future internet

MDPI

Article

A Survey of Ledger Technology-Based Databases

Dénes László Fekete [1] **and Attila Kiss** [1,2,*]

[1] Department of Information Systems, ELTE Eötvös Loránd University, 1117 Budapest, Hungary; denes710@inf.elte.hu
[2] Department of Informatics, J. Selye University, 945 01 Komárno, Slovakia
* Correspondence: kiss@inf.elte.hu

Abstract: The spread of crypto-currencies globally has led to blockchain technology receiving greater attention in recent times. This paper focuses more broadly on the uses of ledger databases as a traditional database manager. Ledger databases will be examined within the parameters of two categories. The first of these are Centralized Ledger Databases (CLD)-based Centralised Ledger Technology (CLT), of which LedgerDB will be discussed. The second of these are Permissioned Blockchain Technology-based Decentralised Ledger Technology (DLT) where Hyperledger Fabric, FalconDB, BlockchainDB, ChainifyDB, BigchainDB, and Blockchain Relational Database will be examined. The strengths and weaknesses of the reviewed technologies will be discussed, alongside a comparison of the mentioned technologies.

Keywords: ledger technology; permissioned blockchain; centralised ledger database

Citation: Fekete, D.L.; Kiss, A. A Survey of Ledger Technology-Based Databases. *Future Internet* **2021**, *13*, 197. https://doi.org/10.3390/fi13080197

Academic Editors: Spyros Makridakis, Peter Kieseberg and Thomas Moser

Received: 3 July 2021
Accepted: 30 July 2021
Published: 31 July 2021

1. Introduction

In recent years, we have been witness to the rise of multiple new ledger technologies. Since Satosi Nakamoto wrote the Bitcoin white paper [1] in 2009, many different systems that endeavour to provide decentralization and trustless collaboration have been developed based on blockchain technology. Of late, a great amount of research and academia has been focused on the further development and reformation of the basis of blockchain in an attempt to solve the issues that continue to arise [2–4]. However, blockchain is still very much in its infancy; new technologies appear seemingly daily, and yet, countless unanswered questions and unsolved performance issues still exist. Thus, the attention that ledger databases has received in both global marketplaces and in academia provides excellent opportunities for future development. Ledger technology, which utilises blockchain data structure and decentralised structure, is one of the best possible options in regards to security against malicious behaviour and privacy management; thus, it is already being used in, for example, global healthcare sectors [5–9], the finance field [10–14], and IOT networks [15–20]. Thus, the need for new improvements is constantly growing, which are as diverse as the fields in which ledger technology is used.

Blockchain is an append-only ledger technology that endeavours to be decentralised. From their genesis, blockchain-based systems were easily accessible and permissionless and have remained such. Thus, connecting to the network and accessing public data was and is attainable for all. However, the emergence of private enterprises utilising blockchain technologies gave rise to new needs and requirements. The creation of a private network that maintained the ideological basis of blockchain technology, one in which only the creators and maintainers of a network have access to it (referred to as permissioned blockchain), fulfilled one of the greatest requirements of private entities.

Blockchain technologies are decentralised, as they are based on Decentralised Ledger Technologies (DLT), which have lower performance and transaction throughput compared to traditionally distributed databases or the later discussed centralised approach to ledger technology. By default, blockchain-based networks are safer in regards to malicious

behaviour targeting the system than traditionally distributed databases. DLT's weaknesses, when compared to traditionally distributed databases, led to the need for a centralised approach that is able to compete with the performance of traditionally distributed databases. As a result, private enterprises were driven to venture towards a centralised approach. This approach is based on Centralised Ledger Technologies (CLT), which uses blockchain data structure to store data as mentioned in blockchain technology. This is referred to as a Centralised Ledger Database (CLD) [21]. The primary differences between blockchain and CLD are the maintenance of the system by a trusted third party and the lack of a need for a consensus mechanism. When examining secure and tamper-proof databases, it is critical to discuss the technologies within CLD and permissioned blockchain-based databases. This paper classifies CLD and permissioned blockchain-based databases as ledger databases. From a corporate perspective, this has the opportunity to be an ideal technology, as it provides trustless collaboration between enterprises.

First, there will be a short summary of the basis of ledger technology and blockchain and an introduction to the corresponding definitions and concepts, then a discussion of the relationship between blockchain and distributed databases. Following this, a variety of ledger databases will be briefly examined with the greatest focus being on transaction processing as this provides the backbone of a system. Furthermore, both types of ledger databases will be examined, with consideration regarding the type of database used, transaction processing flow, and the use of the smart contract. These are the only shared characteristics that can be compared between centralised and decentralised ledger databases. The replication model and consensus mechanism will only be discussed regarding decentralised ledger databases, centralised ledger databases that either do not allow access to such information or source code, or operations that occur on a higher level as a result of using cloud service, and all corresponding nodes being located in the same cloud hosted by a single service provider. In contrast, the majority of decentralised databases are open-source projects. The replication model is important regarding the examination of decentralised ledger databases, as it can define the mode of use of the ledger. Later discussed, permissioned blockchain-based databases in which the replication model is storage-based, where transaction processing is dependant on the existing blockchain on which the given databases are based. The architecture of these databases is comprised of different functionality layers. Finally, the strengths and weaknesses of the discussed technologies will be compared and contrasted.

2. Basic Concepts

Blockchain has become something of a catchall in recent times, leading to many misconceptions and misunderstandings. Its unanimous association with crypto-currency disregards the fact that blockchain is most widely used in financial sectors. The most appropriate definition of blockchain is that of a Decentralised Ledger Technology (DLT), which uses peer-to-peer connections between nodes. As has been previously mentioned, the primary aim, and one of the most important characteristics, of this technology is the creation of a decentralised system, which can support the creation of a trustless system. In this technology, sensitive data are organised into a block, which then form chains. The basis of blockchain and its transaction processing is the consensus mechanism, which provides a transaction order between nodes. The new block, which is to be added to the chain, is determined by the consensus mechanism. In addition to the structure of the technology, there is an opportunity to examine and deconstruct blockchain networks into two types in regards to the accessibility of nodes. Blockchain can be permissioned or permissionless [22,23], determined by the regulations regarding the joining of nodes into the network. Permissionless blockchains can be joined by any nodes and are public, whereas participation in permissioned blockchains are defined by given maintainers, usually private enterprises, and the blockchains are, therefore, private. As a result of networks being permissioned, Sybil attacks are not an issue. It is true for both these types that they provide a secure and tamper-resistant network in which malicious behaviours

are quickly detected. Data are stored in the ledger, allowing for easy proof and validation. The very first block is referred to as the 'genesis block'.

These key characteristics are also applicable to CLD, wherein the system is maintained by a trusted third party. Thus, with this approach the degree of centralisation is increased. Due to this increase, a consensus mechanism is not necessary or typical, in contrast with the decentralised structure. Often, these trusted third parties are enormous tech companies that provide Blockchain as a Service (BaaS) [24,25]. This service allows smaller enterprises to integrate blockchain functions effortlessly into their systems. When the two types of blockchain are examined from a security perspective, permissionless blockchain proves to be more secure as a result of its architecture, which requires no identity assumption, as opposed to permissioned blockchain, where identity assumption is necessitated due to its public nature. Nonetheless, CLD proves to be more secure than even permissioned blockchain as a result of the trusted third party.

The higher security of permissioned blockchain is noticeable even when comparing the differing types of consensus mechanisms that can be used in the two blockchains. Permissionless blockchains generally use probabilistic consensus mechanisms to reach agreement in the nodes regarding new blocks. Thus, they guarantee consistency in the ledger. However, in the use of probabilistic consensus mechanisms, there is a possibility that certain nodes have different states of the ledger—referred to as a ledger fork. In these cases, the transaction order is not globally agreed and defined. This possibility, a ledger fork, in a permissioned blockchain would cause inappropriate and uncertain results from an enterprise's perspective, as single enterprises would be using different states of the ledger in their blockchain network. As a result, permissioned blockchains generally use deterministic consensus mechanisms, which always have a globally agreed ledger state. Within blockchain-based databases globally agreed ledger states are an important criteria; thus, permissioned blockchains are most often considered when discussing blockchain-based databases generally.

Smart contracts are a useful feature of blockchain technology; however, they are not necessary, nor are they implemented in every blockchain. Smart contracts are an executable code on blockchain that result in changes in the state of the blockchain. They work as a deterministic program. There are options when it comes to the programming language; for example, Solidity or Java can be used. Smart contracts can be controlled using predefined rules regarding the allowed operations and the state of the database.

Distributed Database vs. Blockcahin

Many research papers [26–29] examine the similarities and differences between blockchain and distributed databases. These will be unpacked in this section, considering the ways in which these two technologies with shared similarities in their fundamentals also differ in many aspects. As previously mentioned, blockchain is most often used when there is no trust between nodes, whilst distributed databases are implemented in performance, rather than security, oriented cases. Distributed databases offer better performance as they have higher transaction throughput and lower latency. Blockchain's higher latency is a result of its use of consensus mechanisms, which provide strong consistency, as well as the decentralised structure that is fundamental to blockchain. This structure provides new opportunities for applications despite the decreased performance. An important difference between the two technologies is the deletion and modification of the data; in blockchain, this is impossible as the ledger is immutable. However, in distributed databases, this feature can be enabled, although it is not often put into practice. CAP theorem's [30] basic principles, as introduced 20 years ago, can be used to group distributed databases. CAP comprises of consistency, availability, and partition tolerance; however, it states that all three features cannot be achieved simultaneously. In actual use, partition tolerance is crucial as the system needs to work with arbitrary network failures. Thus, the compromise is between consistency and availability. In the same way that CAP applies to distributed databases, so was DCS theorem [31] proposed to define blockchain technology's property-

set. DCS consists of decentralisation, consistency, and scalability. Similarly to CAP, only two system properties are available at once of the three. However, in permissioned blockchain, total decentralisation cannot be achieved due to its enterprise-associated characteristics; thus, in these cases, consistency and scalability must be best exploited.

3. Centralised Ledger Databases

3.1. LedgerDB

According to the creators of LedgerDB [21], many applications that are built on permissioned blockchains stumble into performance issues, resulting in low throughput, high latency, and significant storage overhead. Thus, if a decentralised structure is not necessary, it is possible to shift towards a centralised system, as current trends in the industry lean towards a single service provider supplying the total nodes of an application. As a result, the CLD solves the mentioned issues through its inherent architectural characteristics. LedgerDB, in an effort to meet customer needs, concentrates on the achievement of high write performance, whilst maintaining acceptable read and write verification performance. A new possibility presents itself for LedgerDB on Alibaba Cloud, which is a centralised database with tamper-evidence and non-repudiation features similar to blockchain and provides strong auditability. The majority of ledger databases tend to assume Ledger Service Provider (LSP) trust; however, LedgerDB takes up multi-granularity verification and non-repudiation protocols, thus foregoing the trust assumption. Therefore, it opposes user and LSP malicious behaviour.

There are three key node types in LedgerDB: ledger master, ledger proxy, and ledger server. As a ledger master manages the corresponding cluster-level events and all the corresponding cluster data, it is at the heart of LedgerDB. The ledger servers and ledger proxies issue resolution is seamless as a result of their statelessness. Clients' requests are preprocessed and forwarded to the corresponding ledger server by a ledger proxy. The preprocessed client requests are completely processed by a ledger server, which then communicates with the underlying storage layer. The storage layer, which stores data, can take a variety of forms, for example, key-value stores, file systems, and so on. The storage layer, following Raft protocol, is implemented in the shared abstraction, which makes the replication of data possible within various zones and regions. There is a possibility to delete from the database through API without breaking the verification or audit process. Therefore, the requirements for storage can be reduced.

LedgerDB requires a third-party timestamp authority (TSA) to be trusted. This ensures the maintenance of the above-mentioned threat model. The TSA attaches a true and valid timestamp to a given piece of sent data. A new, special journal type which contains a ledger snapshot and a timestamp, called a TSA journal, is approved by the TSA. The use of a TSA is of assistance in proving the existence of a piece of data before a given point in time. This is a two-way pack protocol, wherein a ledger digest is first signed by the TSA, then it will appear as a TSA journal on the ledger. As a result, the vulnerability time window is reduced.

3.1.1. Verification

LedgerDB, through its verify app, allows for the possibility of verification and authentication of returned journals from journal proofs. In LedgerDB, proofs are tamper resistant, as they are signed by a component. There are two options for verification, server side or client side. Verification can be sped up through the use of trusted anchors. A trusted anchor is a ledger position where the verification of the digest has been successful.

3.1.2. Transaction Processing

The state of the ledger can be modified through transactions, which result in journal entries that will be added to the ledger. A block is made up of multiple journal entries. Instead of the usual order-then-execute approach, LedgerDB uses a new execute–commit-index transaction model, as is depicted in Figure 1. This model makes the most of the

centralised structure and connects the execution and validation phases. There is no consensus mechanism implemented, as a single service provider manages the network.

Figure 1. Execute–commit-index transaction processing model.

The use of permissionless or permissioned blockchain verification models for data management, such as smart contract or UTXO, continue to give rise to issues regarding performance. For example, a recursive search is necessary for the UTXO verification model to locate the appropriate data. As a result, LedgerDB uses clues, which are user-specified labels (keys) that realise business logic. Clues can be ordered for given assets with a unique ID. Therefore, as opposed to key-value models, a clue indicates every state of an asset is connected through only one clue index.

Execute Phase: the transaction sent from the client is verified and checked for necessary permissions for the operations. If checks are successful, the transaction is stored in the storage layer, decreasing communication and storage costs. After execution is completed and transactions have been executed, successfully or unsuccessfully, they are sent to an appropriate ledger server.

Commit Phase: following the sending of transactions by a ledger proxy to an appropriate ledger server, the journals of the previous phase are collected and are then processed in a group. These collected executed transactions are ordered in this phase in a global order. The transactions are forwarded to the storage system, where they are stored. The success of a transaction is determined in the commit phase, and only successful transactions are stored. A client is issued a receipt regarding their transactions, regardless of the transaction's success. Thus, clients receive updates in the form of these receipts regarding the status of their transactions.

Index Phase: the index phase is important due to its retrieve and verification mechanisms. The necessary indexes for data stored in the commit phase are created. There are three types of indexes in LedgerDB: clue index, bAMT accumulator, and block information. The block information is not a digest of a single block, rather it is a summary of the entire ledger. Thus, all ledger information is easily globally verifiable as they were all created at a given time using this summary. After indexes are built at the end of the index phase, the receipt is sent back to the client.

4. Permissioned Blockchain Based Databases

4.1. Blockchain Relational Database

The main idea behind Blockchain Relational Database (BRD) [32] is similar to Hyperledger Fabric [33], but this system is implemented at the low level, as BRD is a part of PostgreSQL. As a result, a huge dependency on PostgreSQL developed. Thus, BRD works as a relational database. Four thousand lines of C code are supplemented in PostgreSQL. Despite this, it has one of the most developed approaches to transaction management in the blockchain-based database world. Researchers' experiments show it has one of the highest transaction throughput. Note that smart contracts in this system are stored procedures in PostgreSQL. Database nodes execute transactions independently of one another and connect with decentralised consensus to commit the order for transactions, as the consensus mechanism determines the order. There are two techniques to process transactions: the more straightforward consists of lower performance as ordering takes place first, while in the other, executing happens first and ordering occurs parallel. This system has three particular types of nodes:

- client: An administrator is responsible for clients' access to the network. Clients' digital certificates are stored in database peers. Clients can receive the transaction status through a notification channel.
- database peer node: The blockchain's crux is the nodes' storage of all important data. Every database note maintains their ledger and executes smart contracts. Validation and the addition of new blocks happen through these nodes.
- ordering service: The ordering service is made up of the consensus mechanism or orderer nodes. If the ordering service is ordered nodes, differing ordered nodes must be owned by different organisations. The ordering service is pluggable and agnostic.

4.1.1. Serializable Snapshot Isolation (SSI)

Serializable Snapshot Isolation (SSI) achieves serializability using a modified Snapshot Isolation (SI) [34] technique. A Multi-Version Serialization [35] History Graph is used to detect SI anomalies. Every transaction is represented as a node in the graph. An edge signifies a transaction to another transaction, if the first one is preceded in the order of execution. Three types of dependencies can create these edges:

- rw-dependency
- ww-dependency
- wr-dependency

A cycle in the graph presents an anomaly; thus, the execution does not correspond to any serial order. Anomalies are automatically detected and resolved by tracking rw and wr dependencies, which abort during commit. These are helped by the maintenance of two lists per transaction.

SSI Based on Block Height: In order to permit transaction execution to happen in parallel with ordering based on consensus, a new variant of SSI based on block height is devised. By default, SSI cannot guarantee the same committed state of executed transactions on all database nodes. It is used only to execute and order in parallel. To guarantee transaction execution of the same committed data on all database nodes, each row of a table contains a creator and deleter block number. These represent row creation and deletion by the appropriate block number. All submitted transactions include the block height and number on which it should be executed.

As previously mentioned, it is used by only one of the approaches, but implementation of the order-then-execute approach incorporates this modification of SSI for provenance queries. To guarantee the appropriate serializability, analogous transactions that result in phantom reads and stale data reads will be aborted.

SSI variant—block-aware abort during commit: To ensure consistency between nodes, we need to ensure that the same set of transactions are aborted on all nodes. This technique uses an abort rules table with flags from supposed SSI for which transaction must be aborted. A phantom read or stale read problem can occur as a result of concurrent transactions that commit during or after current-block. This devised SSI tracks these problematic transactions in the commit phase.

4.1.2. Transaction Processing

Two approaches are identified to achieve a blockchain-based relational database between untrusted nodes. The first step of the first technique, order-then-execute, is the ordering of all transactions by a consensus service, which are then executed by database peer nodes simultaneously. The second technique has a different approach; first, database peer nodes, without knowledge of ordering, execute transactions. During execution, ordering is determined by an ordering service. The second technique has better performance compared to the first; however, the tradeoff is a greater requirement of implementations and modifications on the relational database.

Order then Execute: submitted client transactions in the order-then-execute are required to be firstly ordered by an ordering service, followed by the execution and commit-

ment on the database peer node and, finally, being recorded in the checkpoint phase. This model can be seen in Figure 2.

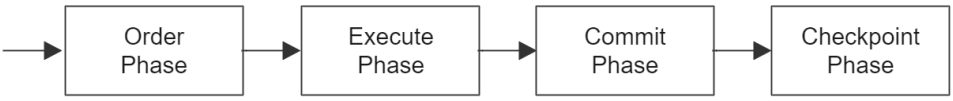

Figure 2. Order-then-execute transaction processing model.

Order Phase: Clients send transactions directly to any one of a number of ordering service nodes. On an intermittent timeout (for example, every one second), ordering service nodes attempt to agree on a consensus to create a new block of sent transactions from clients. Following the construction of a block, all nodes receive the block from the ordering service by broadcast.

Execute Phase: Following the order phase, every database node must verify that the current block is in sequence and was sent by the ordering service. Every transaction from the received block is appointed a thread. The PL/SQL procedures are executed separately on each thread with the received arguments corresponding with the transaction. Using the 'abort during commit', the SSI variant ensures the same committed state on all database nodes. The next block will only be processed if the current block of transactions is committed.

Commit Phase: The commit phase begins after the execution of all transactions. These transactions can be committed or aborted. The order in which transactions appear in the block is the same as the order in which they are committed. This is to ensure that the commit order aligns on all database nodes. As in the execute phase, 'abort during commit' SSI variant is used to ensure execution of transactions on the same committed state.

Checkpoint Phase: Following the processing of all transactions in a block, to ensure that the execution and commitment on all non-faulty nodes was successful, these nodes calculate the hash of the write set. This hash represents all changes made to the database by the block. This is submitted as proof for the ordering service that all prior phases were successful. This submitted proof will be the same if there were no faulty nodes that sent proof. It is not necessary for all blocks to include a record of a checkpoint.

Execute and Order in Parallel: Having better performance than the aforementioned, as the ordering by the ordering nodes and execution by the database nodes happens simultaneously, the execute and order phases occur parallel. Following this are the commit and checkpoint phases. This model can be seen in Figure 3.

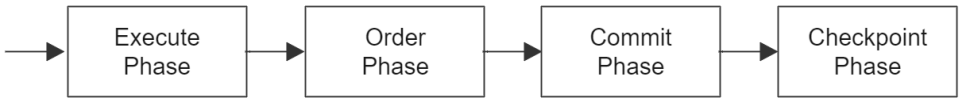

Figure 3. Execute and order in parallel transaction processing model.

Execute Phase: Clients send transactions in the same way as has been detailed prior. However, here, the database node that receives this request, sends it to the other nodes and as well as to the ordering service to be ordered in the background. Every transaction received by a database node is designated a thread. This thread is used to forward and execute the transaction. Thus, execution and ordering happen in parallel. As a result of this parallel action, a shared transaction includes the block height to guarantee the same committed data on all database nodes.

Order Phase: The ordering service receives transactions from the database nodes, differing from the order-then-execute approach. Therefore, the ordering can occur simultaneously as a result of the ordering service, while the execution of transactions in the database nodes occurs. The remainder of the processes correspond with the order-then-execute approach.

Commit Phase: The transactions pertaining to a block are completely executed and can be committed or aborted as in the previous order-then-execute approach. However, order-then-execute approach differs in two primary aspects due to its parallel architecture:

1. If a received block has transactions that have not been executed, these transactions begin to be executed. After all the missing transactions have been executed, they continue to the commit phase, as in order-then-execute techniques.
2. Blind updates are not supported due to the feasibility that transactions may be executed at differing snapshot heights. On one node, a transaction has the possibility to not be concurrent, whilst on another, it may be concurrent. Thus, the block-aware abort during the commit variant of SSI is used as opposed to the abort-during-commit variant of SSI.

Checkpoint Phase: The order-then-execute approach's checkpoint phase is identical to this checkpoint phase.

4.2. BigchainDB

Released as an open source project in 2016, BigchainDB [36] has been steadily improving since. However, lately its active programming community has diminished. It is one of the most well-known blockchain-based databases in the research field. It attempts to examine the questions and issues of blockchain-based databases' transaction models from a compelling position, as it utilises the concept of owner-controlled assets. Therefore, BigchainDB presents the most appropriate solutions to questions of supply chains, intellectual property rights management, and data governance, for example. In these instances, where comparable assets are considered, e.g., products or people, the opportunity still exists to use it as a key value database. In opposition, certain research [37] suggests the implementation of smart contract features ensures more and better opportunities for supply chains. Another study [38] has brought attention to the minimal hardware resources and low energy consumption observable in supply chain management due to the high use of IOT devices, which in the case of BigchainDB, does not occur. The key concept behind BigchainDB was to approach the characteristics of databases such as low latency, high transaction rate and high capacity. BigchainDB uses the MongoDB database. Every node has its own local replica of the database. Tendermint [39] is used as BigchainDB's consensus mechanism. Due to Tendermint being Byzantine Fault Tolerance (BFT) [40], the resulting system is also BFT.

4.2.1. Owner-Controlled Assets

The concept of owner-controlled assets, as with most blockchains, is present in BigchainDB. Thus, a transfer of an asset is possible only by the owner/s. Transfers are not possible even by node operators. Usually, there is only one built in asset in a blockchain technology, such as in Bitcoin or Monero [41]; however, in BigchainDB, as many assets as are required can be created. An asset can store arbitrary data in JSON with only one restriction as opposed to a limit. This restriction is the transaction size, which is adjustable. The owner-controlled asset is divisible but cannot be less than a single unit. Data and metadata are where arbitrary JSON documents can be stored.

As previously mentioned, stored asset data cannot be changed or deleted; the only possibility to add new information to transactions is to use the metadata field. This is a useful tool in the TRANSFER transaction to add new information to transactions regarding what has been changed, for example, the kilometres driven on a car, the age of the vehicle, or any type of data. There are five types of transaction operations:

- CREATE
- TRANSFER
- VALIDATOR_ELECTION
- CHAIN_MIGRATION_ELECTION
- VOTE

The final three operations are connected to the consensus mechanism. The CREATE and TRANSFER operations are more important from the data creation and modification point of view.

Create: To register any type of asset, a CREATE transaction can be used with arbitrary data and metadata. A CREATE transaction can be used to register any kind of asset (divisible or indivisible), along with arbitrary metadata. An asset can have any number of owners, including the possibility of none. Later, only owners can use their assets. The number of owners agreed on the validity of a transaction regarding a certain asset is configurable.

Transfer: A user must fulfil the condition in order to transfer an asset. Thus, permission to transfer is only afforded to certain users. In this way, the owner/s of an asset can be changed or metadata can be modified.

4.2.2. Transaction Processing

BigchainDB's transaction processing is the simple order-then-commit, as can be seen in Figure 4. It is similar to order-then-execute transaction processing. Firstly, received transactions are ordered by Tendermint, then, based on the answer from Tendermint, the block is committed to the local MongoDB by each server, and each server sends a response back to their own Tendermint component.

Figure 4. Order-then-commit transaction processing model.

Order phase: Clients can communicate with BigchainDB via APIs. Gunicorn web server, inside the server node, accepts HTTP requests. Then, as mentioned, the Tendermint instance receives the message. Messages are then sent through the Tendermint Broadcast API. If all validation has occurred, consensus is reached by Tendermint. All transactions in the memory pool will be made part of the block and a block will be successfully created, following which all servers vote on the validity of the block.

Commit phase: Consensus having been reached between all Tendermint component of servers, BigchainDB commits the block to the local MongoDB database. BigchainDB does not store transactions which failed validation and does not provide information about them in APIs. In Tendermint both failed and successful transactions are included in the current blocks.

Checkpoint phase: Each server indicates to their own Tendermint component via a response message the successful commit of the block to their local MongoDB database.

4.3. FalconDB

FalconDB [42] nodes can be server or client, which assist clients who have limited hardware requirements. Thus, low storage, computation, and bandwidth do not influence the operations of the decentralised network. To achieve this goal, only the server nodes store the whole database, while clients only store the blocks' headers. As server nodes store the whole database, queries and update requests are received and processed by only the server nodes. Authentication Data Structures (ADS) are stored in the database on the server nodes. This ensures the possibility for clients to verify requests sent to the server nodes. A small piece of digest is generated for the current database content by the ADS, which can be used by clients to grant authentication of the returned server results. Every block header contains the digest generated for the given state. As clients store blockchain headers, they can access the digest of the current database content. The design of the authenticated database with this digest enables FalconDB to be made up of 1/3 malicious nodes and can continue to function with all server nodes being malicious excluding one.

FalconDB is independent from the blockchain platform. Thus, any blockchain platform can be built in FalconDB, which can be used in an incentive model. FalconDB has the ability to use a variety of ADS solutions, whichever satisfies the model. The FalconDB model is built in an incentive model to ensure the appropriate honest replies to any queries or update requests. It is not a priority for server nodes to produce or collect profits, but an environment can be created in which costs can be shared according to use. Furthermore, a consequence is assured in the case of a dishonest node; the consequence being a monetary loss.

4.3.1. Authenticated Data Structure

Recognising malicious server nodes in outsourced databases is a function of Authenticated Data Structures (ADS) [43], where databases are uploaded to a cloud server and used remotely. ADS provides clients the opportunity to verify server results; thus, it can validate query and update requests. ADS has five key functions for querying, validating, and calculating the digest. The digest is generated for the current database content.

The correct execution of the request can be validated through running verification and proof validation over the results and proofs received from the server node. The validation is successful if the server executed the appropriate functions of the ADS.

Limitation of ADS: ADS has two major limitations, which are solved by blockchain architecture:

1. outdated, malicious digest causes issues, as the user must have the most recent valid digest.
2. an attack can cause changes to the data, for example inserting or deleting records, this data cannot be recovered by ADS, nor can the attacker be identified.

The first limitation is solved with a BFT consensus mechanism, which synchronises the digest among the clients easily. The second limitation is solved through immutable and transparent property.

4.3.2. Transaction Processing

FalconDB is incentive model-based and creates the opportunity for low hardware requirement clients to use an honest service in which servers are incentivised to provide services and penalised for dishonest behaviour. At the core of this model are smart contracts (e.g., on Ethereum [44]). Clients and servers can communicate with one another and make deposits through these contracts. As a result of the incentivised model, either the client or the server must pay a fee, or if a server is malicious, all funds in the smart contract account will be lost. There are two key aspects to the smart contacts:

- Service fee contract: a server receives a fee from the client to process queries and requests.
- Authentication contract: for the duration of this contract the server's account is temporarily frozen until it shares proof regarding the requested smart contract.

The architecture of FalconDB can be split into two layers. These are the authentication layer and the consensus layer. To better understand the basics of FalconDB, let us take a look at the details of a sent query through the layers.

Consensus layer: Queries are sent from a client to the server nodes via the service fee smart contract. The server account receives a fee for the processing of the queries. Immediately upon executing the query, the result is returned to the client. After the agreement through the BFT consensus protocol, the new block will be committed in the blockchain, and the new generated digest is available for all clients to update their own digest.

Authentication layer: As has been mentioned, clients have the opportunity to use the authentication contract to be reassured of the success of the request. However, in this case, an extra fee must be paid to the server from the client. Thus, ADS proof is generated by the server to validate the digest by the client.

4.4. ChainifyDB

Chainfy's [45] main aim is the creation of heterogenous transaction processing systems. This means that organisations and companies do not have to execute transactions similarly; thus, transactions are not expected to be deterministic on every server node. Therefore, ChainifyDB can connect numerous database managers with various implementations. Hence, the above-mentioned divergence in transaction execution is a cause of the different interpretation of the SQL standard or the used data types. The usual transaction processing models are not able to meet this set target. As the execute phase usually follows the consensus phase, there must be an assumption of proper execution. A new model needs to be created, which ChainifyDB will use. Referred to as the Whatever-Ledger Consensus model, this will make it possible to chainify heterogeneous infrastructures, as opposed to the usual model, which is not compatible due to its assumptions. To approach different transaction processing through a heterogeneous blockchain network, in the whatever phase, each organisation processes a batch of inputs however they like. Thus, there are no assumptions in the whatever phase. Following this, the whatever phase passes to the ledger-consensus phase, where it attempts to achieve a consensus on the transactions. There is a necessary requirement in the use of database systems; a trigger mechanism must be supported, as defined in SQL 99. This trigger assists in creating digest tables for every block. The digest table assures tamperproofing and the achievement of consensus. To achieve consensus, ChainifyDB uses a lightweight voting algorithm. This is configurable as to how many organisations must be in agreement to achieve consensus. If the defined number of organisations have the same hash and the single organisation likewise, the single organisation adds a block to its own ledger.

A Chainify network is built on the servers provided and run by organisations—an untrusted OrderingServer and a set of Kafka nodes. The blocks created by the OrderingService are broadcast by the Kafka nodes. Within the ChainifyDB network, a single organisation can run on various servers. Of these, at least one server is necessary so that an organisation can participate completely in the network. The server can be a ChainifyServer, an AgreementServer, an ExecutionServer, or the ConsensusServer. The ChainifyServer communicates with the AgreementServer, which collects the agreements required for transactions. These agreements are defined by the policies, which describe the set of organisations needed to be in agreement. The ExecutionServer receives the created blocks from the OrderingServer. The transactions in the blocks are executed in the underlying database system. Once all the transactions have been executed, the ExecutionServer generates the corresponding hash with the aforementioned trigger. This will be used by the ConsensusServers in communication with one another.

Transaction Processing

ChainifyDB uses Whatever-Ledger Consensus transaction processing, as depicted in Figure 5. As has been mentioned, Chainfy is made up of various servers. The processing of the transaction through these servers is ChainifyServer, AgreementServer, ExecutionServer, and the ConsensusServer. Firstly, ChainifyServer receives the signed transaction from the client. It then communicates with the AgreementServer in the already detailed manner to collect the agreements required for the transaction. If this collection is successful, the OrderingServer receives the transactions and creates a block. This block is forwarded to a Kafka node. Following this, all the ExcecutionServers receive the block from this node. The transactions contained in the received block are executed by the ExecutionServers, and the corresponding block hash is generated for the consensus. The ConsensusServer receives the hash corresponding to the block and can decide if a consensus has been achieved or not. If it has been achieved, the block is added to the ledger; if not, recovery is performed.

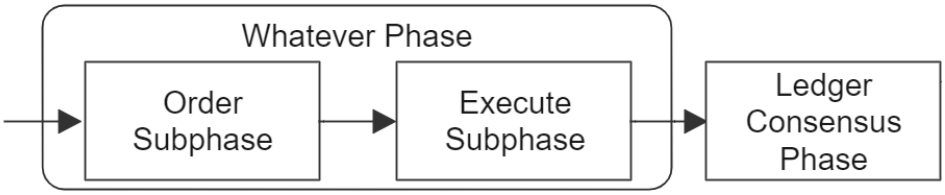

Figure 5. Whatever-Ledger Consensus transaction processing model.

By default, the batch of transactions is executed sequentially. However, this approach has low performance issues, is time consuming, and is inefficient. Better performance can be achieved through the parallel execution of transactions. This must be achieved through a well-defined parallel model, as it naturally has consequences regarding commit-order, increasing the likelihood of inconsistency. The actual model builds on many small batches of transactions that can be executed safely in parallel. To achieve this, in the execute subphase, conflicts between transactions are located.

Whatever Phase: Order Subphase: The order subphase consists of block creation; the global ordering and grouping of a block from a batch of input transactions. In opposition to the OCM model, a consensus round does not occur after the ordering is completed.

Execute Subphase: If a transaction in the block is valid, it is executed in the local database. Notice that opposed to the OCE model, this execution is not deterministic in all nodes.

Ledger Consensus Phase: In this phase, consensus can be established, as has been mentioned, on the basis of whatever phase produced effects.

4.5. BlockchainDB

In a unique way, BlockchainDB [46] uses classical data management techniques, which consist of data sharing. The primary notion behind BlockchainDB is the use of blockchains as a storage layer as well as implementing a database layer on top. Thus, the native storage layer can be an existing blockchain technology, which can be then used as decentralised storage. With the assistance of the database layer, shared tables can be accessed. As a result of this architectural choice, a possibility is ensured that nodes do not store all data in the blockchain network locally. Due to the sharding, an arbitrary number of peers store the shards. This intention is observable in the architectural features of and changes to Ethereum 2.0 [47] and Rapidchain [48]. The use of data sharing does not negatively impact security or the aforementioned basic principles of blockchain.

The participating nodes in a BlockchainDB network can be full peer or thin peer. Thin peer does not store any shards; thus, it can take part in a blockchain network with limited resources and use the shared table, while the full peers manage databases and store at least one shard replica. Clients must trust in their own BlockchainDB peers. They can request transactions through a limited put–get interface. This key–value interface can be used to read and write data without knowledge of the blockchain technology. Moreover, clients can verify the methods of database layers through the use of off-chain verification. As has been mentioned, the architecture of BlockchainDB can be split into two layers. The aforementioned key–value interface is used by the database layer, while the storage layer stores all data built on blockchains.

Database Layer: The database layer within BlockchainDB is found on top of the storage layer. It serves the clients through the aforementioned put–get interface. With the assistance of the Shard Manager, the necessary shard of a given table is found. Depending on the definition of the partitioning scheme and the replication scheme, the sought after shard can be in either local storage or remotely in another peer's storage. Alongside the Shard Manager, a Transaction Manager assists in the maintenance of consistency levels and defines the behaviour for concurrent transactions.

Storage Layer: The storage layer is built on an existing blockchain technology, the backend of BlockchainDB. Its task is the storage of all data, which interacts with the BlockchainDB. Furthermore, it is responsible for the creation of shards and providing access for shards through the mentioned well-defined interface. The creation of shards is based on the key value data model. The interfaces are independent from the blockchain technology, as BlockchainDB uses them through backend connectors. The backend connectors ensure a stable, predefined interface for the management of data via the smart contracts. These smart contracts need to be installed in BlockchainDB to use their basic read/write functions. There is a possibility to process data simultaneously, across different blockchains, wherein various blockchains are various shards. These blockchain technologies can be Ethereum or Hyperledger Fabric, for example.

Transaction Processing

As has been presented, BlockchainDB's transaction model and consensus mechanism relies in part on the chosen existing blockchain. Thus, the route of a transaction from a client to the database and the storage layer, up until the chosen blockchain receives the given request, will be examined here in abstract.

Database Layer: Clients communicate with their trusted BlockchainDB peer through put/get requests. These requests are received by the database layer and then are processed by the transaction manager. The transaction manager processes transactions with the customisable consistency level. Before a put/get request is sent to the storage layer, it is forwarded to the ShardManager. The task of the ShardManager is to ascertain the appropriate shard for a given key.

Storage Layer: The storage layer is already aware of the exact location of the necessary shard on the basis of the data received from the database layer. Thus, it can interact with the blockchain network and can perform the necessary function with the aforementioned interface, for example, read-write-async.

Offchain verification: Clients can verify the appropriate execution of their sent requests through the database layer. This offchain verification can be completed on- or offline. With online verification, clients can be assured of the validity of their received results through the verify operation. Offline verification does not call verify operation for single requests; it defers verifications until multiple operations can be executed at once.

4.6. Hyperleder Fabric

Hyperledger Fabric [33] is the first blockchain system and one of the Hyperledger projects supported by the Linux Foundation. Almost 400 organisations and companies use and test this recently popular open-source blockchain project. It is able to execute distributed applications within standard programming languages, for example, Java, Go, and Node.js. As a result of its support, it is able to achieve the effect of execution on a single globally distributed blockchain computer. Thus, Fabric is the first distributed operating system [49] for permissioned blockchains. However, this can be used easily in even simple applications [50].

Hyperledger Fabric is a jack of all trades. It attempts to solve an organisation's every problem. The basic unit is the channel, which is the separate ledger of transactions. The rights to this ledger are granted by a group of members. From a business perspective, a useful feature is the ability to create complex networks. Furthermore, Hyperledger Fabric has a number of pluggable options, such as the use of an arbitrary Membership Service Provider (MSP) and consensus mechanism.

Hyperledger Fabric contains smart contracts, which define the business logic. With the assistance of smart contracts, the state of the ledger can be changed, as the ledger stores the current and historical state. Typically, a chaincode is used by administrators to create groups relating to smart contracts. These are run in isolation in Docker containers. The GetState, PutState, and DelState are used to reach local states in the chaincode.

The MSP assists in the identification and authentication of members of a blockchain network. The communication between nodes takes place with the assistance of the gRPC framework. There are three types of nodes in a Fabric network:

- clients: Clients can send transactions to peers for execution, and having executed, send for ordering service.
- peers: All peers store the full replica of the ledger. Transactions are executed on the peers according to the given policy of the transaction.
- Ordering Service Nodes (OSN): There is one task for the nodes—the total order of all transactions. Thus, it has no role in the validation-and-execute phase. The application state is unknown to the ordering service.

In regards to data storage, along with the ledger, the world state assumes a key position, as in a given moment the world state is the state of the ledger. The world state is the database of the ledger. The world state database can be two similar key/value store databases. By default, it is a LevelDB; however, it supports fewer opportunities for queries as opposed to CouchDB, which supports rich queries. If CouchDB is the world state database, the data on the ledger is stored as JSON. The other important component of the ledger is the transaction log. The transaction log records before and after values of the ledger database being used by the blockchain network.

The consensus mechanism can be adjusted to best serve the requirements and needs of the blockchain network. Since the secon version, Raft is the advised crash fault tolerant ordering service. Set up is done more easily in Raft as opposed to the older Kafka-based ordering service. Raft is established on a "leader and follower" model, where leader nodes are chosen per channel, and its decision is reproduced in the other follower nodes. The transaction model of Hyperledger Fabric is not built on the usual order-validate-execute model, rather it uses execute-order-validate, which is very similar to the aforementioned model used in the BRD.

Transaction Processing

The transaction model of Fabric uses an innovative execute-order-validate model, as can be seen in Figure 6, wherein there are three separate stages that are capable of executing on separate nodes in the blockchain network. Before transactions are ordered, the client sends it to the endorsing peers to execute the transactions and endorse its validity. Then, transactions are ordered by the ordering service, following which they are validated according to the specific assumptions.

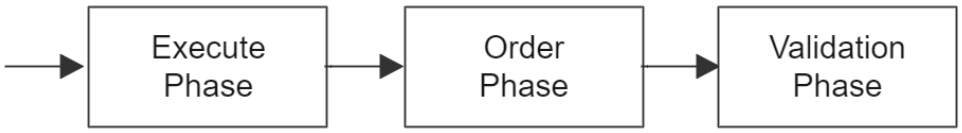

Figure 6. Execute-order-validate transaction processing model.

Execute Phase: In this model, clients send transactions to one or more endorsing peer/s, which execute the necessary chaincodes. During this process, no communication or synchronization occurs between the peers. After execution has been completed, otherwise known as the simulation, every node can prepare its own endorsement, which contains the read-write sets of the simulation. The client is required to collect as many of these endorsements as is dictated by the chaincode. Following this, the client is able to send the transaction to the ordering service.

Order Phase: As has been mentioned, once the client has collected the necessary number of endorsements, the ordering service receives the transaction and determines the order as consensus is achieved on transactions. In the interest of better performance, the ordering service can organise and group multiple transactions on a single block.

Validation Phase: The ordering service, having completed its task, forwards the block to the peers, which complete the following three sequential steps to validate the received block:

1. Endorsement policy evaluation: The peers evaluate all the transactions in the block simultaneously to determine if each satisfies the given endorsement policy and the peers mark the transactions as either valid or invalid. Only transactions marked as valid will have effects in the database.
2. Read-write conflict check: The keys in the current state of the ledger are compared with the keys in the readset to determine if they match. If they match, they continue to be marked as valid; if not, they become invalid.
3. Ledger Update Phase: Following the completion of the previous two validity checks, the state of the blockchain is updated with the results of the validations, and the block is added to the local ledger.

4.7. Corresponding Studies to Hyperledger Fabric

As Hyperledger Fabric is a popular topic nowadays, and hundreds of companies are trying to achieve better and better performance, much research has been devoted to the development of increased performance. This research usually either creates completely new systems whilst maintaining the basics or suggests improvements on certain aspects of Hyperledger's architecture. This research is also interesting in the way that they illuminate the various weaknesses of Hyperledger Fabric, for example, in the throughput and processing model of transactions.

4.7.1. Fabric++

Two simple experiments recognised a number of issues with Hyperledger Fabric's simultaneous transaction processing [51]. These problems limit parallel transaction processing, the main issue being the abortion of very high numbers of transactions caused by serialisation conflicts. If the number of successful transactions is to be increased, two necessary modifications must be done. The ordering and abortion of transactions must be further developed. The ordering of transactions is developed through the use of an advanced transaction reordering mechanism whilst implementing the concept of early abort in multiple phases is a solution to transaction abortions.

Transaction Reordering: Firstly, the conflicting transactions must be identified. It is challenging to determine these transactions as they are executed in isolation. Thus, it is not possible for the transactions to see one another's later problematic and conflicting modifications. Transactions must be ordered such that the reading of the given key occurs before the given key is written. Cycles of conflicts can occur. As a result, the suggested mechanism for transaction reordering deletes certain transactions to form a cycle-free subset. The transactions present in the highest number of cycles are aborted first until a cycle-free state is reached. The remaining transactions are integrated into a serialisable schedule.

Early Transaction Abort: To reduce the number of broadcast transactions in the network, the transactions that were removed in the mentioned reordering mechanism can now be aborted in the order phase rather than later in the validation phase. A form of early abort is also accessible to reduce performance issues. Eventually, this concept of early abort places only those transactions in the block that have a likely possibility of commit. To further reduce the number of problematic and invalid transactions in the pipeline, abortions are also attempted in the execute phase.

To allow for simultaneous validation in the execution and validation phase within a peer, Fabric's concurrency control mechanism must be made more efficient. With this further developed concurrence mechanism in the execute phase, it is possible to recognise stale reads and abort the corresponding transaction during execution. In the order phase, early abort is similar to the previously described execute phase. The primary aim is to have two transactions read the same version of the key within the block.

4.7.2. FastFabric

Architectural changes were made to the 1.2 version of Hyperledger Fabric [52], all of which have a great impact on transaction processing, thus achieving major improvements in performance. The research claims to have increased transactions per second from 3000 to 20,000. Two primary changes have been made to the ordering service, and a number of improvements have been implemented in the peers.

Ordering Improvements: The first change in the ordering service is that it is able to decide on the transactions' order based on the transaction's ID. Thus, the communication overhead is decreased, with which throughput capability is increased. The second change implements the parallel validation of transactions. Therefore, instead of by default sequentially validating transactions, transactions are able to be validated simultaneously.

Peer Improvements: In the case of peer tasks, there is much room for improvement. One of these improvements has to do with the use of a hash table as opposed to the world state database. Thus, servers are able to reach necessary keys faster as they are stored in memory. This modification has another consequence: the blockchain log and the state base are stored using a peer cluster in various hardware. The next modification frees up resources on the peer through separating commitment and endorsement onto different hardware, as up until now, the same peer has been completing these tasks.

The research parallelizes all things that have a positive effect on performance; thus, the validation of the block and transaction headers will be parallelised. The last improvement considers transaction validation by temporarily caching unmarshalled data until the end of validation.

4.7.3. Nexledger Accelerator

Nexledger Accelerator [53] attempts to improve throughput with the assistance of an accelerator, which is an intermediate standalone server between applications and blockchain network/Hyperledger Fabric. Thus, no modifications occur in Hyperledger Fabric. The primary concept of this approach is the collection of those transactions that do not read or write the same key into smaller batches. These collected transactions, which are uncorrelated, are compressed into a new transaction, and this is forwarded to the blockchain network. As a result, multiple transactions can be processed in a single consensus phase.

The Accelerator has three essential components that happen sequentially. First, the type of received transaction is determined through a combination of its channel name, chaincode name, and function name in the chaincode, resulting in classification. Following, the classified transactions are grouped on the basis of their type into a new transaction. The new transaction is then routed to the blockchain network for consensus.

5. Summary

The discussed ledger databases are summarised in Tables 1–3. Key-value and document-oriented databases are dominant in ledger databases; however, only some approaches provide SQL databases. The differences in performance and architecture between centralised and decentralised structures are easily observed in the tables. While CLD does not require a consensus mechanism, decentralised permissioned blockchain-based database necessitates the consensus mechanism to reach agreement between participants in the network. The other observable divergence between the two structures is the implementation of smart contracts. The API interfaces in the centralised structure is generally more easily modified and expanded then the decentralised structure, where necessary software updates must be completed separately on every node, which are maintained by given enterprises. As a result, smart contracts provide a better approach due to their ability to be efficiently deployed on blockchain. Thus, similarly to an API interface update, they are accessible by all participants. As a result of the mentioned differences, summarising these technologies into CLDs and decentralised permissioned blockchain-based databases categories is logical.

Table 1. Overview of discussed Centralised Ledger Technology.

Name	LedgerDB
Database type	key-value
Transaction processing	Execute-commit-index
Smart contract supported	not supported
Transactions per second	100k

Table 2. Overview of discussed permissioned blockchain technologies.

Name	Blockchain Relational Database	BigchainDB	FalconDB
Database type	PostgreSQL	MongoDB	MySQL and IntegriDB
Replication model	Txn-based	Txn-based	Storage-based
Transaction processing	Execute and order in parallel	Order-then-commit	Order-then-execute
Consensus mechanism	Kafka	Tendermint	Tendermint
Smart contract supported	procedures as smart contract	not supported	Underlying Blockchain required
Transactions per second	1.5k	600	2k

Table 3. Overview of discussed permissioned blockchain technologies.

Name	ChainifyDB	BlockchainDB	Hyperledger Fabric
Database type	PostgreSQL and MySQL	Key-Value	CouchDB
Replication model	Txn-based	Storage-based	Txn-based
Transaction processing	Whatever Ledger Consensus	Order-then-execute	Execute-order-validate
Consensus mechanism	Kafka	Proof of Work	Raft
Smart contract supported	not supported	Underlying Blockchain required	supported
Transactions per second	1k	<100	600

The information found in the table relates to the setup of experiments corresponding to the given transactions per second. Some mentioned information is adjustable depending on the specification of the given technology. The transactions per second measurement is a median value to be able to order the performance of technologies. Transaction throughput depends on the type and size of transaction, the use of smart contracts, the bandwidth, and the discussed variations in architectural structure. The data found in the transactions per second row are calculated based on numerous sources from articles [54,55] and research [21,32,42,45,46]. During experiments, transactions included both read and write operations. The configurations for each discussed ledger database's experiment:

- **BigchainDB:** four nodes were in the same Azure data centre. Each node had 2nd Generation Intel Xeon Platinum 8272CL processor and Intel Xeon Platinum 8168 processor and 4 GB RAM. Three different types of test were done with one million and 16.0000 transactions.
- **BlockchainDB:** four nodes were in the same Azure data center with 16 vcpus and 32 GB memory. All experiments used an Ethereum backend. During experiments 4000 and 8000 transactions were sent. Experiments were run for online verification, offline verification and no verification.

- **ChainifyDB:** Two types of Nodes used, one being Two quad-core Intel Xeon CPU E5-2407 running at 2:2 GHz, equipped with 48 GB of DDR3 RAM, and the other Two hexa-core Intel Xeon CPU X5690 running at 3:47 GHz, equipped with 192 GB of DDR3 RAM. There were three organizations in experiments. Each organization owned two nodes. Experiments were for different consensus mechanism configuration. During the experiments, a hundred thousand transactions were sent.
- **Hyperledger Fabric:** five or ten nodes were in the same IBM cloud. Each node had 3.8 GHz Intel Xeon-CoffeeLake (E-2174G-Quadcore) processor and 2× 32 GB Hynix 32 GB DDR4 2Rx8 NON REG RAM. Experiments used different smart contracts as empty contract, asset read, and asset creation, written in GO language and using CouchDB as world state database.
- **FalconDB:** five server nodes were in the same cloud. Each node had 2.4 GHz Ten-core Intel E5-2640v4 processor and 64 GB DRAM. During experiments the authentication process was skipped. Tendermint was used as an existing blockchain.
- **Blockchain Relational Database:** from 4 nodes to 16 nodes were in different clouds. Each node had with 32 vCPUs of Intel Xeon E5-2683 v3 2.00 GHz and 64 GB of memory.
- **LedgerDB:** measurements were in two-node cluster. Each node had Intel Xeon Platinum 2.5 GHz CPU, 32 GB RAM.

5.1. Centralised Ledger Database

CLDs have a mandatory requirement of trust in a third party, which facilitates the database service. LedgerDB provides opportunities, interfaces, the deletion or modification of stored data in the database, as well as the immutability of the ledger. The discussed CLD technology, LedgerDB, uses TSA peg protocol opposed to LSP. In the case of AWS Quantum Ledger Database [56], full trust in the LSP is necessary. As a result, malicious behaviour from the LSP or collusion between participants and the LSP is not noticeable for participants. In contrast, TSA peg protocol eliminates the aforementioned issues but causes a new dependency on a third party. As has been mentioned, the use of a consensus mechanism is not required in databases with a centralised structure, e.g., LedgerDB does not use one, in contrast with the discussed approaches with a decentralised structure. The replication model takes place at a higher level, and it does not fall under the LedgerDB's tasks.

LedgerDB's performance rises above the discussed decentralised databases, but it has one of the best performances among centralised ledger databases as well, for example, better than ProvenDB [57] and AWS QLDB. One of the main causes of this excellent performance is the implementation of an index phase in the transaction processing. In the last step, LedgerDB builds different index structures to later access data faster. The user-specified label, clue index, uses this one of pre-built index structures. Smart contract is not implemented in LedgerDB, but it is barely missed, as interface changes are easier and less costly than in an permissioned blockchain-based database due to its centralised structure.

5.2. Permissioned Blockchain-Based Databases

When it comes to the implementation of decentralised blockchain, the comparisons diverge more so than with centralised systems. The primary reason for this is that the maintenance of trustless decentralised networks through numerous nodes from different enterprises provides a variety of opportunities for the use of different consensus mechanisms and transaction processing models in an attempt to achieve better performance in a secure and tamper-proof network. The majority of the discussed blockchain technologies implement pluggable consensus mechanisms, which are naturally changeable. This is also true regarding the mentioned databases that require the whitepaper, for example, ChainfyDB, where there is a requirement for the support of the SQL-99 compliant.

The parallelisation of the transactions and the separation of tasks into different nodes, such as specialised validation or execution nodes, can achieve better performance. Moreover, the use of transaction monitoring can also be a tool to improve performance, as in this way problematic transactions which cannot be committed in a given block are detected

earlier, and thus, the amount of data in processing and in communication are reduced. Transactions that conflict with one another or that are not valid are labelled as problematic. However, the use and implementation of parallelisation must be done carefully, as it can cause issues, in most cases being the modification of transaction ordering. Hyperledger Fabric's mentioned expanded version, FastFabric, in which although performance was improved through the implementation of parallelisation during transaction processing and validation, it nevertheless experienced this common issue. In another expanded version of Hyperledger Fabric, Nexledger Accelerator, which offers the best performance of decentralised networks, it also encounters numerous issues. As Nexledger Accelerator adds a standalone server between the user and Hyperledger Fabric, it has reduced the level of decentralisation present in Hyperledger Fabric due to more centralised processing. This mentioned standalone server is able to modify transactions and mis-sign transactions. The largest issues it faces as a result of the use of this server is non-standard APIs and further development tasks, as wrappers must be written for chaincode usage. Thus, the use of Nexledger Accelerator must be well considered as it best serves specific, and often extraordinary, purposes. Nexledger Accelerator is most suited to cases in which the network is maintained by a single enterprise, the senders of transactions are not relevant, and high transaction throughput is crucial.

The aforementioned conflicting transactions are best managed in BRD, where SSI is implemented to attain a highly competitive transaction throughput per block. In the instances of the mentioned permissioned blockchain-based databases, there are a variety of validation models, for example, in FalconDB, where it may take up to hours to validate the commit of a transaction to the blockchain, as a result of the use of an incentive model. Thus, the maintainers must consider whether the detection of malicious behaviour requiring hours is appropriate for their predefined conditions regarding the security of the blockchain network. The use of this incentive model enables lightweight nodes to also participate in the network, which is able to validate request proofs. Similar time-consuming validation issues are also present in BlockchainDB.

The replication models of permissioned blockchain-based databases can be categorised as either transaction-based or storage-based. The transaction-based replication model stores transactions in the ledger. These systems are based on three primary characteristics; each node having its own database, global ordering being achieved through consensus, and the storing of transactions on the ledger, which serves as a secure shared log. Hyperledger Fabric uses a transaction-based replication model as well; however, it varies from the aforementioned approaches, as it is a complete blockchain system. As a result of this, BlockchainDB is also able to implement it as an underlying blockchain. In this case, the use of databases is facilitated through smart contracts. In contrast, the storage-based replication model replicates storage operations using a consensus mechanism. These two, FalconDB and BlockchainDB, decentralised permissioned blockchain-based databases use established blockchain technology, as well as implementing interfaces with a database feature set on top of this. Naturally, the storage-based replication model has higher storage usage than the transaction-based.

In recent times, more and more blockchain networks use sharding, which allows for stored data to be distributed onto various nodes, but not all. However, of the mentioned technologies, there is only one that allows this feature. BlockchainDB is unique and rather rudimentary in the implementation of this sharding feature as it can only achieve this through the use of already established blockchain technologies.

Much examination and comparison of the various aforementioned consensus mechanisms have already been discussed in previous studies [58–60]; thus, they will not be expanded on here. The consensus mechanisms used in centralised blockchain technologies all share similar features, generally being deterministic.

The mentioned various transaction processing models indicate that effective performance can be best achieved through the prioritisation of the validation of and conflict detection in transaction processing. As a result, problematic or conflicting transactions,

which are not committed in a given block, can be abandoned as soon as possible. Furthermore, the parallelisation of validation and execution increases the transaction throughput of the blockchain network.

6. Conclusions

In the past few years, both interest and development in the field of ledger databases, especially blockchain, have been growing exponentially, thanks to the increased attention from both academia and industry sectors. However, at present, less consideration is given to permissioned blockchain in contrast with permissionless blockchain. It can be observed in given technologies that they gain inspiration from permissionless blockchain networks, which are occasionally questionable, as the structural differences between these two blockchain types, specifically their either public or private nature, may not always best serve the interests of the participants. Alongside this, the question arises for private entities of the necessity of a blockchain-based network without any third parties. The introduction of CLD facilitates the use of blockchain-esque technologies for smaller private enterprises, which is a more economical option for these enterprises, as opposed to the implementation and maintenance of a whole permissioned blockchain technology. The discussed technologies, when compared to databases, can be considered to be less effective and still in their infancy, as the majority does not attempt to replace traditional database technologies, even if they were capable of doing so. Therefore, the use of permissioned blockchain or CLD as a database manager requires much consideration [61,62] as to whether it meets the requirements of the system specification.

Author Contributions: Conceptualization, D.L.F. and A.K.; methodology, D.L.F. and A.K.; validation, D.L.F. and A.K.; investigation, D.L.F. and A.K.; writing—original draft preparation, D.L.F. and A.K.; writing—review and editing, D.L.F. and A.K.; supervision, A.K.; project administration, A.K. All authors have read and agreed to the published version of the manuscript.

Funding: The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002). This research was also supported by grants of "Application Domain Specific Highly Reliable IT Solutions" project that has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2019. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 30 July 2021).
2. Ahram, T.; Sargolzaei, A.; Sargolzaei, S.; Daniels, J.; Amaba, B. Blockchain technology innovations. In Proceedings of the 2017 IEEE Technology & Engineering Management Conference (TEMSCON), Santa Clara, CA, USA, 8–10 June 2017.
3. Risius, M.; Spohrer, K. A blockchain research framework. *Bus. Inf. Syst. Eng.* **2017**, *59*, 385–409. [CrossRef]
4. Grech, A.; Camilleri, A.F. *Blockchain in Education*; Publications Office of the European Union: Luxembourg, 2017.
5. Agbo, C.C.; Mahmoud, Q.H.; Eklund, J.M. Blockchain technology in healthcare: A systematic review. In *Healthcare*; Multidisciplinary Digital Publishing Institute: Basel, Switzerland, 2019; Volume 7, p. 56.
6. Tanwar, S.; Parekh, K.; Evans, R. Blockchain-based electronic healthcare record system for healthcare 4.0 applications. *J. Inf. Secur. Appl.* **2020**, *50*, 102407. [CrossRef]
7. Hyla, T.; Pejaś, J. eHealth integrity model based on permissioned blockchain. *Future Internet* **2019**, *11*, 76. [CrossRef]
8. Hölbl, M.; Kompara, M.; Kamišalić, A.; Nemec Zlatolas, L. A systematic review of the use of blockchain in healthcare. *Symmetry* **2018**, *10*, 470. [CrossRef]
9. Bigini, G.; Freschi, V.; Lattanzi, E. A review on blockchain for the internet of medical things: Definitions, challenges, applications, and vision. *Future Internet* **2020**, *12*, 208. [CrossRef]
10. Tapscott, A.; Tapscott, D. How blockchain is changing finance. *Harv. Bus. Rev.* **2017**, *1*, 2–5.
11. Treleaven, P.; Brown, R.G.; Yang, D. Blockchain technology in finance. *Computer* **2017**, *50*, 14–17. [CrossRef]

12. Shekhtman, L.; Waisbard, E. EngraveChain: A Blockchain-Based Tamper-Proof Distributed Log System. *Future Internet* **2021**, *13*, 143. [CrossRef]
13. Ibba, S.; Pinna, A.; Lunesu, M.I.; Marchesi, M.; Tonelli, R. Initial coin offerings and agile practices. *Future Internet* **2018**, *10*, 103. [CrossRef]
14. Cocco, L.; Pinna, A.; Marchesi, M. Banking on blockchain: Costs savings thanks to the blockchain technology. *Future Internet* **2017**, *9*, 25. [CrossRef]
15. Reyna, A.; Martín, C.; Chen, J.; Soler, E.; Díaz, M. On blockchain and its integration with IoT. Challenges and opportunities. *Future Gener. Comput. Syst.* **2018**, *88*, 173–190. [CrossRef]
16. Bellini, A.; Bellini, E.; Gherardelli, M.; Pirri, F. Enhancing IoT data dependability through a blockchain mirror model. *Future Internet* **2019**, *11*, 117. [CrossRef]
17. Tseng, L.; Yao, X.; Otoum, S.; Aloqaily, M.; Jararweh, Y. Blockchain-based database in an IoT environment: Challenges, opportunities, and analysis. *Clust. Comput.* **2020**, *23*, 2151–2165. [CrossRef]
18. Du, Y.; Wang, Z.; Leung, V. Blockchain-Enabled Edge Intelligence for IoT: Background, Emerging Trends and Open Issues. *Future Internet* **2021**, *13*, 48. [CrossRef]
19. Bouras, M.A.; Lu, Q.; Dhelim, S.; Ning, H. A Lightweight Blockchain-Based IoT Identity Management Approach. *Future Internet* **2021**, *13*, 24. [CrossRef]
20. Li, Y. An integrated platform for the internet of things based on an open source ecosystem. *Future Internet* **2018**, *10*, 105. [CrossRef]
21. Yang, X.; Zhang, Y.; Wang, S.; Yu, B.; Li, F.; Li, Y.; Yan, W. LedgerDB: A centralized ledger database for universal audit and verification. *Proc. VLDB Endow.* **2020**, *13*, 3138–3151. [CrossRef]
22. Mohan, C. State of public and private blockchains: Myths and reality. In Proceedings of the 2019 International Conference on Management of Data, Amsterdam, The Netherlands, 30 June–5 July 2019; pp. 404–411.
23. Helliar, C.V.; Crawford, L.; Rocca, L.; Teodori, C.; Veneziani, M. Permissionless and permissioned blockchain diffusion. *Int. J. Inf. Manag.* **2020**, *54*, 102136. [CrossRef]
24. Zheng, W.; Zheng, Z.; Chen, X.; Dai, K.; Li, P.; Chen, R. Nutbaas: A blockchain-as-a-service platform. *IEEE Access* **2019**, *7*, 134422–134433. [CrossRef]
25. Singh, J.; Michels, J.D. Blockchain as a service (BaaS): Providers and trust. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), London, UK, 23–27 April 2018.
26. Ruan, P.; Dinh, T.T.A.; Loghin, D.; Zhang, M.; Chen, G.; Lin, Q.; Ooi, B.C. Blockchains vs. Distributed Databases: Dichotomy and Fusion. In Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data, Xi'an, China, 20– 25 June 2021; pp. 543–557.
27. Muzammal, M.; Qu, Q.; Nasrulin, B. Renovating blockchain with distributed databases: An open source system. *Future Gener. Comput. Syst.* **2019**, *90*, 105–117. [CrossRef]
28. Bergman, S.; Asplund, M.; Nadjm-Tehrani, S. Permissioned blockchains and distributed databases: A performance study. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5227. [CrossRef]
29. Raikwar, M.; Gligoroski, D.; Velinov, G. Trends in Development of Databases and Blockchain. In Proceedings of the 2020 Seventh International Conference on Software Defined Systems (SDS), Paris, France, 30 June–3 July 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 177–182.
30. Gilbert, S.; Lynch, N. Perspectives on the CAP Theorem. *Computer* **2012**, *45*, 30–36. [CrossRef]
31. Zhang, K.; Jacobsen, H.A. Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–5 July 2018.
32. Nathan, S.; Govindarajan, C.; Saraf, A.; Sethi, M.; Jayachandran, P. Blockchain meets database: Design and implementation of a blockchain relational database. *arXiv* **2019**, arXiv:1903.01919.
33. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
34. Berenson, H.; Bernstein, P.; Gray, J.; Melton, J.; O'Neil, E.; O'Neil, P. A critique of ANSI SQL isolation levels. *arXiv* **2007**, arXiv:cs/0701157.
35. Adya, A.; Liskov, B.; O'Neil, P. Generalized isolation level definitions. In Proceedings of the 16th International Conference on Data Engineering (Cat. No. 00CB37073), San Diego, CA, USA, 28 February–3 March 2000; IEEE: Piscataway, NJ, USA, 2000; pp. 67–78.
36. McConaghy, T.; Marques, R.; Müller, A.; De Jonghe, D.; McConaghy, T.; McMullen, G.; Henderson, R.; Bellemare, S.; Granzotto, A. Bigchaindb: A Scalable Blockchain Database; White Paper, BigChainDB. 2016. Available online: https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf (accessed on 31 July 2021).
37. Yiu, N.C. Decentralizing Supply Chain Anti-Counterfeiting Systems Using Blockchain Technology. *arXiv* **2021**, arXiv:2102.01456.
38. Rejeb, A.; Keogh, J.G.; Treiblmaier, H. Leveraging the internet of things and blockchain technology in supply chain management. *Future Internet* **2019**, *11*, 161. [CrossRef]
39. Buchman, E. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains. Ph.D. Thesis, University of Guelph, Guelph, Canada, June 2016.

40. Lamport, L.; Shostak, R.; Pease, M. The Byzantine generals problem. In *Concurrency: The Works of Leslie Lamport*; ACM: New York, NY, USA, 2019; pp. 203–226.
41. Lab, M.R. Monero. 2021. Available online: https://www.getmonero.org/resources/research-lab/ (accessed on 31 July 2021).
42. Peng, Y.; Du, M.; Li, F.; Cheng, R.; Song, D. FalconDB: Blockchain-based collaborative database. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Portland, OR, USA, 14–19 June 2020; pp. 637–652.
43. Martel, C.; Nuckolls, G.; Devanbu, P.; Gertz, M.; Kwong, A.; Stubblebine, S.G. A general model for authenticated data structures. *Algorithmica* **2004**, *39*, 21–41. [CrossRef]
44. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
45. Schuhknecht, F.M.; Sharma, A.; Dittrich, J.; Agrawal, D. Chainifydb: How to blockchainify any data management system. *arXiv* **2019**, arXiv:1912.04820.
46. El-Hindi, M.; Binnig, C.; Arasu, A.; Kossmann, D.; Ramamurthy, R. BlockchainDB: A shared database on blockchains. *Proc. VLDB Endow.* **2019**, *12*, 1597–1609. [CrossRef]
47. Ethereum 2.0. 2021. Available online: https://ethereum.org/en/whitepaper/ (accessed on 31 July 2021).
48. Zamani, M.; Movahedi, M.; Raykova, M. Rapidchain: Scaling blockchain via full sharding. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 931–948.
49. Tanenbaum, A.S. Distributed operating systems anno 1992. What have we learned so far? *Distrib. Syst. Eng.* **1993**, *1*, 3. [CrossRef]
50. Lin, J.J.; Lee, Y.T.; Wu, J.L. The Effect of Thickness-Based Dynamic Matching Mechanism on a Hyperledger Fabric-Based TimeBank System. *Future Internet* **2021**, *13*, 65. [CrossRef]
51. Sharma, A.; Schuhknecht, F.M.; Agrawal, D.; Dittrich, J. Blurring the lines between blockchains and database systems: The case of hyperledger fabric. In Proceedings of the 2019 International Conference on Management of Data, Amsterdam, The Netherlands, 30 June–5 July 2019; pp. 105–122.
52. Gorenflo, C.; Lee, S.; Golab, L.; Keshav, S. FastFabric: Scaling hyperledger fabric to 20,000 transactions per second. *Int. J. Netw. Manag.* **2020**, *30*, e2099. [CrossRef]
53. Samsung; IBM. Accelerating Throughput in Permissioned Blockchain Networks. 2019. Available online: https://github.com/nexledger/accelerator/ (accessed on 31 July 2021).
54. McConaghy, T. BigchainDB Performance Experiments. 2018. Available online: https://blog.bigchaindb.com/and-were-off-to-the-races-1aff2b66567c/ (accessed on 31 July 2021).
55. Hyperledger Fabric Performance Experiments. 2021. Available online: https://hyperledger.github.io/caliper-benchmarks/fabric/performance/ (accessed on 31 July 2021).
56. Service, A.W. Amazon Quantum Ledger Database (qldb). 2019. Available online: https://aws.amazon.com/qldb (accessed on 31 July 2021).
57. ProvenDB. Provendb: A Blockchain Enabled Database Servic. 2019. Available online: https://www.provendb.com/litepaper/ (accessed on 31 July 2021).
58. Cachin, C.; Vukolić, M. Blockchain consensus protocols in the wild. *arXiv* **2017**, arXiv:1707.01873.
59. Panda, S.S.; Mohanta, B.K.; Satapathy, U.; Jena, D.; Gountia, D.; Patra, T.K. Study of blockchain based decentralized consensus algorithms. In Proceedings of the TENCON 2019–2019 IEEE Region 10 Conference (TENCON), Kochi, India, 17–20 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 908–913.
60. Wang, W.; Hoang, D.T.; Hu, P.; Xiong, Z.; Niyato, D.; Wang, P.; Wen, Y.; Kim, D.I. A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access* **2019**, *7*, 22328–22370. [CrossRef]
61. Golosova, J.; Romanovs, A. The advantages and disadvantages of the blockchain technology. In Proceedings of the 2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), Vilnius, Lithuania, 8–10 November 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.
62. Chowdhury, M.J.M.; Colman, A.; Kabir, M.A.; Han, J.; Sarda, P. Blockchain versus database: A critical analysis. In Proceedings of the 2018 17th IEEE International Conference On Trust, Security And Privacy in Computing And Communications/12th IEEE International Conference on Big Data Science And Engineering (TrustCom/BigDataSE), New York, NY, USA, 1–3 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1348–1353.