Renovating blockchain with distributed databases: An open source system

Muhammad Muzammal, Qiang Qu, Bulat Nasrulin

PII: S0167-739X(18)30873-2
DOI: https://doi.org/10.1016/j.future.2018.07.042
Reference: FUTURE 4360

To appear in: *Future Generation Computer Systems*

Received date : 11 April 2018
Revised date : 7 June 2018
Accepted date : 17 July 2018

Renovating Blockchain with Distributed Databases: An Open Source System

Muhammad Muzammal[a,b], Qiang Qu[a,*], Bulat Nasrulin[c]

[a]*Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences*
[b]*Department of Computer Science, Bahria University, Islamabad*
[c]*Shenzhen College of Advanced Technology, University of Chinese Academy of Sciences*

Abstract

A blockchain is a decentralised linked data structure that is characterised by its inherent resistance to data modification, but it is deficient in search queries primarily due to its inferior data formatting. A distributed database is also a decentralised data structure which features quick query processing and well-designed data formatting but suffers from data reliability. In this work, we showcase ChainSQL, an open-source system developed by integrating the blockchain with the database, i.e. we present a blockchain database application platform that has the decentralised, distributed and audibility features of the blockchain and quick query processing and well-designed data structure of the distributed databases. ChainSQL features a tamper-resistant and consistent multi-active database, a reliable and cost effective data-level disaster recovery backup and an auditable transaction log mechanism. The system is presented as an operational multi-active database along with the data-level disaster recovery backup and audibility features. A comprehensive experimental evaluation is performed to demonstrate the effectiveness of the system.

Keywords: Blockchain, Distributed Databases, Blockchain Application

1. Introduction

Digital or crypto currencies such as, Bitcoin [1], Ethereum [2], Ripple [3] and others, have recently witnessed a tremendous interest from the user as well as the developer community [4, 5, 6]. The crypto currencies are essentially smart contracts between users which are executed using a data structure referred to as 'blockchain'. Thus, a blockchain stores transactions whilst satisfying the following two constraints: (i) anyone should be able to write to the blockchain, and (ii) there should not be any centralised control.

A blockchain is a database and an application software on top of it [7] that dictates the data definition and data update mechanism for the blockchain. A blockchain not only allows

*Corresponding author
Email addresses: muzammal@siat.ac.cn;muzammal@bui.edu.pk (Muhammad Muzammal), qiang.qu@siat.ac.cn (Qiang Qu), bulat@siat.ac.cn (Bulat Nasrulin)

to add new data to the database but it also ensures that all the users on the network have exactly the same data. Thus, a blockchain is a distributed and decentralised linked data structure for data storage and retrieval which also ensures that the data is resistant to any modification.

One of the limitations of blockchain is its inherent deficiency in search query processing [8] primarily due to the linked data storage and the absence of a well-defined data indexing structure for various queries. Bitcoin, for instance, is the most notable blockchain network; however, it has two limitations: (i) it takes a considerable amount of time, possibly up to ten minutes, for a transaction to be issued and verified and the final confirmation may take up to an hour, and (ii) a new block can only be generated by miners which requires extensive computational efforts.

Databases, in addition to having a defined data structure are optimised for fast query processing, but are not resistant to data modifications [9]. More specifically, distributed databases have the following limitations: (a) database can be tampered either by a malicious user or by the database administrator, (b) the backup-based disaster recovery scheme of the database cannot be normally activated in the event of a system failure and causes data loss, and (c) the multiple copies of the database are not always entirely consistent and the data synchronisation operations are required to resolve data conflicts.

Therefore, a blockchain-based database system is desirable that has the features of the blockchain and the distributed databases combined together such that the inherent resistance of the blockchain to data modification and the query speed of the distributed databases are simultaneously achieved. This is not obvious as the blockchain and the distributed database are two different data structures and are designed to serve entirely different business needs. For example, whilst distributed database queries are designed based on the 'parallelism' paradigm, for instance, inter- and intra-query parallelism or multiplex approaches, blockchain is a strictly sequential data structure that is aimed at guaranteeing data integrity. Similarly, the privacy of the user data, although is a concern in the distributed databases, it is a design requirement in a blockchain. The presence of the notion of 'trust' in a distributed database and its absence in a blockchain is also a primary design consideration and has contrasting consequences.

However, the decentralised and trustless data storage in blockchain could be employed to advantage as follows. As a node in the blockchain network has its own copy of the blockchain data, if node X requires to execute fast queries on the blockchain, it can execute the local copy of the blockchain to generate a local database, and thus the integrity of the data is verified by the blockchain and the fast queries are executed on the local database.

In this work, we showcase ChainSQL[1], a novel database powered blockchain system that integrates the blockchain with the distributed databases to yield a system that has the integrity of the blockchain and the fast query processing of the distributed databases. We illustrate three usecases of ChainSQL; each of which is implemented as a middleware between the enterprise application and the underlying database:

[1]The source code for ChainSQL is available online at: https://github.com/ChainSQL/chainsqld

(1) The first usecase is a multi-active database middleware that connects the enterprise application with the database system. The middleware provides both symmetric and asymmetric encryption for data security. The expandability of the blockchain network is achieved by integrating a new node into the blockchain network seamlessly and when the new node is established as a 'valid' production node, it can participate in operations such as consensus and synchronous data writing, similar to the existing production nodes.

(2) The second usecase is a database disaster recovery middleware that connects the database production nodes with the disaster recovery nodes. During disaster recovery, a backup node is elevated to a production node and it is ensured by the disaster recovery centre that the backup node has exactly the same data as the production node. In case of a production node failure, the user seamlessly switches to the 'new' production node which is a node in the recovery centre to complete the task.

(3) The third usecase is an audibility feature that enables flexible access control along with tracing support. The audibility is controlled by the owner of the data and is performed by a read-only 'grant' access to the auditor that can read the data for the auditing purposes.

The details about the aforementioned usecases are presented in Section 5.

ChainSQL *Significance.* Distributed databases feature enhanced transparency, easy expansion, better data-loss resistance and optimised query performance, but suffer from data reliability and consistency. The most compelling requirement for a distributed database is the notion of 'trust' as it requires a centralised control mechanism to maintain the authenticity of the data. Blockchain, on the contrary, is a 'trustless' data structure that ensures data reliability and integrity at its core. However, blockchain lacks in throughput and query performance. We propose the integration of the distributed databases with the blockchain. We argue that this is a promising idea and should be incorporated into the blockchain systems to address the challenges that have been already resolved by the distributed databases but still require attention in blockchain systems. ChainSQL solves the data integrity and reliability issues of the distributed databases by using the blockchain technology and still keeps the fast query processing of the distributed databases.

ChainSQL *Highlights.* ChainSQL features a secure design due to the authorisation requirement to access the personal user data. The transactions are stored in the blockchain whereas the actual data is stored in the database. The data is distributed to improve service availability. Many-to-one disaster recovery architecture allows a single backup centre to be used with multiple production nodes. The backup database can be operated without data recovery. Thus, ChainSQL not only provides the instantaneity of the traditional database but also the security of the blockchain. It can be easily configured with commonly used traditional databases such as MySQL[2] by way of APIs. As the database log is immutable, the history database actions are preserved, therefore, it allows auditing using the data stored

[2]www.chainsql.net/api_mysql.html

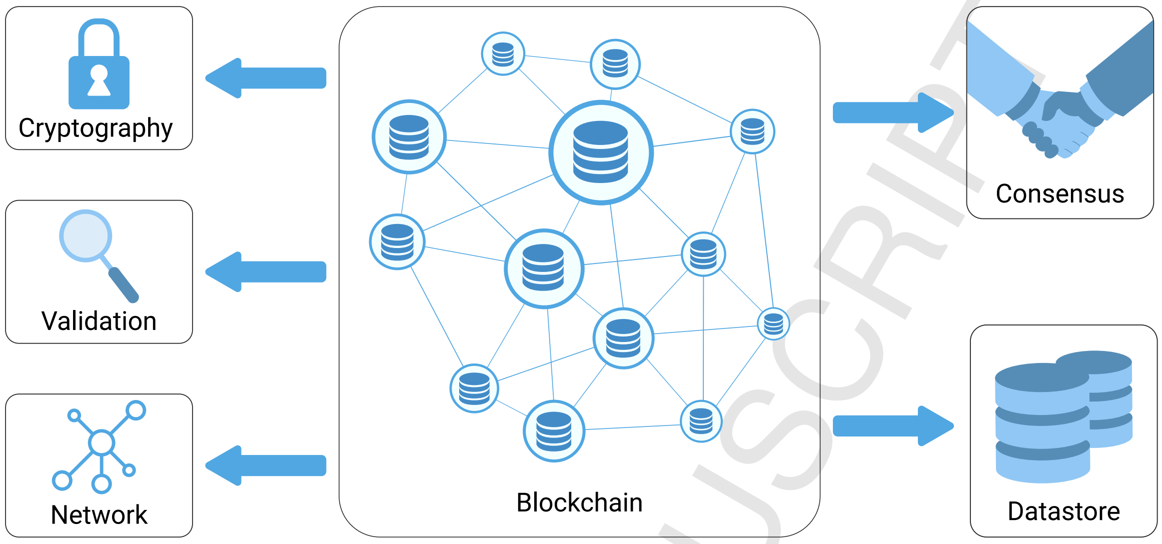


Figure 1: The components of a typical blockchain system.

in the blockchain. Another exciting feature is the integration of the blockchain with new applications via ChainSQL interface rather than the database interface. This enables incorporation of the security and audibility features of the blockchain into the existing database systems.

The rest of this paper is organised as follows. An overview of blockchain systems and related concepts are stated in Section 2. ChainSQL as a blockchain database application platform is presented in Section 3 and the consensus algorithm is described in Section 4. The ChainSQL usecases are elaborated in Section 5. The system is empirically evaluated in Section 6, and Section 7 concludes this work.

2. Blockchain Technology and Systems

Blockchain technology has come to forth recently. A variety of applications using the blockchain technology are being proposed spanning a multitude of domains including finance, healthcare, supplychain, online games, social media and others. However, current blockchain systems suffer from issues such as low throughput and inefficient query processing, therefore, the applicability of the blockchain technology is still limited in different application domains. In this section, we overview the blockchain technology and systems and discuss the recent developments in the field.

We first discuss the architecture of blockchain systems.

2.1. Blockchain Architecture

A blockchain is a cryptographically-secure transactional singleton machine with shared state [2] and is packaged as a set of concepts shown in Figure 1. An outline of blockchain components is as below:

1. *Datastore* is the blockchain data structure that holds all the blockchain data.
2. *Consensus* is the blockchain agreement mechanism that ensures the data integrity in the system.
3. *Validation* is the process that ensures the correct state transition in the blockchain.
4. *Peer-to-peer network* is a distributed computing environment that performs the blockchain system operations (1-3 above).
5. *Cryptography* ensures the security and privacy of the data in blockchain.

We now proceed with some details.

2.1.1. Datastore

Datastore is a state-machine replication [10] based sequential-access data structure which has the entire blockchain data replicated on each node in the blockchain network. The transactions are assembled into blocks such that every consequent block is connected to the previous block via a hash-value. As the blocks are only forward reachable, a change in a blocks' hash-value affects all the subsequent blocks, and thus violates the integrity of the blockchain. As shown in Figure 2, a block has two parts: (i) a header with metadata and (ii) a set of associated transactions. The integrity of the blockchain is established by traversing the headers of the blocks in the chain whereas the current states of the accounts and transactions are needed for validation. The size of the blockchain is a concern due to the sequential access, therefore, techniques similar to Merkle tree [11] have been proposed to speedup the validation process. Similarly, graph based approaches [12, 13, 14] have also been proposed that store blocks and transactions in a graph format rather than a list.

2.1.2. Consensus

A blockchain is a trustless decentralised entity that has consensus as its operational engine. Consensus algorithms have long been studied in distributed systems and recently are drawing attention from researchers for blockchain systems. See the study [15] for a comprehensive overview of consensus algorithms for blockchain systems.

Typically, a blockchain is a state transition system where a state is the current status of the stakeholders in the blockchain. As shown in Figure 3, each new transaction changes the state of the system. A set of transactions are bundled into blocks which are signed and hashed together. Thus, blocks define the blockchain transition from one state to another. It should be clear that the consensus protocol is fundamentally different across blockchains and has motivations in technology as well as business. However, a consensus algorithm has to define a set of rules to achieve an agreement on the transactions and the order in which they appear. It should also be noted that the consensus problem in asynchronous systems with stochastic processes is known to be 'hard' for deterministic termination [16]. Therefore, a consensus algorithm has to make some simplifying assumptions for termination. Partial synchrony, process reliability, probabilistic termination are examples of such assumptions.

The Proof-of-Work (PoW) [17] based algorithm proposed by Nakomoto [1] for the Bitcoin is probably the most well-known consensus algorithm. PoW is a probabilistic consensus algorithm where miners solve cryptographic challenges, which are hard to solve but easy

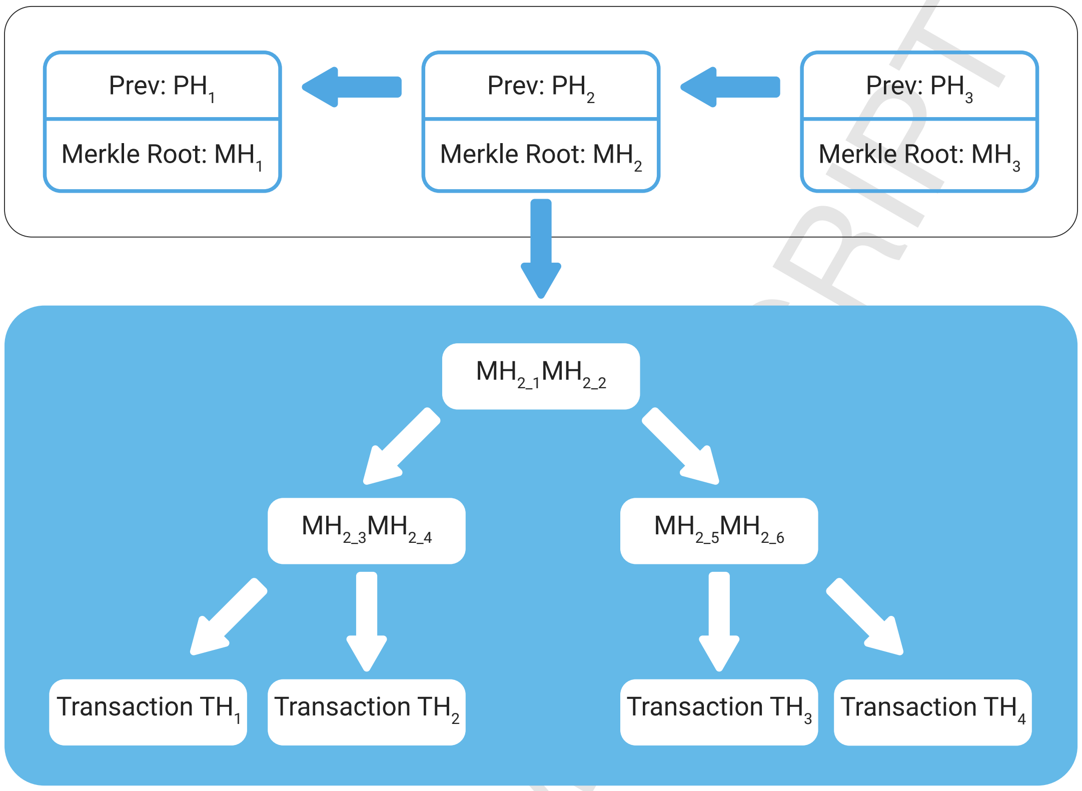


Figure 2: A typical datastore and associated block representation.

to verify, and are rewarded accordingly. To solve these cryptographic challenges requires extensive computational efforts; therefore, the use of PoW consensus in real-life application scenarios is limited due to efficiency and scalability constraints. An alternative to PoW is the concept of Proof-of-Stake (PoS) where a peer is rewarded based on the stakes in the system. PoS works on the assumptions that (i) if an adversary acts maliciously it looses its stake, and (ii) the adversary nodes can not exceed 1/3 of the total nodes. PoS solves the energy consumption problem of PoW but only works under some assumptions [18].

Examples of emerging consensus algorithms inspired from practical Byzantine fault tolerance [19] include Hyperledger Fabric [20]. Practical Byzantine fault tolerant consensus algorithms [19] rely on 3-step all-to-all communication with complexity $O(n^2)$, where n is the number of nodes in the network. Recent proposals include a leader-based consensus algorithm [20] where a leader is elected for a number of rounds to optimize communication workload by decreasing the average network communication complexity.

2.1.3. Validation

The integrity of the blockchain is maintained by a validation process which aims to avoid issues such as double spending in crypto-currencies. For crypto-currencies [21], a transaction is validated as a set of following checks: (i) transaction is cryptographically valid, i.e. it

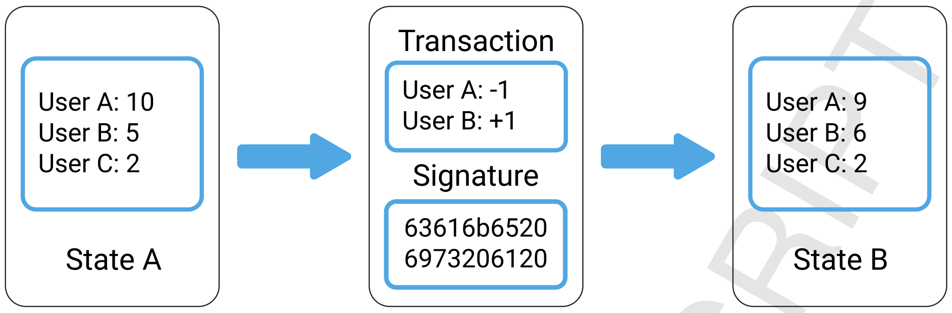


Figure 3: The consensus process as a state-machine replication. When a transaction is applied, it changes the state of the peer from state A to state B.

has a verifiable signature, (ii) transaction format is valid, i.e. all the transaction fields have a valid range, and (iii) transaction state is valid, i.e. the transaction spending constraints are satisfied. It is obvious that the receiver's existential identity should also be verified to avoid financial loss. The validation mechanism employed by Ethereum is by way of smart contracts. A smart contract is an agreement between interested parties and is executed only if a set of pre-defined constraints are satisfied. A smart contract is executed in parallel for a distributed environment [22]. Smart contracts must be defined in a transparent and shared space to provide better security for the system [23]. A study [24] presents a review on the prevalent validation protocols including smart contracts. Sergey et al. [25] show that smart contract implementations lack formal methods for verification and it is still questionable whether current usecases need a Turing-Complete feature as it creates additional attack vectors.

2.1.4. Blockchain Network

Blockchain is a peer-to-peer network for information exchange between nodes where transactions are relayed in the network using a secure broadcasting protocol. The stability of information propagation is a concern in the network and Denial-of-Service (DoS) attacks are common in public blockchains, such as Bitcoin and Ethereum. For example, in Bitcoin network, large mining pools can attack smaller pools to get more incentives [26]. In Ripple [27], a node implements a Unique Node List (UNL) and thus DoS attack is avoided as traffic from the unwanted nodes is filtered out.

The protocols in blockchain networks are either for dense networks or for sparse networks. Dense networks expand the peer connections to decrease the probability of malicious channels along with the usage of authenticated channels with the guarantee that if the message is signed it is delivered from the correct sender. In sparse networks, failures are measured as distance between two Byzantine nodes. Transactions are spread faster without repetition in a sparse network with a gossip protocol [28], but it is harder to achieve a deterministic Byzantine fault tolerant agreement.

2.1.5. Security and Privacy

Security and privacy of the blockchain systems primarily relies on cryptography and cryptographic assumptions. Blockchains are required to authorize users, define the account balance and establish the validity of a transaction. One of the most important instruments for the purpose are digital signatures [29]. A digital signature provides three properties: (i) verification, i.e. it is easy to verify the authenticity of the signature, (ii) non-forgeability, i.e. no one should be able to copy or forge it, and (iii) non-repudiation, i.e. once you sign an object, it is impossible to un-sign it.

Digital signatures based on symmetric cryptography are a common practice. A private key k is used to sign a transaction with hash-value H. In addition to a signature, each transaction also has the public key of the user that is used for user identification. A digital signature scheme based on elliptic curves [30] allows multi-signature transactions.

An example usage of symmetric cryptography in a blockchain system is shown in Figure 4. Sender generates $(public, private)$ key pair. Note that in a permission-less blockchain, user can generate many key pairs as consensus protocol does not depend on the public key whereas in permissioned blockchain, public key must be known to all the validating peers. Message is signed and sent in the form of a transaction which is verified at the receiver by using a verification algorithm that is compatible with a signing algorithm used by the sender. Thus, a symmetric verification algorithm requires public key of the sender to identify the validity of the signature. To achieve privacy, crypto-currencies use coin shuffling with mixing [31] or zk-SNARKs [32]. More discussion on privacy and security issues in blockchain systems can be found in the study [33].

2.2. Blockchain Evolution

The Bitcoin [1] proposal triggered the introduction of many crypto-currencies, and consequently, many blockchain systems are being proposed. We now give an overview of the blockchain evolution.

2.2.1. Blockchain 1.0

The recent interest in blockchain system started with the introduction of Bitcoin proposed by Nakomoto [1]. The aim of the Bitcoin was to introduce transparent cross-border payments. Bitcoin scripting language extends the system capability to represent asset manipulations, however it makes development of Bitcoin applications difficult as is evident by a limited number of applications. Bitcoin issues include high power consumption due to PoW consensus, low throughput, high latency and inability to make rich queries. These drawbacks compelled the introduction of Blockchain 2.0.

2.2.2. Blockchain 2.0

Ethereum [2] introduced smart contracts that allow to run code written in Solidity on top of blockchain. Ethereum maintains *account* and *state* Particia Merkle Tree and the smart contract values in a $(key, value)$ repository. Malicious never-ending smart contract are prohibited by the introduction of 'Ether'. Ether is the Ethereum fuel and is used for executing smart contracts. The creation of Ethereum based business applications is restrictive because

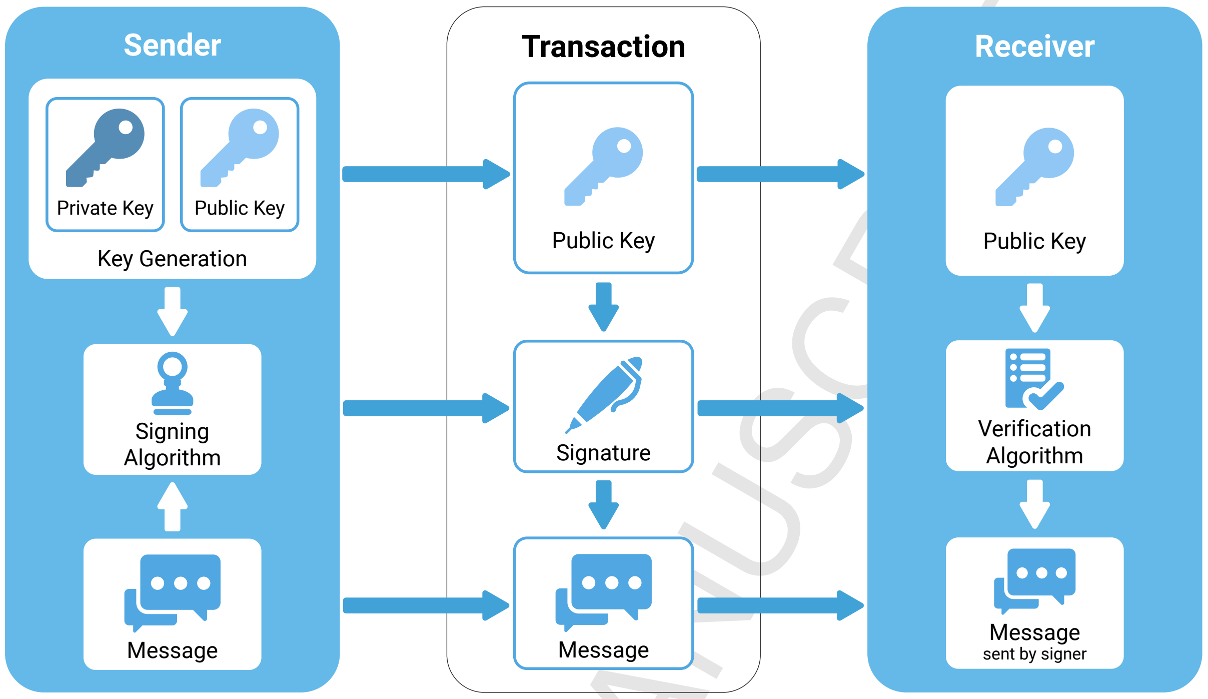


Figure 4: An overview of the cryptography scheme for transactions in a blockchain. Sender uses private key to sign a transaction and the public key of the sender is used to establish the transaction validity.

of two reasons: (i) the design of smart contract module is not optimized for a particular business usecase, and (ii) there is no distinction between the data and the code storage.

Ripple [27] solves the scalability problem in Ethereum and Bitcoin by utilizing semi-centralized ledger. Ripple consensus is based on a unique node list which is a set of trusted nodes and these nodes participate in the consensus process. The consensus process itself is based on validation and votes and only requires as few seconds as demonstrated by Armknecht et al. [34]. The business case for Ripple is interbank exchanges and financial applications. Ripple supports a variety of databases and data analytics.

2.2.3. Beyond Blockchain 2.0

The study [5] is a detailed note on data processing for blockchain systems. A number of initiatives have been taken from the data storage and retrieval perspective. For example, Hyperledger Fabric [35] is an initiative that allows to create a private blockchain for an enterprise. Fabric implements a pluggable consensus where each specific Fabric network can utilize different consensus algorithm to achieve required level of fault tolerance. Vukolić [36] argued that classical Byzantine Fault Tolerance consensus algorithms should be considered to achieve stronger consistency. Gaetani et al. [37] and Blockstack [38] stress the need of better data integrity with the current cloud environments.

A notable blockchain system is BigChainDB [39] that facilitates NoSQL document-based database capabilities for fast queries with blockchain systems. The tamper-resistance is

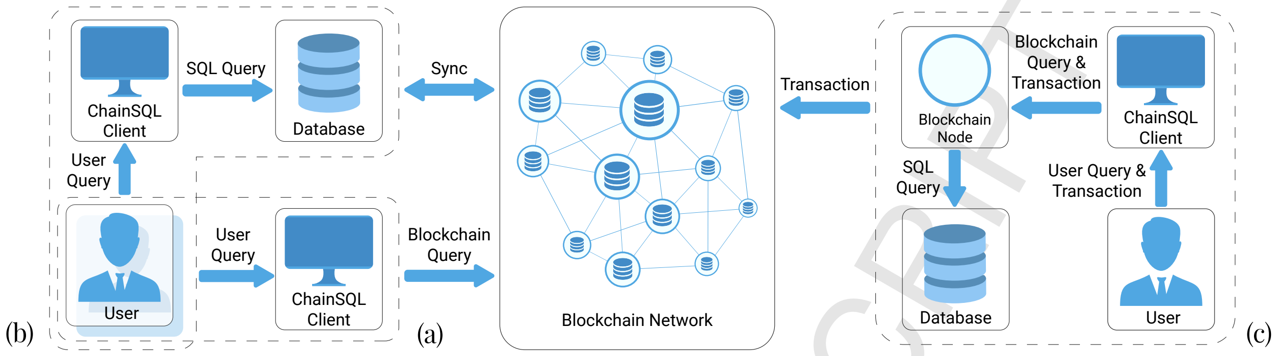


Figure 5: An overview of the ChainSQL access mechanism; the blockchain network is accessed (a) directly by the client for write operations, (b) by using an underlying database for read operations, or (c) by configuring a database on a blockchain node for read-database or write-blockchain operations.

achieved by way of shared replication, reversion of disallowed updates or deletes, and cryptographic signing of all transactions. However, one of the limitations of BigChainDB is the support for MongoDB only and no support for SQL databases. The consensus algorithm in BigChainDB is based on Paxos [40] that does not guarantee Byzantine fault tolerance. One of the issues in BigChainDB is the absence of a unified language for queries to provide a database-agnostic interaction which limits the database choice.

Another system (currently in development) is Catena[3] that replaces transactions in a blockchain system with a limited set of SQL expressions. However, the applicability of Catena in business applications is limited due to the choice of the consensus protocol which is based on PoW and is prone to system forks and low throughput.

In summary, blockchain is an interesting technology and many promising initiatives have been taken recently. Still, it is just a beginning and a lot of research effort is required on theory and systems.

3. ChainSQL: A Blockchain Database Application Platform

In this section, we give an overview of ChainSQL, a blockchain-based log database system that has the integrity of the blockchain and the fast query processing of the distributed databases, as the context for understanding the system that we present in Section 5.

One of the initial design considerations in a blockchain systems is the choice of the underlying blockchain network. The choice is based on the intended application scenario. The enterprise case for ChainSQL is to process high volume business transactions in geo-separated financial institutions; therefore, *Ripple* is a suitable choice due to high throughput and fast validation process.

We now present key design aspects in ChainSQL.

[3]https://github.com/pixelspark/catena

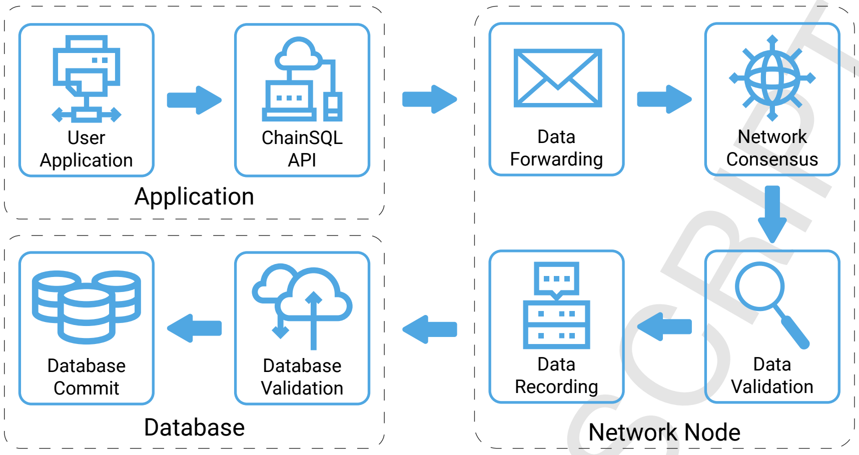


Figure 6: The architecture of ChainSQL

3.1. User Management

We first discuss user management in ChainSQL. The cryptographic schemes supported for user authorisation are both symmetric and asymmetric. A user is identified by a unique public key and the tables owned by the user and the transactions initiated by the user are signed with the user public key. The access to user table is regulated using *grant permission*. ChainSQL supports customisable table access, i.e. *grant permission* to a subset of records, attributes, or both. A user-based access mechanism defined in the master table, *TableList*, as a triple (*user*, *table*, *operations*), enables transaction validation. The master table also stores the indexes of the recent transactions and the associated datastore entries for fast synchronization and validation. The peers are able to quickly synchronize the unsynchronized parts using the master table.

The *user* in ChainSQL operates as one of the following: (i) as a complete-node that participates in the consensus process and thus is responsible for the network integrity, (ii) as a partial-node or a listen-in that is interested in transactions related to itself only, and (iii) as a light-weight client connected to a complete-node for network operations. An overview of the ChainSQL user access mechanism is shown in Figure 5.

3.2. ChainSQL *Architecture*

We now present the building blocks of ChainSQL for which an overview is given in Figure 6 and the components are detailed below:

Application Interface. The access to the blockchain network is by way of an API. In the user context, the instruction set for the network access commands is similar to usual database operations and is therefore easy to use. The support for multiple programming languages through APIs[4] enhances the flexibility and applicability of ChainSQL. The user commands, in SQL or JSON format, are signed and wrapped-up into network transaction format by the

[4]https://github.com/ChainSQL

API and if a transaction is authenticated, it is forwarded to the blockchain network for consensus.

Network Consensus. To maintain the integrity of the data, network consensus is of fundamental importance. In ChainSQL, a transaction is authenticated as follows: a subset of the network nodes selected by the implementation of a UNL scheme validate a transaction. If consensus is not achieved on a transaction, the operation is rolled back and the transaction is not added to the blockchain. The validation process is completed within a few seconds and the user is updated promptly about the transaction status.

Database Operations. The blockchain network transmits the transaction data to the corresponding database for processing. The database operations are performed in near real-time and the validating and the backup nodes maintain the complete database record along with user tables. A complete-node stores all the transactions on the network whereas a partial-node stores only related transactions and headers.

Permission and Grant system. The user authorization in ChainSQL is by way of public and private keys. Default read and write permissions on a table rest with the owner of the table however, the owner can *grant* both read and write permissions to other users. The database tables and the (*user*, *table*, *access*) relationship triples are stored in the master table, 'TableList'. Table synchronization is one of the following: default, auto or none. The operations on a table are customized by the owner and the owner may choose among operations, for instance, select, insert, update and delete. A check on the number of user operations on a table can also be placed by the owner.

3.3. ChainSQL *API*

We now discuss the operations supported by the ChainSQL API. The API incorporates rich semantics for multiple languages including C++ and Java, and supports queries in SQL and JSON with special predefined format. The API support for Java and Javascript is by way of remote procedure calls employing web socket interactions. The user queries, in SQL or JSON format, are similar to a server-client communication environment and are transformable one way or the other, conveniently. A sample query in both SQL and JSON formats is shown in Figure 7.

The ChainSQL API supports (i) user commands, for example, create, update, delete and insert, and (ii) financial commands such as create offer, update offer, make payment, and others. When a user queries for information on the network, a 'valid' peer in possession of read-only transaction data answers the query. As mentioned already, the queries can also be performed on the local synchronized database of the user.

The database operations are of three types, namely *sqlStatement*, *tableListSet* and *sqlTransaction* where *sqlStatement* is used to insert, update or delete records; *tableListSet* is used to create or mutate user-tables; and *sqlTransaction* is used to perform a set of operations atomically, i.e. as a single transaction. Data synchronization is controlled by the user. A node may choose between synchronize-at-each-transaction and synchronize-at-each-interval options. For a synchronize-at-each-transaction option, write operations are required

```
{
  "method": "r_insert",
  "params": [
    {
      "offline": false,
      "secret": "xnoPBzXtMeMyMHUVTgbuqAfg1SUTb",
      "tx_json": {
        "TransactionType": "SQLStatement",
        "Account": "zHb9CJAWyB4zj91VRWn96DkukG4bwdtyTH",
        "Owner":  "zHb9CJAWyB4zj91VRWn96DkukG4bwdtyTH",
        "Tables": [
          {
            "Table": {
              "TableName": "peersafe"}
        }],
        "Raw": [{ "age": 20, "name": "B"}]
        }} ]}
```

```
INSERT INTO peersafe (age, name)
VALUES (20, "B");
```

Figure 7: A sample query in SQL (below) and JSON (above) format using the ChainSQL API.

to be confirmed prior to subsequent write operations. A system table, *SyncTableState*, defines the data synchronization mode.

Similarly, data privacy is also specified by the user and a table can be created by the user with encryption flag 'ON' or 'OFF'. The support for both symmetric and asymmetric encryption is available. A user that wishes to access the data fetches the encrypted data and also requests the access to the data. If the access is granted, the data can be accessed accordingly using the public/private key combinations. Thus, data privacy is ensured and data availability is also facilitated.

Another interesting feature is the event subscription for the 'data change' event for an instance of data using the *subscribeTable* command. Data audit is performed (i) locally, if user maintains a local database, and (ii) remotely on a peer, if a local database is not maintained. In this case, user listens to each new datastore update and proof of validation shared on the network. The datastore is queried for related transactions, i.e. the set of transactions that modify the user data and have a *grant* approval for audit.

3.4. Data validation and forwarding

A transaction is cryptographically signed by a user, is packaged in appropriate query format and is forwarded to the consensus module. ChainSQL peer validates the command signature that uniquely identifies a user. As mentioned already, for the transaction data, the encryption can be turned 'ON' or 'OFF' by the user. If data encryption is turned 'ON', only the data owner or the authenticated user can access the data.

The validation process is based on pre-defined business rules. The business rules are implemented primarily to prevent system forks. Therefore, if two conflicting transactions are included in the datastore, only one transaction is validated in the datastore at most. The default validation rules are built-in ChainSQL and are customisable as per business needs. The validation is performed in three steps. Firstly, user permissions to perform the transaction operations are validated. Secondly, the transaction fields and the correctness of the transaction is validated. Thirdly, the consequence of the transaction execution is validated on the local database.

In order to prevent replication and double-spending attacks, user provides a nonce number with each transaction. If two transaction with the same nonce number and hash are submitted to the datastore, this is considered as an inconsistency and at most one of the two is applied to the datastore.

4. ChainSQL Consensus

The consensus is at the core of a blockchain system and ensures the data integrity in the network. The choice of a consensus protocol also determines the system throughput. Hence, it is important to implement an appropriate consensus protocol. As the intended business case for ChainSQL is enterprise financial transactions, the consensus protocol is based on Ripple that satisfies following three constraints:

(1) *Correctness:* a non-valid transaction is confirmed during consensus if the number of Byzantine faulty nodes are below the tolerance threshold.
(2) *Convergence:* a valid node decides on a transaction in finite time.
(3) *Agreement:* all valid nodes eventually agree on exactly the same datastore state.

We first discuss some concepts related to the consensus protocol and then present the consensus algorithm.

4.1. Consensus Dynamics

ChainSQL consensus is characterised by high throughput and fast validation time. The output of a consensus round is an updated datastore state for which the network nodes take different roles. We give some details below:

Consensus Objective. The objective of the consensus is to attain an agreed upon datastore state. Formally, the objective of the consensus is to achieve the agreement among the validation nodes with a known probability of Byzantine failures. Consensus must prevent situations where two disjoint sets of nodes agree on different states of the datastore. Therefore, a transaction is accepted or otherwise subject to majority support, or the decision is postponed for the upcoming rounds. Thus, the outcome of a consensus round is a set of transactions for which the consensus is achieved. In each consensus round, all the validated and confirmed transactions are applied to the datastore.

Datastore State Transition. At each consensus stage, the current state of the datastore is agreed upon as 'valid'. The datastore state is identified with a sequence number which increments over time. The sequence number contains the metadata about the transactions. Each transaction has a code that indicates the changes to the datastore as a result of transaction execution. Nodes in the blockchain network communicate with each other on a subset of transactions labelled as 'proposal set'. As for a transaction to be accepted, it has to be known to the majority of the nodes in the network, the nodes communicate with each other to prepare a proposal set that is agreed upon by the majority of the nodes to attain the consensus.

Consensus Round. Each node receives and validates transactions only from a subset of nodes considered as 'trusted', namely unique node list or UNL. Trust is a synonym for a verified signature, i.e. network messages only from these servers are validated and processed. This approach filters all external spam transactions and speeds up the consensus process. A transaction is sent by a client to a subset of UNL broadcast nodes which broadcast the information to all UNL validation nodes. Next, the transaction is included in the proposal set and is considered for validation. The transactions that are not considered in the upcoming proposal set are considered for future rounds until the transaction lifetime expires. The transaction lifetime is based on the last datastore set index. Note that broadcast nodes dispatch user transactions and respond to queries whereas validation nodes ensure the integrity of the system by participating in the consensus process. Validation nodes serve as backup nodes in the disaster recovery centre.

4.2. Consensus Algorithm

ChainSQL consensus is a multi-phase process where the first phase is transaction exchange, the second phase is initial proposal exchange and the third phase is the consensus on the proposal set. The pseudo-code for the consensus round is presented in Algorithm 1 and the consensus utility routines are given in Algorithm 2.

During the first phase, the transactions in the current round X and undecided transactions R_{j-1} from the previous round are combined together to prepare a 'proposal set'. All the validation nodes broadcast undecided transactions to the nodes in the UNL. The transactions in the proposal set need to be validated before being added to the datastore. Thus, the transactions are validated and then applied to the new local datastore L_o. After delay t_1, the second phase commences and validation nodes exchange proposal set. Each validation node creates proposal set p from transactions in L_o and then p is sent to the nodes $[s_1...s_n]$ in UNL.

When a node receives p from a node in UNL, the transactions in p are compared to the transactions in L_o and for the transaction match, the match vote is updated. If a transaction x is included in p from server s, vote from server s for the transaction x is updated in the datastore.

The necessary and sufficient condition for consensus on a proposal set is a 'majority vote'. If a majority vote is not achieved, peers exchange new proposals with transactions having number of votes $\geq \tau * |\mathrm{S}|$. The acceptance threshold τ is incrementally increased

Algorithm 1 An outline of the ChainSQL consensus algorithm

```
function ConsensusRound_j($R_{j-1}$: undecided transaction set; $S = [s_1 \dots s_n]$;
        $\tau$: acceptance threshold; $\delta$: acceptance threshold step)
    $L_o \leftarrow \emptyset$                                    ▷ $L_o$: New open ledger
    $L_c \leftarrow \emptyset$                                    ▷ $L_c$: Closed ledger
    $L_o \leftarrow \{all\ x \in X = R_{j-1}\}$        ▷ Add undecided transactions from previous round
    for all transaction $x \in X$ do
        ReceiveTransaction($x$, Sender($x$), $O$)
    Delay($t_1$)
    $p \leftarrow L_o$
    Send initial proposal $p$ to servers $[s_1 \dots s_n] \in S$
    ReceiveProposal($L_o$, $p$, $s_i$) from servers $[s_1 \dots s_n] \in S$
    $L_c \leftarrow$ ConsensusAchieved($L_o$)
    if $L_c = \emptyset$ then
        repeat
            $p \leftarrow \{\forall x \in L_o \mid |x.\text{votes}| \geq (\tau) * |\text{S}|\}$
            Send $p$ to $[s_1 \dots s_n] \in S$
            ReceiveProposal($L_o$, $p_i$, $s_i$) from servers $[s_1 \dots s_n] \in S$
            $L_c \leftarrow$ ConsensusAchieved($L_o$)
            $\tau \leftarrow \tau + \delta$
        until $L_c \neq \emptyset$
    apply($L_c$)
    $R_j \leftarrow L_c \cap L_o$                      ▷ $R_j$: remaining transactions for next round
    return $R_j$
```

after each round to achieve consensus even though there is a disagreement in an earlier round. In practice, the threshold values are set as, $\tau = 0.5$, $\delta = 0.1$, to achieve fast convergence. The acceptance threshold for a transaction is initially set to 51% agreement. If participants disagree on the proposal set, the threshold is increased gradually until no disputed transactions are left.

When consensus is achieved on the datastore, it is applied to the database for the updates to take affect. Undecided transactions in the previous round are forwarded to the subsequent round. Validation nodes publish a cryptographic proof of the datastore update. If consensus is achieved, each validation node has the same proof. The participants listen to the validation proof and have one of the following possibilities:

(1) If the validation state L_c has the same state as the local datastore, consensus is achieved.
(2) The validation nodes have disagreement on the datastore state due to state inconsistency. The validation nodes synchronize the datastore state.
(3) If majority vote is not achieved, then the round is re-started. The consequence is a loss in time, typically a few seconds. This happens when validation nodes do not have

Algorithm 2 ChainSQL consensus routines

```
function ReceiveTransaction($x$: Transaction, $s$: Sender, $L_o$: Open Ledger)
    if $s \in S \wedge$ validate($x$) then
        $L_o \leftarrow$ apply($x$)
    return $L_o$

function ReceiveProposal($L_o$: Open Ledger, $p$: Proposal, $s$: Sender)
    if $s \in S$ then
        for all $x \in p$ do
            $L_o[x]$.votes $\leftarrow L_o[x]$.votes $\cup s$
    return $L_o$

function ConsensusAchieved($L_o$: Open Ledger, $\theta$: vote threshold)
    $l \leftarrow \emptyset$
    for all $x \in L_o$ do
        if $|x.\text{votes}| \geq \theta * |S|$ then
            $l \leftarrow l \cup x$
    return $l$
```

a consensus, but are to proceed to the next round.

Once the consensus is achieved on the proposal set, the proposal set is written to the datastore to get the updated datastore state.

5. ChainSQL Usecases

In this section, we present three ChainSQL usecases: (i) a multi-active database middleware for connecting the user application with the underlying database; (ii) a disaster recovery middleware that connects user application production nodes with the disaster recovery nodes; (iii) an audit middleware that is connected with the production centre to process transaction traces.

5.1. Multi-active Database Middleware

Multi-active database is a middleware that connects the enterprise application with the underlying database. The underlying database is either a traditional relational database or a NoSQL database. All the data definition and data manipulation operations for the database are recorded in an operation log that is maintained using the blockchain technology and is immutable, i.e. it cannot be modified or deleted.

User application calls ChainSQL API to send the transaction to a network node. Once the consensus is achieved on the transaction, each validation peer has exactly the same datastore state. The nodes are configured with the database to synchronise all incoming database operations. As shown in Figure 8, a user can either directly query the blockchain

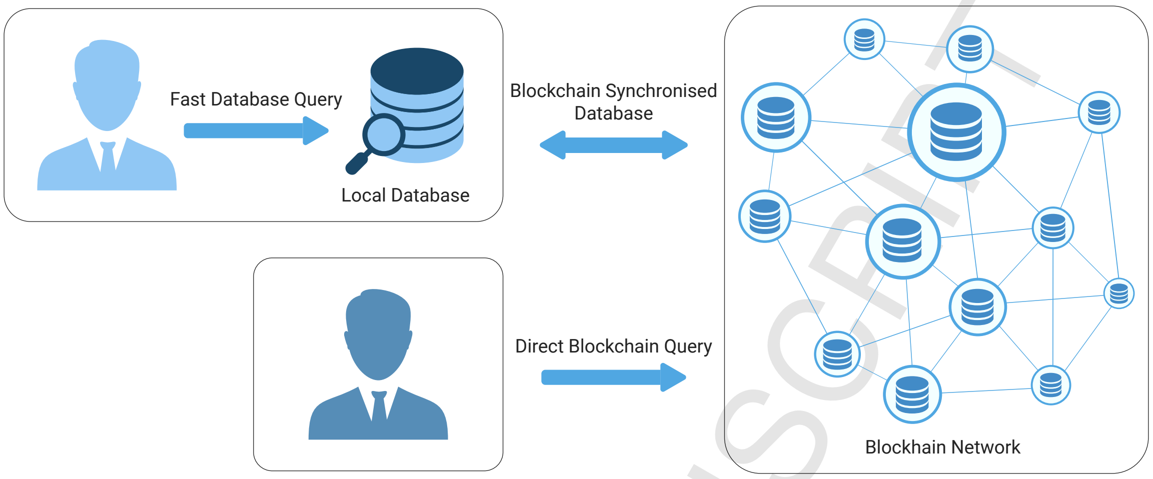


Figure 8: ChainSQL as a multi-active database.

or keeps a synchronised local database for fast queries. If the user keeps a local database, then one of the network nodes serves as a backup node. The write to the blockchain is by way of ChainSQL consensus protocol.

In case of a node failure, the user switches to another node on the network seamlessly. This ensures zero-recovery-time and a multi-active database deployment in real-time. The fault node is restored from the most recent checkpoint during the recovery process. An example database failure is shown in Figure 9. Since local database of the user is not available, each user query is forwarded to one of the nodes. If a node does not respond, query is sent to another node in the network.

One of the concerns in a multi-active database is the security of data when it is being transmitted across the network. The middleware provides both symmetric and asymmetric encryption schemes from which the user can choose the appropriate security mechanism for the data. The supported encryption schemes include $secp256k1$ and elliptic-curve signatures $ed25519$. Another important aspect of the multi-active middleware is the expandability of the blockchain network nodes. A new node can automatically get the log from an existing node in the network and can replay that log to generate its own version of the datastore which is the same as other network nodes. Once it is established that the new node has the same datastore state as the other network nodes, it can also participate in consensus build-up and synchronous data writing.

5.2. Multi-disaster Recovery Middleware

Multi-disaster Recovery is also a middleware that connects the database production nodes of the enterprise application with the disaster recovery nodes. The architecture of the multi-disaster recovery middleware is shown in Figure 10. As mentioned earlier, the user operations are recorded as log files, e.g. Binlog, Redo Log and so forth, in the production centre and are analysed prior to a blockchain transaction generation. During disaster recov-

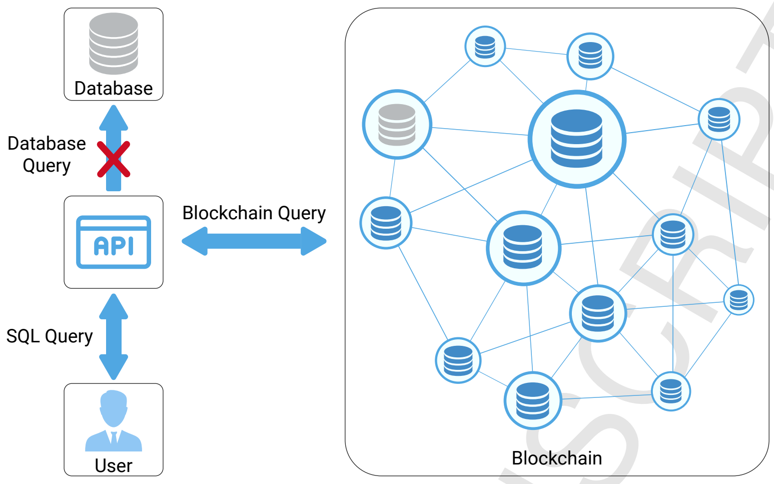


Figure 9: ChainSQL crash recovery with a local database failure

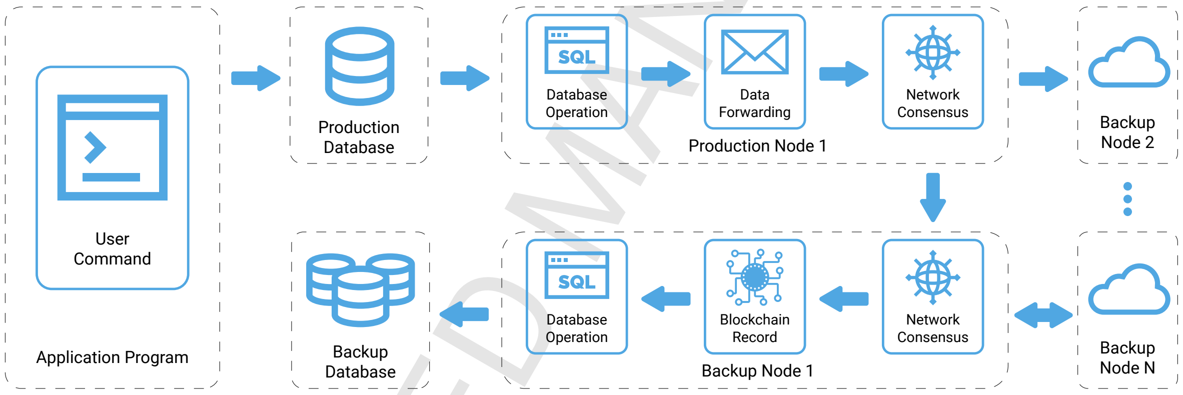


Figure 10: The architecture of the database disaster recovery middleware.

ery, the first step is to achieve the consensus for the blockchain network that must include the backup node such that the backup node has exactly the same data as every node on the blockchain network. When a new block is generated, the backup node reads the block and sends it to the disaster recovery centre. The recovery centre performs the database backup using the transaction data. Thus, if a node fails at the production centre, the users switch to the recovery centre to complete the task. This is achieved by elevating a backup node to the status of a production node.

The data of the production centre is transmitted to the disaster recovery centre immediately and the log is re-executed to achieve the recovery.

Business Application. An important application scenario where ChainSQL has been used is in a financial environment. The business requires the core business data to be protected and the most recent data to be available across the whole business process. The encryption based tamper-resistance feature of the ChainSQL along with the multi-active database

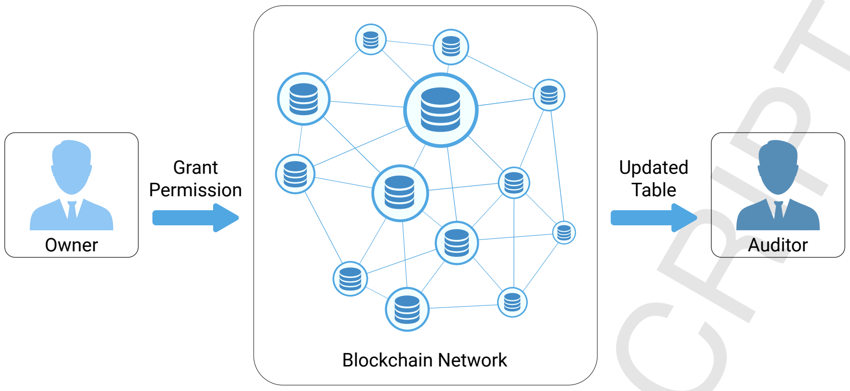


Figure 11: ChainSQL for audit

ensures security of the customer data and continuous business operations. In case of a node failure, the data-level disaster recovery backup system activates seamlessly, thus ensuring the continuity of the business operations.

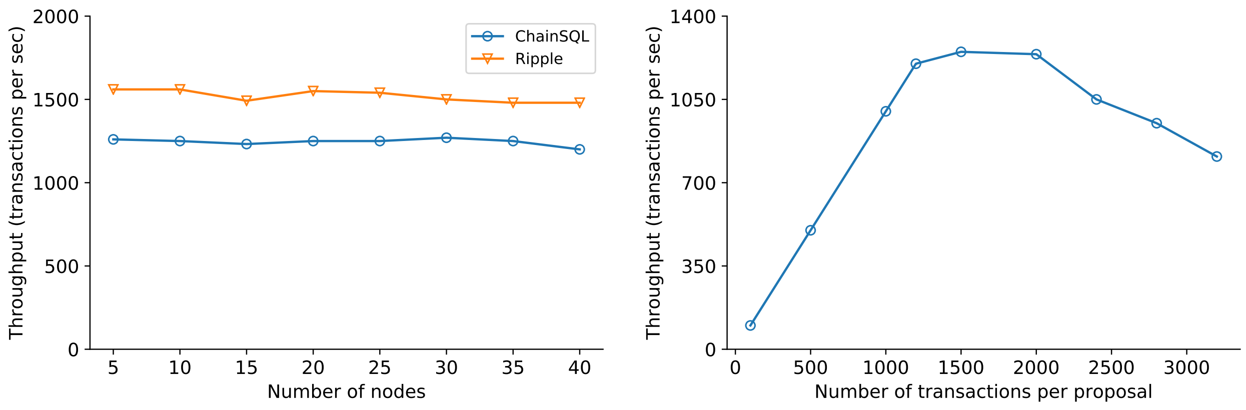
5.3. Audibility

The immutability of a blockchain brings audibility as an inherent feature to the blockchain systems. In a typical scenario, user A performs a *grant* operation for users B and C on table T. Grant operation is recorded as a blockchain transaction and the transaction with *grant* access is signed and sent to concerned peers. When the *grant* operation transaction is finalized and is included in the datastore, A updates account access information.

User B requests a select query on table T and sends it to ChainSQL node P which is a complete node. Node P fetches the identity of user B from the signature of the query. Since user B has permissions to perform select query on table T, node P executes the query and returns the result to B. A typical transaction pipeline for data audit is shown in Figure 11.

User C has permissions to make read-only queries on table T. User C maintains a local database which is synchronised with the blockchain network and listens to the relevant updates. Each time a new datastore state is committed by the nodes, a proof of validation is multicast to the network. Each time a new datastore state is committed, user C requests node P an update. In the request, user C specifies the most recent closed datastore index. Node P creates an update message and sends the datastore headers starting from the most recent closed records of user C. Besides headers, node P sends an order of transactions that should be applied to the datastore. These transactions are related only to table T. Thus, ChainSQL allows a trustless client-server interaction, where user can maintain a database with related data only without the overhead to store the entire blockchain.

Business Application. ChainSQL combines features of data replication and immutability with flexible access control. As blockchains' inherent immutability prevents ex-post-facto changes, auditor maintains a local database and requests access from a specific user. User grants permission and the auditor fetches all user data from ChainSQL network nodes.



(a) Throughput vs. Transaction per proposal (b) Throughput vs. Number of nodes

Figure 12: ChainSQL throughput

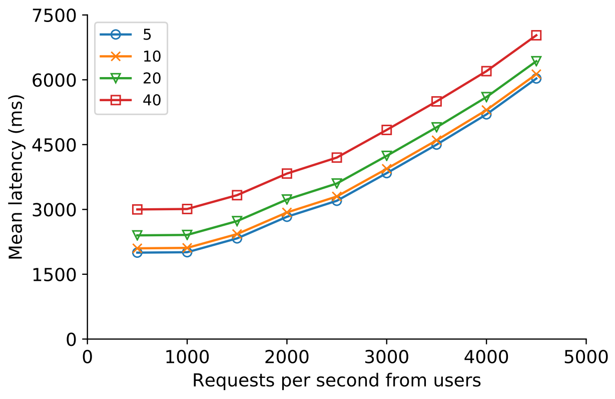
Auditor can perform queries on the local database and can report results to the blockchain network.

6. Empirical Evaluation

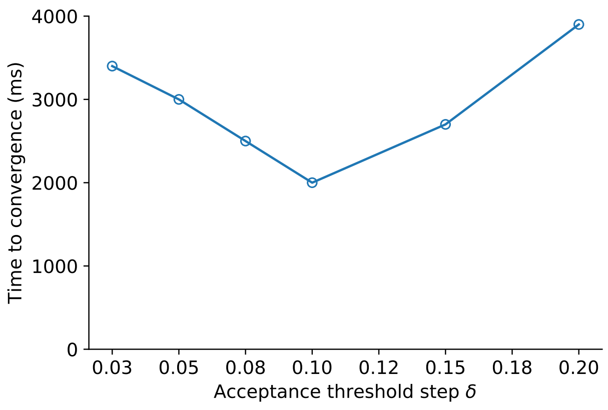
We evaluate the performance of ChainSQL with the help of experiments. We consider parameters such as latency, throughput, scalability, and convergence and perform extensive experiments to demonstrate the effectiveness of the system. We first describe experimental setup.

Setup. We perform the experiments on Amazon EC2 cloud and consider virtual machine instances upto 40. We do not consider the number of nodes beyond 40, as we observe a stable behaviour for ChainSQL in comparison with Ripple. The experiments are executed on c5.large instances. The system directly uses $secp256k1$ signature code from Ripple. Alternatively, a ecliptic-curve signatures $ed25519$ can also be used. A predefined transaction pool is considered for system evaluation and the transactions are fetched at random for each user from the pool. Each transaction updates the datastore state and is recorded in the blockchain. The results are generated as end-to-end measurements from the users' perspective. For the latency and throughput measurements, we employ a test application that sends transactions to the system and populates counts based on the transaction confirmation time. Unless specified otherwise, we set the number of nodes to 20, transactions per proposal to 1500, $\delta = 0.1$, and requests per second to 2000. Each set of experiments is executed 10 times and the averages are reported.

System Throughput and Scalability. We first measure the throughput and scalability of the system. We report system throughput, i.e. number of transactions per second, as we increase the number of nodes from 5 to 40. In another set of experiments, we increase number of transactions per proposal and report the system throughput. In the first set of experiments, Figure 12(a), it can be observed that ChainSQL achieves throughput comparable to Ripple,



(a) Mean latency vs. Number of user requests



(b) Convergence time vs. acceptance step δ

Figure 13: ChainSQL latency

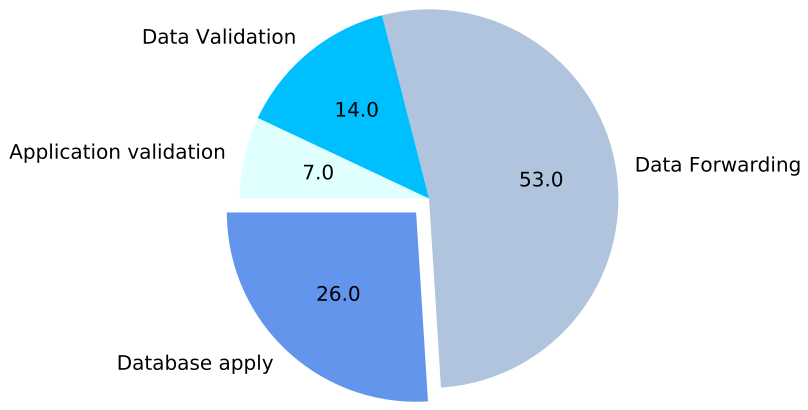


Figure 14: System time distribution

i.e. 1200 transactions per second. Also, a slight decrease in throughput is witnessed with the increase in network size, both for ChainSQL and Ripple. The overall behaviour of ChainSQL is consistent with that of Ripple with little performance downgrade. Note that unlike Ripple, ChainSQL does not only supports financial transactions and is more flexible. In another set of experiments, Figure 12(b), we study the effect of proposal size on system throughput. For the experiments, we considers 20 nodes in the network, and increase proposal size gradually to observe system throughput. It can be seen that the throughput of the system increases with an increase in the proposal size and reaches the peak value at 1500 transactions per proposal. For more than 1500 transactions per proposal, the system throughput deteriorates. For the rest of the experiments, we set the transactions per proposal to 1500.

Transaction Latency. The latency is the delay in transaction confirmation, i.e. the difference in time between transaction submission and confirmation. We report mean latency in milliseconds as we increase requests per second from 1000 to 5000, and number of nodes from 5 to 40. The proposal size is set to 1500. The results, Figure 13(a), show that the latency increases linearly with the number of requests per second. Similarly, when the number of nodes in the network increase, mean latency also increases due to the increased network communication. However, the overall behaviour of the system remains stable, e.g. average latency for 5 nodes is 2000 ms, and increases to 3500 ms for 40 nodes.

Convergence. An interesting parameter, Acceptance threshold δ, determines the convergence rate for the system. We set the number of nodes to 20, proposal size to 1500, requests per second to 2000, and study the convergence behaviour for changing δ. It can be seen, Figure 13(b), that the least time to convergence is achieved for $\delta = 0.1$. For values, smaller or higher than 0.1, the convergence time increases.

Transactions Time-slice. We also compute statistics on the transaction execution stages to get an insight into the transaction execution. We set the number of nodes to 20, proposal size to 1500, $\delta = 0.1$, and compute the time-slice for each transaction execution stage. It can be seen, Figure 14, that network delay is the most time consuming phase in transaction execution.
Remark. ChainSQL is a blockchain system that also supports fast queries by way of integrating database with the blockchain. The performance of the system for the DDL queries, i.e. create, alter, and drop, and DML queries (except select), i.e. insert, update, and delete, is similar to the blockchain network Ripple. Note that the blockchain based DDL and DML queries are slow due to the requirement of the consensus. However, as the select queries do not change the datastore state, they are executed fast by integrating a database with the blockchain at the node level.
In summary, the performance of ChainSQL is comparable (i) to Ripple for the queries that change the datastore state and (ii) to database for queries that do not change the datastore state.

7. Conclusion and Future Work

In this paper, we presented ChainSQL and its novel applications through three usecases that are implemented as a middleware between the user application and the database. The first usecase is a tamper-resistant multi-active database, the second usecase is a data-level disaster recovery backup and the third is an audit middleware. The effectiveness of the system is demonstrated with the help of a detailed experimental study. ChainSQL is the first system of its kind that features the tamper-resistance of the blockchain and the fast query processing of the distributed databases. The utility of the ChainSQL is evident from its business usecases in domains including finance and supplychain, therefore, it offers promising application scenarios for future. A number of considerations from system implementation point of view are also in the pipeline. For instance, the support for big data analytics in ChainSQL, and the implementation of complex indexes for query optimisation.

Acknowledgement

The work was partially supported by the CAS Pioneer Hundred Talents Program [grant number Y84402, 2017], SIAT-Peersafe IoT Security Lab supported by PeerSafe and PeerCome in Beijing and Shenzhen, respectively [grant number Y7Z0181001, 2017], and CAS President's International Fellowship Initiative [grant number 2018VTB0005, 2018]. The authors would also like to acknowledge the application development contributions made by Xiaoming Lu, and Chengcheng Ji.

References

[1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf (2009).
[2] G. Wood, Ethereum: A Secure Decentralised Generalised Transaction Ledger, http://gavwood.com/paper.pdf (2015).
[3] D. Schwartz, N. Youngs, A. Britto, The ripple protocol consensus algorithmOnline; accessed: 08-Jan-2018.
[4] D. D. Rosa, Blockchain programming, `http://davidederosa.com/basic-blockchain-programming/`, online; accessed: 09-Jan-2018 (2015).
[5] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, J. Wang, Untangling blockchain: A data processing view of blockchain systems, CoRR abs/1708.05665. arXiv:1708.05665.
[6] B. Nasrulin, M. Muzammal, Q. Qu, ChainMOB: Mobility analytics on blockchain, in: 19th IEEE International Conference on Mobile Data Management, MDM 2018, Aalborg, Denmark, June 25-28, 2018, 2018, pp. 292–293.
[7] S. Suzuki, J. Murai, Blockchain as an audit-able communication channel, in: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Vol. 2, 2017, pp. 516–522. doi:10.1109/COMPSAC.2017.72.
[8] A. Wright, P. D. Filippi, Decentralized blockchain technology and the rise of lex cryptographia, Available at: `https://ssrn.com/abstract=2580664` (Mar 2015).
[9] C. P. Chen, C.-Y. Zhang, Data-intensive applications, challenges, techniques and technologies: A survey on big data, Information Sciences 275 (Supplement C) (2014) 314 – 347. doi:https://doi.org/10.1016/j.ins.2014.01.015.
[10] F. B. Schneider, Implementing fault-tolerant services using the state machine approach: A tutorial, ACM Computing Surveys (CSUR) 22 (4) (1990) 299–319.
[11] M. Szydlo, Merkle tree traversal in log space and time, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 541–554.
[12] S. Popov, O. Saa, P. Finardi, Equilibria in the Tangle, arXiv.org (2017) arXiv:1712.05385arXiv:1712.05385.
[13] Y. Sompolinsky, Y. Lewenberg, A. Zohar, SPECTRE - A Fast and Scalable Cryptocurrency Protocol., IACR Cryptology ePrint Archive, 2016.
[14] Q. Qu, S. Liu, F. Zhu, C. S. Jensen, Efficient online summarization of large-scale dynamic networks, IEEE Trans. Knowl. Data Eng. 28 (12) (2016) 3231–3245.
[15] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, G. Danezis, Consensus in the age of blockchains, CoRR abs/1711.03936. arXiv:1711.03936.
[16] M. J. Fischer, N. A. Lynch, M. S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM (JACM) 32 (2) (1985) 374–382.
[17] C. Dwork, M. Naor, Pricing via processing or combatting junk mail, in: Annual International Cryptology Conference, Springer, 1992, pp. 139–147.
[18] A. Kiayias, A. Russell, B. David, R. Oliynykov, Ouroboros: A provably secure proof-of-stake blockchain protocol, in: Annual International Cryptology Conference, Springer, 2017, pp. 357–388.
[19] M. Castro, B. Liskov, et al., Practical byzantine fault tolerance, in: OSDI, Vol. 99, 1999, pp. 173–186.

[20] C. Cachin, Architecture of the hyperledger blockchain fabric, in: Workshop on Distributed Cryptocurrencies and Consensus Ledgers, 2016.
[21] M. Risius, K. Spohrer, A blockchain research framework - what we (don't) know, where we go from here, and how we will get there, Business & Information Systems Engineering 59 (6) (2017) 385–409. doi:10.1007/s12599-017-0506-0.
[22] I. Sergey, A. Hobor, A Concurrent Perspective on Smart Contracts., Financial Cryptography Workshops 10323 (3) (2017) 478–493.
[23] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, G. Danezis, Chainspace: A Sharded Smart Contracts Platform, arXiv.org (2017) arXiv:1708.03778arXiv:1708.03778.
[24] P. L. Seijas, S. J. Thompson, D. McAdams, Scripting smart contracts for distributed ledger technology., IACR Cryptology ePrint Archive.
[25] I. Sergey, A. Kumar, A. Hobor, Scilla - a Smart Contract Intermediate-Level LAnguage., CoRR.
[26] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, T. Moore, Game-theoretic analysis of ddos attacks against bitcoin mining pools, in: International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 72–86.
[27] D. Schwartz, N. Youngs, A. Britto, The ripple protocol consensus algorithm (2014), Ripple Labs.
[28] R. Pass, L. Seeman, A. Shelat, Analysis of the blockchain protocol in asynchronous networks, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2017, pp. 643–673.
[29] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.
[30] D. Johnson, A. Menezes, S. Vanstone, The elliptic curve digital signature algorithm (ecdsa), International journal of information security 1 (1) (2001) 36–63.
[31] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, E. W. Felten, Mixcoin: Anonymity for bitcoin with accountable mixes, in: International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 486–504.
[32] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized anonymous payments from bitcoin, in: Security and Privacy (SP), 2014 IEEE Symposium on, IEEE, 2014, pp. 459–474.
[33] I. Lin, T. Liao, A survey of blockchain security issues and challenges, I. J. Network Security 19 (5) (2017) 653–659.
[34] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, E. Zenner, Ripple - Overview and Outlook., TRUST 9229 (Chapter 10) (2015) 163–180.
[35] E. Androulaki, et al., Hyperledger fabric: A distributed operating system for permissioned blockchains, CoRR abs/1801.10228. arXiv:1801.10228.
[36] M. Vukolic, Eventually returning to strong consistency, IEEE Data Eng. Bull. 39 (1) (2016) 39–44.
[37] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, V. Sassone, Blockchain-based database to ensure data integrity in cloud computing environments.
[38] M. Ali, J. C. Nelson, R. Shea, M. J. Freedman, Blockstack: A global naming and storage system secured by blockchains., in: USENIX Annual Technical Conference, 2016, pp. 181–194.
[39] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, A. Granzotto, Bigchaindb: a scalable blockchain database, BigChainDB, https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf.
[40] T. D. Chandra, R. Griesemer, J. Redstone, Paxos made live: an engineering perspective, in: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, ACM, 2007, pp. 398–407.

MUHAMMAD MUZAMMAL received the Ph.D. degree in computer science from the University of Leicester, U.K. He is currently with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, China. His research interests are in data mining, uncertain data, and blockchain systems.

QIANG QU is an associate professor at Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences (CAS), and the director of Guangdong Provincial R&D Center of Blockchain and Distributed IoT Security. He is a candidate of the CAS Poineer Hundred Talents Program. He received the MSc degree in computer science from Peking University and the PhD degree from Aarhus University. His current research interests are in data-intensive applications and systems, focusing on efficient and scalable algorithm design, blockchain, data sense-making, and mobility intelligence. His recent research has been published in leading journals and international conferences, including ACM SIGMOD, VLDB, AAAI, the IEEE transactions on Data Engineering, the IEEE Transactions on Intelligent Transportation Systems, and Information Sciences. He was TPC member of several prestige conferences, and he chaired workshops in VLDB 2018, VLDB 2017, ICDM 2015, and APWEB-WAIM 2017 on mobility analysis.

BULAT NASRULIN received his BS in Applied Mathematics from Kazan Federal University and MS in Data Science from Innopolis University in 2015 and 2017, respectively. He is currently pursuing the PhD degree in Shenzhen College of Advanced Technology, University of Chinese Academy of Sciences. His research interests include blockchain, distributed computing, and smart contracts.

Highlights

The highlights of this work are as follows:

- We present, ChainSQL, a blockchain database application platform that has the decentralised, distributed and audibility features of the blockchain and quick query processing and well-designed data structure of the distributed databases
- ChainSQL features a tamper-resistant and consistent multi-active database, a reliable and cost effective data-level disaster recovery backup and an auditable transaction log mechanism.
- ChainSQL solves the data integrity and reliability issues of the distributed databases by using the blockchain technology and still keeps the fast query processing of the distributed databases.
- The incorporation of the security and audibility features of the blockchain into the distributed database systems by the implementation of a blockchain network.
- The support for both NoSQL and SQL databases is a desirable feature of the system and enhances applicability of the system.