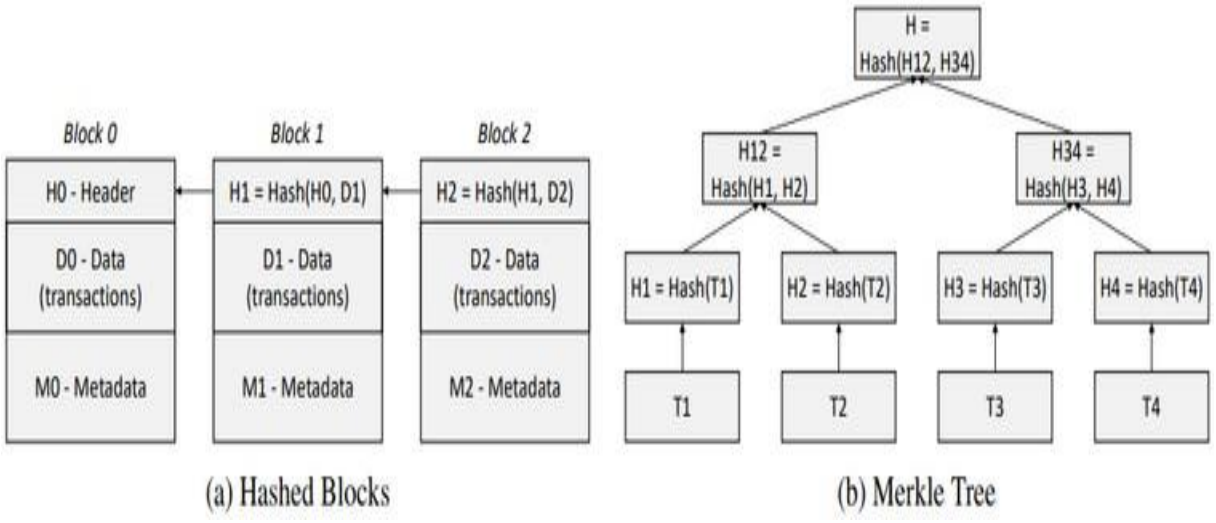


Fig. 3: Ledger Data Structures

As shown in Figure 3 it is a linked list of blocks where a block points to its predecessor using a cryptographic pointer, except the first block which is called as the genesis block. Each block consists of data, metadata, and header sections. The data section contains all the transactions that are the part of the block. The header is a digest of the block add up using a hashing function. Lion's share of the blockchains use SHA3 or Keccak hashing algorithms. The header of all the blocks except the first one is added up 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 3. It 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. We direct the reader to for an analysis of advanced Merkle tree data structures.

V. PERMISSION BLOCKCHAIN

Hyperledger Fabric is a permissioned blockchain developed by the Linux Foundation with significant input from IBM. There are three types of nodes in Fabric namely Clients, Peers and Orderers. A client sends a transaction request to a set of peers that are subject to an approval policy. For example, the AND policy includes all peers in the network. That is, a customer must submit the transaction and receive confirmations from all peers. A peer runs the transaction request in mock mode and creates read and write arrays to mark which world states are affected by the transaction. Because it is in simulation mode, the peer does not save the changes to its local database. The client then sends the peer responses to the payers along with its transaction. These managers pack the transaction into a block and broadcast the block to all peers in the network. Fabric is currently taking the Raft CFT consensus between orders. In the last phase, all peers validate the block and keep valid transaction changes in the local database. Note that the peers don't need to redo the transaction: they just keep the write set. The validation phase also checks the read record to see if any state has changed since the transaction was simulated. In such a case, the transaction is aborted. In short, Fabric implements an Execute-Order-Validate (EOV or XOV) transaction lifecycle, unlike many other blockchains that adopt an Order-Execute (OX) lifecycle. Fabric supports Level DB (default) and CouchDB for world states database. The ledger is stored in the file system as a linked list of blocks, with the block headers linked together using hashes. Fabric has been extensively tested and optimized by the database research community. Many works compare and analyze fabric performance bottlenecks. In our recent work, we show that a fabric with up to 10 peers can achieve around 1,000 transactions per second (TPS). Other work improves the abandonment rate by relaxing the concurrency model (e.g. through optimistic concurrency controls) and by rearranging transactions. Quorum is a permissioned blockchain that gets its source code from Ethereum (implemented in the Go programming language). Of course, Quorum supports Solidity's smart contracts, but it replaces the energy-inefficient proof-of-work consensus with a few alternatives, of which Raft is the default. In addition to Raft, Quorum also supports IBFT (Istanbul BFT), QBFT (Quorum BFT), and Clique Proof- of-Authority (POA). IBFT is inspired by PBFT, while QBFT is an optimized version of IBFT that is also interoperable with Hyperledger Beau, 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 Level DB as its local database, but it adopts Merkle Patricia Trie for the ledger. In our recent work, we show that Quorum with Raft exhibits a throughput of 250 TPS, which is relatively low for a permissioned blockchain. Corda 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. 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. Diem is a permissioned blockchain that was developed by a consortium of companies led by Facebook. It was previously known as the Libra blockchain. The entire project has been discontinued in 2022. However, Diem implements some powerful features which are worth mentioning. For example, it uses Libra FT , a BFT consensus based on Hotstuff which further improves PBFT. For the ledger, Diem uses Jellyfish Merkle tree which is a sparse Merkle tree inspired by the Merkle Patricia Trie used in Ethereum. Rocks DB, a fast key-value store derived from Level DB and developed by Facebook, is used as the underlying database. A recent study shows that Diem achieves around 600 TPS on 4 nodes, which is a decent performance for a BFT-based blockchain.

VI. HYBRID BLOCKCHAIN DATABASE SYSTEM

- **Veritas** 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(N2) communication complexity. The original design of Veritas uses Redis, an in-memory NoSQL database, and Apache Kafka, a CFT broadcasting service. The re-implementation of Veritas in achieves around 30,000 TPS, making it the fastest system among all those analyzed in this paper.
- **Blockchain DB** 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 shading to partition the shared database. Firstly, the blockchain represents the storage layer of a Blockchain DB node. By default, Blockchain DB uses Ethereum, but other blockchains can be used as well via a plugin interface. With Ethereum, the ledger

  The structure is based on Merkle Patricia Trie. Second, a node has a database layer with a simple key-value interface. Third, there is a shard manager that helps the database layer identify the shard where a particular key is stored. Due to the use of such a slow blockchain as Ethereum with Proof of Work (Pow) or Proof of Authority (PoA), Blockchain DB has a performance of around 50 TPS. Falcon DB is another off-chain database that starts on a blockchain and provides clients with a shared database. Unlike other systems, Falcon DB Clients offers a relational database interface. In
- **Falcon DB**, both clients and peers must maintain a summary of data. The difference is that clients only keep the blockchain headers to save storage space. However, these headers are sufficient to verify the correctness of the requested data from the peers. Falcon DB uses IntegriDB, an auditable SQL database, for general ledger storage, Tendermint for consensus, and MySQL as the local database. System performance on a write-intensive YCSB workload (50% reads and 50% writes) is around 3000 TPS. Note that a similar YCSB workload is used to evaluate Veritas, BigchainDB, and Blockchain DB. Blockchain Relational Database (BRD) is similar in design to Veritas, but is part of a PostgreSQL relational database. In this sense, BRD is an off-database blockchain. Also, unlike Veritas, the streaming service orders chunks of transactions (updates) rather than serializing transactions in a chunk. To speed up transactional execution, BRD implements concurrent execution with Serializable Snapshot Isolation (SSI). Note that BRD uses PostgreSQL as local database which supports serializable snapshot isolation. BRD also uses Apache Kafka as a streaming service. Unlike Veritas, BRD maintains the general ledger in the same relational database, namely PostgreSQL. According to the BRD document, the system achieves a performance of 2500 TPS with a key value utilization. BigchainDB is another non-database blockchain. It starts with MongoDB, a NoSQL database used as a local database. When using MongoDB, the main data abstraction in BigchainDB is an asset represented in JSON format. Otherwise, the transaction lifecycle is similar to Veritas. A transaction is verified locally by a node, then a request is sent to the streaming service.
- **BigchainDB** is an asset represented in JSON format. Otherwise, the transaction lifecycle is similar to Veritas. A transaction is verified locally by a node, then a request is sent to the streaming service.

  BigchainDB uses a BFT consensus middleware as streaming service, namely Tendermint. Once the majority of nodes approve the transaction, it is committed to 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 peak performance of around 200 TPS with YCSB workloads.
- **ChainifyDB** is a chain of blocks outside the database, starting from a relational database, which can be PostgreSQL or MySQL. Apache Kafka is used to stream transactions, which are SQL statements. The Ledger uses a custom style based on LedgerBlocks.

  A LedgerBlock contains all transactions that are part of a block, where a transaction is in its SQL format. The LedgerBlock then contains a list of bits representing successful transactions, a SHA256 hash digest of 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 that used by Fabric. ChainifyDB achieves throughput of around 1000 TPS on three nodes using the SmallBank workload when all three nodes need to reach consensus. If only two out of three nodes need to reach consensus, the throughput increases to around 5000 TPS.

VII. LEDGER DATABASE

Amazon Quantum Ledger Database QLDB 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.

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]

Table 2: Summary of Systems, Features, and Performance

LedgerDB could be a verifiable info developed by Alibaba associate degreed provided as a cloud service. LedgerDB updates the ledger, that is predicated on a Merkle tree, asynchronously.

Specifically, the transactions are batched and also the Merkle tree is updated with the batched transactions. Hence, this approach is named batch accumulated Merkle tree (bAMT). LedgerDB supports multiple underlying storage engines, however L-Stream, a custom storage developed by Alibaba, is that 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 work balancing. Our preliminary analysis of LedgerDB shows a turnout of 20,000 TPS, twice higher compared to QLDB. SQL Ledger could be a ledger info developed by Microsoft and offered as a service on its Azure cloud. it's an identical design to QLDB and Ledger DB, however it uses Microsoft's SQL Server because the underlying storage engine. SQL Ledger keeps a ledger organization supported Merkle trees and 2 tables, namely, the Ledger Table and also the History Table. The Ledger Table reflects the most recent record for a given key, whereas the History Table records the previous version of that record. It is not clear what variety of agreement is employed to coordinate among multiple nodes in SQL Ledger. Moreover, the rumored analysis wasdone on one server with seventy two cores. during this evaluation, SQL Ledger achieves a turnout of 70,000 TPS with TPC-C workloads. it's expected to check lower SQL Ledger performance in an exceedingly distributed setting

VIII. CHALLENGS

If we look at systems that combine blockchain and database functions, we see a lack of open source code for most hybrid blockchain and ledger databases. Therefore, it is difficult to understand the exact implementation and evaluate the performance of these systems. In our previous work, we reimplemented Veritas and BlockchainDB in a modular way that allows us to replace some of the components like the consensus mechanism and the local database. However, needs to do more to achieve a flexible and modular open-source hybrid blockchain database system where consensus and the underlying database can be replaced in a plug-and-play manner. At the same time, such systems must provide users with relational and key-value interfaces.

We have found that most existing systems offer simple key-value interfaces, with the exception of FalconDB, ChainifyDB, QLDB and SQL Ledger. What remains to be seen is the performance impact of a flexible user interface. For example, what are the implications of a relational interface when the underlying database is NoSQL? For such designs, the server node must be flexible enough and at the same time have good performance

IX. CONCLUSION

In this white paper, we discuss systems that combine blockchain and database techniques. We classify these systems into three categories, namely (1) permissioned blockchains, (2) hybrid blockchain database systems, and (3) ledger databases. Although they share a similar architecture, each category and system within a category has its own unique characteristics. Then we look at some representative systems, such as Fabric, Quorum, Veritas, QLDB, and LedgerDB, among others. The exact performance of these systems is difficult to assess 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 UI, consensus, and local storage are plug-and-play, we were able to answer more existing questions. For example, can we replace a CFT broadcast framework with a newer BFT consensus framework without a performance hit? These questions need to be answered in the future.

REFERENCES

[1.] Z. Amsden, et al., The Diem Blockchain, https://archive.ph/1xfcy, 2021.
[2.] 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.
[3.] 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.
[4.] E. Buchman, Tendermint: Byzantine Fault Tolerance in the Age of Blockchains, PhD thesis, The University of Guelph, 2016.
[5.] V. Buterin, A Next-Generation Smart Contract and Decentralized Application Platform, http:// archive.fo/Sb4qa, 2013.
[6.] Z. Gao, Y. Hu, Q. Wu, Jellyfish Merkle Tree, https://archive.ph/s7pPF, 2019.
[7.] M. Hearn, R. G. Brown, Corda: A Distributed Ledger, https://bit.ly/3iLajrI, 2019
[8.] 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.
[9.] V. Buterin, A Next-Generation Smart Contract and Decentralized Application Platform, http:// archive.fo/Sb4qa, 2013.
[10.] https://kafka.apache.org/, 2017.
[11.] BigchainDB 2.0 The Blockchain Database, Technical report, 2018. https://github.com/ConsenSys/quorum, 2021.
[12.] immudb, https://codenotary.io/technologies/immudb/, 2021.
[13.] MongoDB, https://www.mongodb.com/, 2021.
[14.] PostgreSQL, https://www.postgresql.org/, 2021.
[15.] Amazon Quantum Ledger Database https://aws.amazon.com/qldb/, 2022. https://archive.ph/edzMi, 2022.