VQL: Efficient and Verifiable Cloud Query Services for Blockchain Systems

Haotian Wu, Zhe Peng, Songtao Guo, *Senior Member, IEEE*, Yuanyuan Yang, *Fellow, IEEE*, and Bin Xiao, *Senior Member, IEEE*

Abstract—Despite increasingly emerging applications, a primary concern for blockchain to be fully practical is the inefficiency of data query. Direct queries on the blockchain take much time by searching every block, while indirect queries on a blockchain database greatly degrade the authenticity of query results. To conquer the authenticity problem, we propose a Verifiable Query Layer (VQL) that can be deployed in the cloud to provide both efficient and verifiable data query services for blockchain systems. The middleware layer extracts data from the underlying blockchain system and efficiently reorganizes them in databases. To prevent falsified data from being stored in the middleware, a cryptographic fingerprint is calculated based on each constructed database. The database fingerprint will be first verified by miners and then written into the blockchain. Moreover, public users can verify the entire databases or several databases that interest them in the middleware layer. We implement VQL together with the verification schemes and conduct extensive experiments based on a practical blockchain system. The evaluation results demonstrate that VQL can efficiently support various data query services and guarantee the authenticity of query results for blockchain systems.

Index Terms—cloud query service, verifiable query, data authenticity, blockchain systems.

1 Introduction

Cryptocurrencies embodied by Bitcoin and its descendants, acting as a modern form of digital currency, have sparked a surge of innovation in decentralized computing. Blockchain, as the fundamental technology of cryptocurrencies, offers many characteristic advantages including decentralized storage and immutability. Besides payment, the blockchain technique can be used in a far wider area such as smart contract [1], supply chain management [2], healthcare [3], distributed storage [4] and IoT [5]. Current blockchain-based systems have tremendous potential in reducing operating costs, increasing resistance to manipulation, preventing fraud and facilitating execution of contracts.

Though the blockchain technique can bypass data storage fraud using distributed ledger with consensus mechanisms, most current schemes only provide limited query services. In pursuit of excellent writing performance, many blockchain systems adopt the key-value database as the underlying database, e.g., LevelDB for Bitcoin and Go client of Ethereum. However, this kind of databases are usually based on LSM-tree [6], which provides barely satisfactory reading performance due to the complicated processing operations, especially for random reading [7]. In addition to the query inefficiency, the query types that native clients support are also limited. Thus, how to provide versatile queries efficiently for all kinds of applications has not been well solved yet.

One approach to tackling the problem of query limitation in the blockchain system is to maintain several extra structures on the peer node, e.g., Project Toshi [8] and ECBC [9]. Project Toshi saves much more information and indexes besides the native client for richer queries. ECBC builds a tree structure to support efficient query on transactions. In the Bitcoin network, for instance, the raw blockchain data does not contain the balance value of each address. Thus, query service providers can pre-compute and maintain the current balance of each address using the extra list structure so that they can quickly return the result of the balance query without traversing all transaction data. Regretfully, this architecture does not meet the requirement of various queries since the balance list can solely solve the balance query problem. In other words, the extra data structure needs to be customized for the predefined query type. Assume that the Bitcoin node has already supported the query for the address balance. When a user wants to further query about several transaction details related to an address, the peer node still needs to adopt the direct query, searching all blocks in the blockchain for the result. This scheme brings about additional space cost because the node has to maintain an extra and specialized transaction list for each address.

Another method is to take the indirect query by searching the database with high reading performance for blockchain data instead of the original database adopted by the native client. EtherQL [10] integrates the typical database with the Ethereum to expedite the process of data query. Blockchain.com [11] is able to provide the address information since it stores historic transactions in the database in advance. BlockSci [12] incorporates an in-memory database to boost the data query for blockchain analysis. However, these systems assume that the server always returns correct results based on the blockchain data. In fact, the server may return incorrect results that conflict with the true blockchain data due to some interests or security vulnerabilities [13]. In this

- *H. Wu, and B. Xiao are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong.*
  E-mail: {cshtwu, csbxiao}@comp.polyu.edu.hk.
  Corresponding author: Bin Xiao.
- *Z. Peng is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong.*
  E-mail: pengzhe@comp.hkbu.edu.hk.
- *S. Guo is with the College of Computer Science, Chongqing University, Chongqing, P.R. China.*
  E-mail: guosongtao@cqu.edu.cn.
- *Y. Yang is with Department of Computer Engineering and Computer Science at Stony Brook University, New York, USA.*
  E-mail: yuanyuan.yang@stonybrook.edu.

case, a feasible mechanism to verify the data authenticity is highly desired. Therefore, our further research problem will be: *Can we manage to provide efficient and verifiable query services for blockchain systems?*

This problem involves the following challenges that need to be addressed: (1) Supporting versatile query services on blockchain data with high efficiency for different applications. (2) Ensuring the data consistency between the queried data and the underlying blockchain data. (3) Providing a verification scheme for query users to validate partial data on the cloud that interests him.

We give an affirmative answer to the problem in this paper by systematically designing and implementing a Verifiable Query Layer (*VQL*), which is a cloud-based middleware layer providing efficient and verifiable query services for blockchain systems. Superior to the existing designs, our proposed cloud service is capable of meeting the demands of query efficiency and data authenticity simultaneously. A novel framework called vChain [14] achieves the verifiable boolean queries over blockchain data by exploiting an accumulator-based authenticated data structure. However, this work requires radical modification of the existing blockchain systems.

Our system consists of three layers including the underlying blockchain network, the middleware layer and the upper application layer. To cater to various queries from the application layer, the middleware layer will first extract and reorganize the data stored in the underlying blockchain and then store them into the databases. To ensure the validity of the middleware data, each constructed database will generate a fingerprint, which is a cryptographic hash value based on the content and properties of the database (e.g., name, size, timestamp, etc.). This fingerprint will then be verified by miners and further stored in the blockchain. By virtue of the immutability of the blockchain, this verification scheme can prevent any falsified data from being stored by the middleware layer. Public users can also download the entire blockchain data to verify the databases if they do not trust the cloud service. In addition, we provide a simplified query result verification scheme to enable users to just check the validity of the databases that their query involves. Given the abundance and popularity of Ethereum-based applications, in this paper, we employ one of Ethereum testnets to illustrate the feasibility and effectiveness of our proposed system. Our architecture can also be extended to other blockchain applications as VQL can be adapted to any given blockchain system. We conclude our contributions in this paper as follows:

- The *VQL* is a new cloud query service with a three-layer architecture, which efficiently supports various query services in the blockchain system, e.g., from account query to complicated range query, with no need to browse each block in the whole blockchain. The databases in the middleware are dynamically constructed and updated.
- Our proposed cloud service provides a public verification scheme on the constructed databases to ensure its consistency with the underlying blockchain. The fingerprints of databases are verified and stored in the blockchain by the miner. Miners or users with blockchain data can verify the correctness of a database using these fingerprints.
- We utilize the authenticated data structure to manage fingerprints and put forth a simplified query result verification algorithm for users to verify the received result without downloading all blockchain data. Users can issue the verification request about the databases involved and efficiently validate their fingerprints.
- We develop the middleware prototype along with the verification schemes and conduct extensive evaluations based on Ethereum testnet and MongoDB. The results demonstrate that *VQL* can efficiently support various query and verification services for blockchain systems.

The remainder of this paper is organized as follows. We first introduce the preliminary techniques used in our design in Section 2. Then we present the system design and authenticity analysis in Section 3, and show the system implementation and evaluations in Section 4. We review the related works in Section 5 and conclude the paper in Section 6.

2 Preliminary Concepts

In this section, we briefly introduce some techniques related to our system, and clarify their necessity in our system design.

2.1 Blockchain

Blockchain is a distributed ledger that is able to reliably record all transactions in a decentralized network. A typical blockchain usually comprises a series of blocks that are chained in order by referring to the preceding block. A block mainly consists of a block header storing the attributes of the block like the timestamp and the hash value of its predecessor, and a block body, which contains the corresponding list of transaction details in the block [15]. In the blockchain network, each full node keeps a copy of the ledger, the consistency of which is guaranteed by adopting various consensus algorithms. The first implementation of the blockchain-based application is the Bitcoin system [16]. By maintaining a distributed ledger, the Bitcoin system creates a decentralized, open and Byzantine fault-tolerant transaction paradigm, which conforms to the requirements of a new cryptocurrency infrastructure. A blockchain network contains the following features:

Transparency: The network is accessible to all participants. Any participant can get the current state of the blockchain system based on the records in the blockchain.

Consensus: All peer nodes in the network will reach consensus on the blockchain (i.e., no unintentional forks). A valid block discovered by an honest peer will be recorded on the blockchain and accepted by other peers.

Immutable and verifiable: Once a block is discovered and globally accepted, any further modification of this block is impossible. All participants can verify the current state based on the records in the blockchain.

2.2 Merkle Patricia Tree

The Merkle Patricia Tree (MPT) [17] is first introduced in Ethereum [18], which is a cryptographically authenticated data structure combining the Trie Tree and the Merkle tree. MPT can be used to store (key,value) bindings and there are three kinds of nodes provided in an MPT, i.e., Leaf Nodes (LN), Branch Nodes (BN) and Extension Nodes (EN). A leaf node represents [key,value] pair, where key is the public prefix and value is the terminal value at the node. An extension node also represents [key,value] pair, but here value is the hash of the next node. The branch node is a 17-element array node and used to store viable leaf nodes or extension nodes when the prefixes of keys differ. Among the 17 elements, the first 16 elements are the

hex characters, representing the possible prefix of the next node. The last element is used to store the final target value if the path has been fully traversed. In MPT, each node is encoded in Recursive Length Prefix (RLP) code, which is designed to encode arbitrarily nested arrays of binary data, and denoted by its hash. It is noted that the MPT is fully deterministic, which means given the same (key,value) bindings, the MPT constructed from them is guaranteed to be the same regardless of their insertion order and thus have the same root hash.

The superiority of MPT is that it provides $O(\log n)$ efficiency for inserts, deletes and searches, while node insertion and deletion in Merkle Tree incur huge time cost. Moreover, with a publicly known root hash, anyone can prove that there exists a given value at a specific path in the MPT by providing the nodes along the way.

3 VQL Design

In this section, we present the design of our proposed VQL that supports efficient and verifiable data query services for blockchain-based applications. We first introduce the overview of the system architecture and the structure of the middleware layer. In order to guarantee the consistency between the middleware databases and the underlying blockchain (i.e., data authenticity), we then propose a database verification scheme to prevent falsified data from being stored in the middleware. We further simplify the verification process and put forth the simplified query result verification scheme for ordinary users who do not have blockchain data. This scheme enables query users to validate the query results by downloading partial database data. Finally, we conduct a comprehensive analysis on the data authenticity of our proposed design.

3.1 System Architecture

In this subsection, we introduce the architecture of our proposed cloud query service model and the middleware structure along with its update scheme. As illustrated in Figure 1, our cloud service model involves three parties, i.e., a blockchain as a distributed database storing a ledger, a middleware layer supporting efficient data query services through reorganizing the blockchain data, and an application layer providing various services for users.

3.1.1 Underlying Blockchain

In the blockchain system, transactions generated from users are stored in the blocks and form a public ledger. Some blockchain platforms such as Ethereum provide APIs to access the transactions stored in each block. In our system, we utilize these APIs to extract blocks, transaction and balance information stored in the blockchain. This service model can also be applied to other blockchain systems like logistics and supply chain, which record the information of goods delivery and market transaction using consortium blockchain.

3.1.2 Middleware Layer

Based on the blockchain data, the middleware layer extracts and reorganizes all information, e.g., block, transaction and balance, and constructs databases to support efficient data query and data analysis. Figure 2 gives an illustration of the designed middleware structure. Our middleware consists of a list of micro databases that contains the data generated in each time interval (e.g., in every

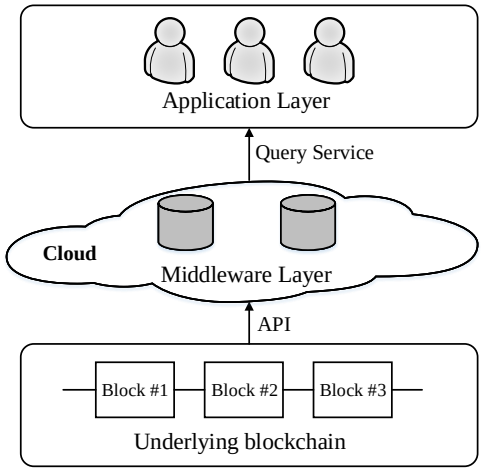


Fig. 1: Middleware-based cloud query service model for blockchain applications.

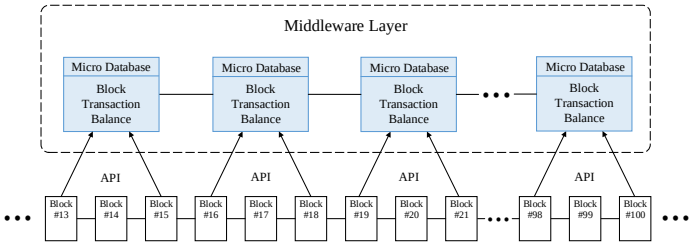


Fig. 2: Structure of the middleware layer.

day) after the specified time point. Each database has a header that contains a cryptographic hash value of the database and some database properties. The hash value of the database can be utilized to verify the data integrity of the database.

Given the underlying blockchain, the algorithm shown in Algorithm 1 updates the middleware layer with newly generated blocks. With the blocks being generated in the blockchain, the system will reorganize the blockchain data and update the middleware layer at a specific frequency, e.g., once a day as shown in the algorithm. At the end of each day, based on the new blocks that have been validated and confirmed by the miners, the middleware layer will be updated in time and support up to date query services. The middleware extracts the block, transaction and balance information from the blockchain data and constructs the corresponding databases. Then the fingerprint of these databases will be calculated. In order to avoid unnecessary modification of the databases, the middleware will extract information and construct databases only from the immutable blocks in a 'pull-based' method. It is noted that the frequency of daily update is our tentative setting, which can adjust based on the query requirement of different applications, e.g., hourly update for logistics system.

3.1.3 Application Layer

Since the middleware layer constructs databases based on a mature database software, it can provide various data query services for the application layer. Thus, the application layer can efficiently conduct various data analysis and machine learning tasks based on the blockchain data. Besides providing query services for normal users and data platforms, our application layer can also support public audit services for audit institutions. The auditors are able

Algorithm 1 Middleware update for Ethereum

Require: BC: Underlying blockchain
Ensure: DB: Middleware database
1: **for** each day **do**
2: Extract block, transaction and balance information from BC;
3: Calculate balance records;
4: Construct a new micro database mDB containing blocks, transactions and balance;
5: Fingerprint(mDB);
6: Merge mDB into DB;
7: **end for**
8: **return** DB;

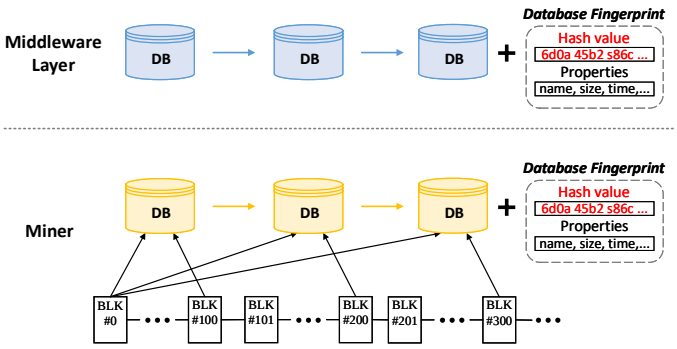


Fig. 3: Database verification scheme.

to audit the information in the underlying blockchain using easily verifiable evidence returned by the middleware layer.

3.2 Database Verification Scheme

In this subsection, we describe the verification scheme of databases, which can be carried out by miners and public users, that guarantees the consistency between the middleware and the underlying blockchain.

3.2.1 Miner Verification Scheme

Figure 3 illustrates the database verification process of the middleware based on the blockchain system. As shown in this figure, various transactions generated by users are stored in the blockchain by the miners. First, the middleware layer extracts transactions stored in the blockchain and reorganizes these data in the databases to provide efficient query services. Second, to prevent falsified data from being stored in the middleware, we generate a unique *fingerprint* for each constructed database in the the middleware layer. Finally, the constructed fingerprint of each database will be verified by miners and then stored in the underlying blockchain.

3.2.2 User Database Verification Scheme

We provide a public database verification scheme to guarantee that the data recorded in the middleware layer is consistent with the blockchain and can be verified. Our proposed middleware layer can be deployed in the cloud to be accessed by the public users for data query. Query users can usually trust the query results returned from the middleware layer since the databases stored in the layer have already been verified by miners. In case users have questions about the databases, they can fetch the block data from any honest miner and verify the authenticity of databases using the database fingerprint as the miners do.

3.2.3 Database Fingerprint

The database fingerprint uniquely represents the constructed individual database in the middleware layer. In our design, the fingerprint of the database is determined by two terms, i.e., the data content stored in the database and the property of the constructed database. For the data stored in the database, we first export the data in a unified and cross-platform format. Then a cryptographic hash value of these data will be calculated based on this format of file using a hash function, e.g., SHA-256. This hash value can be used by miners to check the consistency between the data stored in the database and the underlying blockchain data. The constructed database property contains the database name, the database time, the database size and the database software version. The property of the database can be used to construct the database in the subsequent database verification stage. Finally, the fingerprint can be generated by hashing on the two elements above. It is noted that the fingerprint value is calculated based on the data itself rather than the files stored on the disk. Therefore, the fingerprint is platform independent, which ensures that miners can obtain the identical fingerprint as long as the data stored in the database is the same.

3.2.4 Database Verification

Based on the database and its fingerprint, miners can verify the constructed databases in the middleware to guarantee the consistency between the middleware data and the underlying blockchain data. After constructing a new micro database to support efficient data query, the middleware layer will first give out its fingerprint.

The algorithm shown in Algorithm 2 describes the proposed database verification scheme for the constructed middleware layer. Since the miner stores the blockchain data locally, he can also construct another database based on his own local data using the same database generation program. The corresponding fingerprint will then be generated by the miner using the predefined hash function on this local database. Thus, for each miner, he can verify the consistency of data between the middleware layer and the underlying blockchain through comparing the two fingerprint values, i.e., the database fingerprint published by the middleware layer and the database fingerprint calculated by the miner based on his blockchain data.

Finally, after successfully verifying the consistency between the middleware layer and the underlying blockchain, miners will store the database fingerprint in the form of a transaction in the blockchain. The transaction transfers zero value from the miner to our middleware with the fingerprint information filled in the data field. The fingerprint is also inserted to an authenticated data structure, i.e., MPT, whose state will be written into the blockchain as well. Once the database fingerprint is recorded in the blockchain, this record cannot be tampered with in terms of the consensus scheme. In the application layer of our system, applications can query data from the middleware layer with trust after checking the database fingerprint stored in the blockchain.

3.2.5 Information Record in Blockchain

We propose to write the information regarding the database fingerprints into the underlying blockchain and these information can be

1. The detailed explanation on the MPT update and synchronization will be further elaborated on in Subsection 3.3.

Algorithm 2 Miner Database verification

Require: DB_{mid}: The database constructed in the middleware layer to be verified; DB_{bc}: The database constructed from blockchain by miner; $root_{bc}$: Root of MPT that maintained by miner; BC: Underlying blockchain.

Ensure: return ACCEPT if the database is verified correct; otherwise, return REJECT.

```
Get Fingerprint(DB_mid) from middleware;
Construct DB_bc from BC;
if Fingerprint(DB_mid) = Fingerprint(DB_bc) then
    Insert Fingerprint(DB_mid) into MPT and synchronize MPT to middleware[1];
    Write Fingerprint(DB_mid) and MPT root root_bc into BC;
    return ACCEPT;
else
    return REJECT;
end if
```

assured to be immutable by the consensus algorithm. Each time the miner verifies the validity of databases in the middleware layer, in addition to the database fingerprints themselves, he also records the root of Merkle Patricia Tree, which is used to store all database fingerprints. This tree root is a deterministic hash generated by all database fingerprints and provides a form of cryptographic authentication to the data structure. In other words, the tree root represents a unique state of the entire tree. Therefore, we write the tree root hash into the blockchain as well for the application layer to check. Note that, the information writing policy may differ when our verification scheme is applied in diverse blockchain systems, such as the public blockchain, private blockchain, and consortium blockchain [19]. In case of private blockchain or consortium blockchain, miners can be forced to write some certain information into the block of specific height. However, in the scenario of public blockchain, due to the propagation of transaction information and the competition among transactions, the information cannot be guaranteed to be written in the stipulated block.

3.2.6 Failed Verification Situation

During the database verification process, we also consider the failed verification situation. If the local fingerprint calculated by the miner is different from that provided by the middleware, an error report will be sent to the middleware layer. When the middleware layer receives a certain amount of failed verification reports, it will execute a diagnostic procedure to check the correctness of database until no error reports arrive. In case of extreme situations, e.g., a fork due to network partition, the middleware and miners will find the correct chain to catch up with. Meanwhile, the databases will be rebuilt and the fingerprints are revoked. The failed verification report scheme will help the middleware to correct false database fingerprints.

3.3 Simplified Query Result Verification Scheme

The database verification scheme in the last subsection is designed for the miners. For query users without blockchain data, they need to download the entire blockchain from credible miners and verify all databases by constructing them. It is a quite radical method to guarantee the authenticity of databases, but sometimes it is unnecessary for an ordinary user to download all data for just one simple query. To remedy this issue, we propose to employ the authenticated data structure for fingerprint management and put forth a simplified query result verification algorithm to ease the process of result verification for query users.

3.3.1 Merkle Patricia Tree for Database Fingerprints

Due to the uncertain factors in public blockchain systems, e.g., network delay and transaction fee, the middleware and users cannot get the height of the block where the database fingerprint is stored in advance. The block height is determined only after the fingerprint is indeed written into one block and confirmed by miners. Thus, we employ Merkle Patricia Tree to store these [fingerprint, height] pairs since it is able to prove the existence or non-existence of a given database fingerprint. In this way, query users can directly check the correctness of the given database fingerprint without searching the block containing the information. It is noted that the MPT is maintained by miners and will be updated each time miners finish verifying the validity of databases and writing database fingerprints into the blockchain. Moreover, the MPT data will also be synchronized to the middleware layer by miners so that the cloud can provide Merkle proofs for query users. The reason why miners adopt MPT instead of directly returning true or false for every fingerprint request from users is because providing all users with validation services is over-demanding for miners. The MPT enables miners only need to show the MPT root while leaving the proof work to cloud servers.

We use an example shown in Fig. 4 to instantiate the database fingerprints storage in MPT. We presume that initially there are four database fingerprints as presented in the key-value list, in which the key is the database fingerprint hash and the value represents the height of the block where the fingerprint information is written. Using these fingerprints, we can build the Merkle Patricia Tree as given in the figure. Here we neglect the detailed descriptions of the operations in MPT, e.g., insertion, update and deletion, since there have already been some implementations available.

3.3.2 Simplified Query Result Verification Process

Fig. 5 illustrates the relationships among the miners, users and the middleware-based cloud in our data verification scheme. Our system comprises three parties: *miners*, who mine the blocks and maintain credible block data in the underlying blockchain layer; *users*, who lie in the application layer and send queries to the cloud about the data in blockchain; *cloud query services*, which belong to the middleware layer and provide query services for users. The dashed arrow from miners to the cloud services signifies the aforementioned miner database verification, while the solid arrows represent the interactions in the simplified query result verification scheme. In our simplified scheme, miners only need to synchronize the MPT to the cloud services and provide the MPT root hash for query users if they request verification. Each time the user sends a data query to the cloud, the server will return a query result along with the database back-up files that this query involves and their corresponding Merkle proofs. This function of database back-up and reconstruction can be supported by some commercial database systems, e.g., MongoDB. Combining with the credible MPT root hash obtained from miners, the user can easily check the validity of those database fingerprints based on the Merkle proofs. If the user wants to further confirm the information about the root hash and database fingerprints, he can search the blocks according to the corresponding block height stored in MPT.

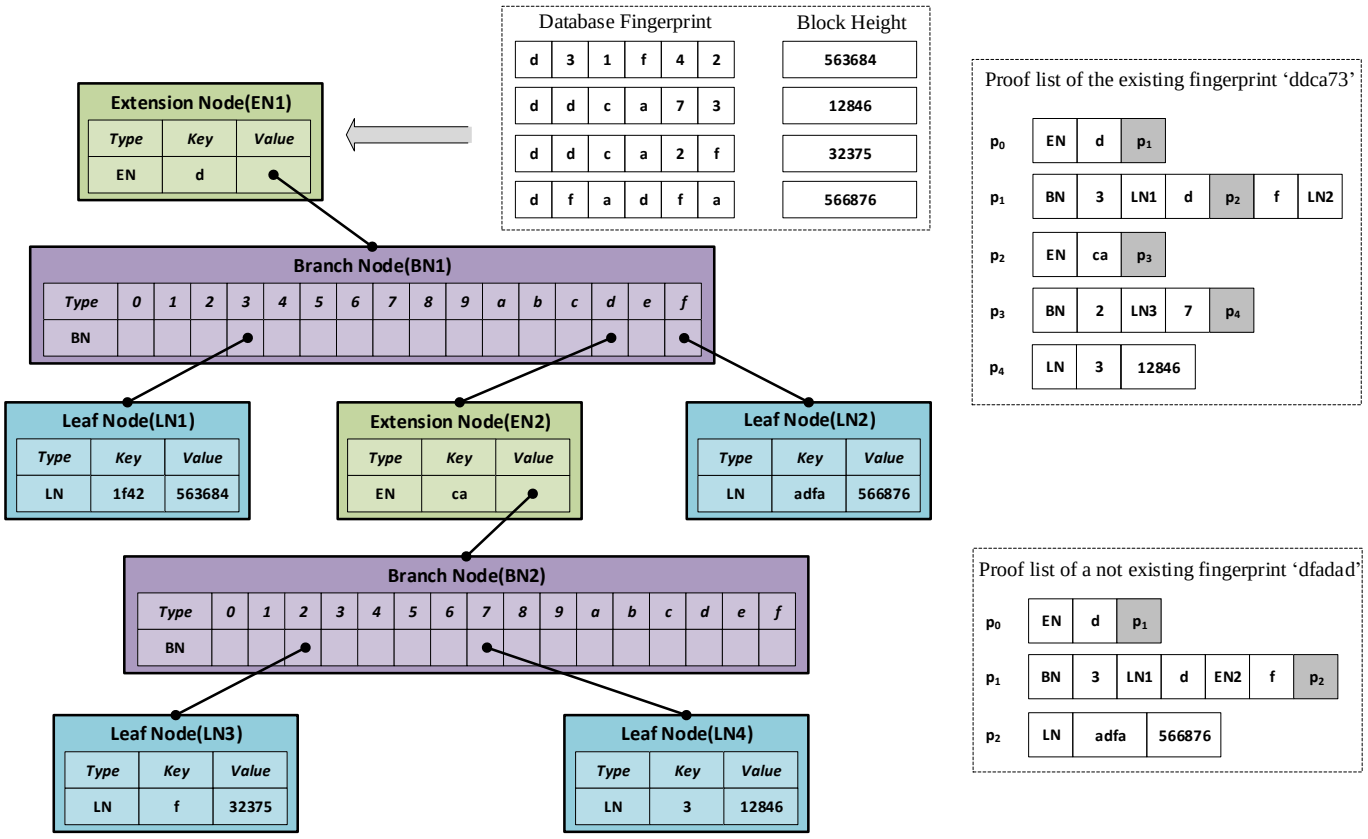


Fig. 4: An illustrative example of database fingerprints MPT.

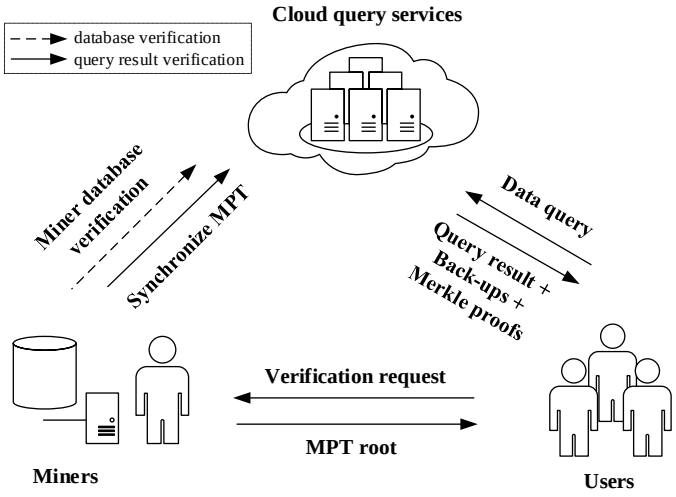


Fig. 5: Data verification scheme.

Algorithm 3 shows the simplified query result verification algorithm performed by query users. When the user requests a data query to the middleware, he will get a query result $result_{mid}$ from the server together with the fingerprints of all databases involved, i.e., DBs. After downloading the corresponding database back-up files, the user can reconstruct these databases and calculate their fingerprints. Meanwhile, he will send a verification request to the miners and thereby get the latest MPT root hash $root_{bc}$, which signifies the newest state of all verified databases. With the database fingerprints calculated, the user can send them to the middleware layer and obtain the Merkle proof for every fingerprint. Based on the proof for each fingerprint, he will calculate the root hash $root_m$ by themselves and then compare with the true root hash $root_{bc}$. When the two root hashes are equal and the key is in accord with the path, the correctness of this fingerprint can be guaranteed. The process of proving the presence of the fingerprint using MPT root and Merkle proof is included in *Prove* function and will be detailed in the Merkle proof part. If all databases involved are confirmed correct, then the user can query the databases that are locally constructed from back-up files and get the query result $result_l$. When this result is identical to the previous result $result_{mid}$ from the middleware, the user can finally trust and accept the result.

3.3.3 Merkle Proof for Fingerprints

Now suppose a query user wants to check the existence of the database fingerprint 'ddca73' which already exists in the MPT (see Fig. 4). The value of this key is stored in the leaf node LN4 and its search path from root to leaf is fEN1,BN1,EN2,BN2,LN4g. Based on the path, our middleware layer can provide a Merkle proof, which is a list of RLP code of the nodes along the path (see the right part of Fig. 4), for the user to prove the existence of the key. In this case, the Merkle proof for 'ddca73' is a 5-element array, i.e., p_0 to p_4. Each node is referenced inside the previous element except the root node p_0. Using this list, the user can check the correctness of the value and RLP code of each element in the array successively from head to tail, i.e., in the order from root to leaf. If the root hash finally calculated is identical to the publicly known root value and the prefixes along the path equal to the fingerprint, then this database fingerprint is

Algorithm 3 Simplified query result verification

Require: $root_{bc}$: Root of MPT that stored in blockchain; DBs: Middleware databases that the user query involves; $result_{mid}$: Query result provided by middleware layer; $proof$: Merkle proof of a given database fingerprint;
Ensure: return ACCEPT if $result_{mid}$ is correct; otherwise, return REJECT.

```
Get the latest MPT root root_bc recorded in blockchain from miners;
verified ← FALSE;
for each DB ∈ DBs do
    Construct DB from the back-ups in the middleware layer;
    Send Fingerprint(DB) to the middleware layer;
    Get the Merkle proof proof from the middleware layer;
    verified ← Prove(root_bc, fingerprint, proof);
    if not verified then
        break;
    end if
end for
if verified then
    Query DBs locally and get the query result result_l;
    if result_mid = result_l then
        return ACCEPT;
    else
        return REJECT;
    end if
else
    return REJECT;
end if
```

Algorithm 4 Prove algorithm

Require: $root_{bc}$: Root of MPT that stored in blockchain; $fingerprint$: Fingerprint of the database to be checked; $proof$: a n-element list of p_i, i.e., Merkle proof of the given database fingerprint;
Ensure: return TRUE if $fingerprint$ exists in MPT; otherwise, return FALSE.

```
if Hash(p_0) ≠ root_bc then
    return FALSE;
end if
for i ← 0 to n − 1 do
    if i = n − 1 then
        if key in p_i conforms to fingerprint then
            return TRUE;
        else
            return FALSE;
        end if
    end if
    if i < n − 1 then
        if key in p_i conforms to fingerprint and key's value =
        RLP(p_{i+1}) then
            continue;
        else
            return FALSE;
        end if
    end if
end for
```

considered to truly exist. Algorithm 4 shows the pseudo-code of the *Prove* algorithm executed by the query user to verify whether the database fingerprint exists in MPT.

Similarly, we can also utilize the Merkle proof to prove the non-existence of a given key. Suppose a user reconstructs a database using broken or tampered files and thus calculates a wrong fingerprint, e.g., 'dfadad' in the figure, which does not exist in the MPT. Our server will return the Merkle proof based on the search path {EN1,BN1,LN2}, i.e., p_0 to p_2 as shown in the figure. Here the hash of root p_0 can be verified by calculating from head to tail and still equals to the root hash obtained from miners. Nevertheless, the prefixes generated by the proof differ from the fingerprint key, which means the fingerprint does not exist in the MPT.

3.4 Data Authenticity Analysis

Since the user receives the query result from the middleware, as long as the queried database is consistent with the underlying blockchain, the authenticity of the queried data is guaranteed. Thus, we conduct the data authenticity analysis from three aspects: the rewarding scheme for miners, the integrity of databases and the verifiability of query results.

3.4.1 Rewarding Scheme for Miners

In our cloud service, the verification of databases in the middleware layer is realized by miners, which may cost some computing resources and storage space. Thus, a rational rewarding scheme is required to incentivize miners to verify the databases. Owing to the different demands and scenarios of the blockchain systems, the rewarding schemes for miners in our query model may differ. For the private blockchain system, since the miners and middleware layer are private to provide services, the verification and record fees are not needed. As for the consortium blockchain system, depending on the various agreements between communities in the consortium, the middleware layer may need to pay the fees or not. When applied to the public blockchain system, our middleware will give some rewards to the miners or mining pools [20] who successfully validate the databases.

Since the above rewarding scheme for database verification is not supported by the existing blockchain systems, we give two possible solutions to make our verification scheme practical. First, we can implement a new blockchain system based on an existing open-source project, incorporating the incentive mechanism for verification. In order to get the rewards, miners can validate the middleware databases and record corresponding database fingerprints into the blockchain. The validation process, the rewarding mechanism and the management of fingerprints are all hard-coded in the blockchain peer nodes. Second, we can deploy the smart contract on the current blockchain system to facilitate the verification process of miners. Miners can construct the database from their own blockchain data using the same database generation code provided by our middleware. Then the database fingerprint can be calculated and sent to the smart contract for confirmation. Finally, our middleware will announce the correct fingerprint and send the rewards to the miners who correctly verified the database. More miners will participate in the verification task if the reward is attractive enough. The smart contract can also maintain the MPT storage of database fingerprints in the simplified query result verification process.

3.4.2 The Integrity of Databases

The consistency between the databases in the middleware layer and the underlying blockchain data is realized through the database verification scheme by the miners. Each time a new database is constructed, the middleware layer will back up the database and publish the back-up files and its fingerprint. In the meantime, miners can construct another database based on his blockchain data following the same rules and calculate its fingerprint using the predefined hash function. If the fingerprint of this database is the same as the one given by the middleware, then the database is verified correct. Moreover, the integrity information is immutable since the fingerprint will be written into the blockchain after verification and managed by the MPT structure.

3.4.3 The Verifiability of Query Results

After the integrity of databases in the middleware layer is guaranteed, the query results that users receive should also be consistent with the middleware databases. We provide two methods to realize the verifiability of query results, i.e., user verification in the database verification scheme and the simplified query result verification scheme. The user database verification requires users to download all blockchain data and check the consistency like miners, the authenticity analysis of which is just conducted. It is noted that when we request data from the miners, we will firstly connect to the anchor nodes in the blockchain network, which means the miners we query are assumed to be absolutely reliable. Therefore, the situation of malicious miners is trivial and out of the scope of this paper. The simplified query result verification scheme allows users to download only the involved databases rather than all databases and check the validity of their fingerprints by leveraging the MPT structure. Since the databases are reconstructed based on the back-up files and their fingerprints are calculated locally by users, the authenticity of the involved databases can be ensured if these fingerprints indeed exist in the MPT maintained by miners. Finally, users can query the valid databases locally and check whether the result is consistent with the query result returned by the middleware layer.

4 Implementations and Evaluation

To test the feasibility and performance of our cloud query service, we implement a prototype on a testnet of the well-known blockchain system Ethereum.

4.1 Prototype Implementation

Our middleware supports user-friendly APIs for user applications and APIs for the underlying blockchain. The user application APIs support various queries and database verification for auditing, including the query interface and validation interface for the block, the transaction and the balance information. Meanwhile, the blockchain APIs support query functions to collect records from the blockchain, e.g., the data request interface for the block, the transaction and the global state of the blockchain. We employ the popular document-oriented database MongoDB for data storage of the middleware. The reason why we use the MongoDB is that it can support efficient query on general and rich data, e.g., arbitrary forms of transactions and smart contracts. It can also achieve good reading performance by building the indexes. The MPT for fingerprint storage is implemented in JavaScript and stored in LevelDB. To evaluate the system performance without the interference of network communication, we build up the experiment platform on a cloud server with Intel Xeon 2.67GHz CPU and 32 GB RAM, running Ubuntu 16.04 LTS. Our data query services are based on the blockchain data of Rinkeby network, one of the popular Ethereum testnets, with block height varying from 0 to 8,000,000.

4.2 Performance Evaluation

The process of synchronization from scratch in blockchain systems usually needs to be done only once because of the fact that blockchain data is immutable. Moreover, the time cost of the synchronization process is generally dominated by the network bandwidth and the performance of the physical machine. Nodes with low network bandwidth or bad performance may take several days to catch up with other peers. Therefore, the evaluation of blockchain synchronization is out of the scope of this paper. We test various data query services in terms of throughput, block query, transaction query, account query and range query. We contrast our proposed VQL with the Geth client, which is an official Go implementation of the Ethereum protocol, in terms of query efficiency.

4.2.1 Throughput

We first evaluate the throughput performance of our proposed system VQL comparing with ETH client. The ETH client synchronizes the blockchain to about 8,000,000 and our VQL also organizes all information within the same block height. In addition to the comparison between the ETH client and VQL, we also evaluate the query efficiency of VQL with different blocks synchronized, i.e., 500,000, 4,000,000 and 8,000,000. Three kinds of queries are conducted, including querying a block by the block number, querying a transaction by transaction hash, and querying the balance of an account by address. As shown in Fig. 6a, the throughput of VQL is about 13.2, 2.1, 5.5 times as that of ETH client in terms of block, transaction and account query respectively. When we query a block by the block number, the VQL and ETH client can support 1.88K queries=s and 142 queries=s, respectively. For querying a transaction by the transaction hash, the VQL and ETH client are able to process about 70.6 queries=s and 33.5 queries=s. If we query the balance of an account by address, both systems can achieve higher throughput (i.e., 2.14K queries=s and 387.9 queries=s) because of the relatively smaller amount of accounts. The results show that our proposed VQL can achieve a higher throughput than the native ETH client. From the performance of VQL under different load scenarios, we can see that the increasing number of synchronized blocks will degrade the query throughput of all query categories. The throughput of transaction query drops rapidly because the amount of transactions increases sharply as the block height grows.

4.2.2 Block Query

In our experiments, query efficiency is a critical criterion for the proposed query supported system. In the blockchain, various transactions generated by users are stored in the blocks. Thus, we first compare the block query time of different systems (e.g., ETH client and VQL) to show the query efficiency of our system. ETH client provides a JSON RPC API to support the block query. Accordingly, we develop an API in the middleware layer to provide query services about blocks.

Since a single block query can usually be completed in milliseconds, we query for a randomly selected list of blocks

(a) Throughput

(b) Block query

(c) Transaction query

(d) Account query

Fig. 6: Query performance of ETH client and VQL.

and record the time of completing these queries. We conduct experiments of block query based on scenarios with block number from 0 to 50,000. As shown in Fig. 6b, the block query time is compared with ETH client and VQL will different loads. With more blocks queried, the query time is signi cantly increased using ETH client, while the time of VQL can still remain at a relatively lower level. This ETH client requires plenty of query time, for example, 351.9 seconds in the evaluation of the 50,000-block scenario. On the contrary, our proposed system VQL can save much query time, which optimizes the data storage for faster queries (e.g., 26.6 seconds in 50,000-block scenario). From the comparison between different loads of VQL, we observe that the number of synchronized blocks has limited effects on the query ef ciency since the query time only increases slightly.

4.2.3 Transaction Query

The query about individual transaction information is also supported in our system and we conduct experiments on the query time of transactions. The native ETH client provides limited APIs for the retrieval of transaction details while our VQL can support queries on transactions by all attributes that a transaction has. In this experiment, we choose the common API, i.e., query by the transaction hash, to present the comparison of the query ef ciency.

As shown in Fig. 6c, the transaction query time is compared between ETH client and VQL with different loads. Since a single transaction query can usually be completed very fast, we query for a bunch of randomly selected transactions to evaluate the time. We test cases with different numbers of transactions in the experiment, from 0 to 50,000 transactions. The number of transactions almost linearly promotes the query time in all cases. But VQL takes only about half of the time that ETH client uses to query the same amount of transactions under the same data load. As for the comparison between different number of synchronized blocks in VQL, the query time of 8,000,000-block scenario is much longer than that of 500,000-block case. It is because the volume of transactions grows dramatically when the block height increases.

4.2.4 Account Query

In our middleware layer, each constructed micro database contains two parts of data: the transaction details and the balance details of all accounts. Apart from the original blockchain data, the account balance also provides an extra historical balance description for each account, e.g., balance change in each day. Since the native ETH client only provides the API for current balance query, we also test the same function in our VQL system.

As shown in Fig. 6d, we conduct experiments to evaluate the query time of account balance for ETH client and VQL. Because the query time of a single account is too small to measure, we still query for a randomly selected list of accounts to test the ef ciency of balance queries. Scenarios with different numbers of accounts, from 0 to 50,000 accounts are tested in the experiment. We can see that the query time of account balance increases linearly as the number of accounts grows. The query of account balance with the same amount in VQL can be completed within one fth of the time that the ETH client takes, i.e., 128.9s. This is because that, in our proposed middleware, the information of account balance is calculated in advance and well organized in databases. In addition, the comparison between different loads in VQL shows that the number of synchronized blocks slightly promotes the query time.

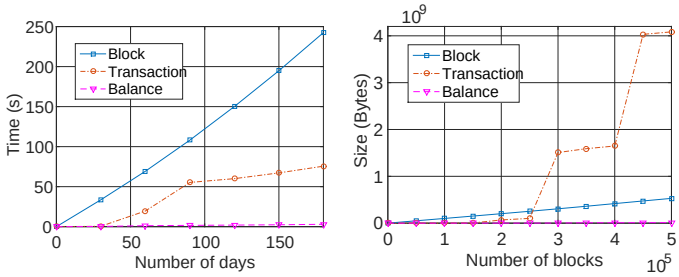
4.2.5 Range Query

Besides the individual account query, range query is also supported by the middleware layer since the application layer is usually required to conduct various data analysis and machine learning tasks. For these tasks, many features will be extracted through a range of data, e.g., accounts that have transactions in one day or transactions with amount over 100 ETH. Our middleware can provide this ability of data query within a speci c range while the native ETH client cannot perfectly support.

In our experiments, we conduct performance evaluation about range query for block, transaction and account, respectively. Considering many applications related to data analysis, we implement two kinds of range queries, i.e., temporal range query and numerical range query, for the information of block, transaction and balance. The temporal range query means the query on blockchain

		ETH client	VQL
Temporal range query	Block	40.53s	0.04ms
	Transaction	1625.88s	0.036ms
	Balance	10.23s	0.041ms
Numerical range query	Block	—	0.034ms
	Transaction	—	0.033ms
	Balance	—	0.036ms

TABLE 1: Evaluation of range query.



(a) Verification time (b) Database size

Fig. 7: Performance of miner database verification.

data within a specific time range, e.g., the transactions generated last month. The numerical range query represents the query on some numerical fields of the data, e.g., the transactions with value less than 1 ETH. As shown in Table 1, the time of different range query categories is compared with ETH client and VQL. We query blocks generated in one day and record the query time. The VQL can finish the query with 0.04ms while the ETH client needs 40.53s. Then we query transactions within a randomly chosen day and record the time used. Our VQL completes the query within 0.036ms and the ETH client costs 1625.88s. Finally, we query the account balances that have changes in one day, which means transactions are performed between these accounts. The experiment result shows that the VQL needs 0.041ms while the ETH client uses 10.23s. It is noted that the ETH client does not directly support temporal range query. To achieve it, we traverse the blocks using the block number and get the transactions inside. However, the numerical range query cannot be supported even using this method since the ETH client has to read all blockchain data to judge the numerical values, which is excessively time-consuming. Therefore, we mark the inapplicability using ^{0} ^{0} in the table for numerical range query. In general, the proposed VQL needs much less time to finish different range queries than the ETH client. Our VQL shows remarkable advantages over the ETH client due to the well-organized micro databases in the middleware, which are very efficient for range queries.

4.2.6 Database Verification

Database verification efficiency is also an important criterion for our proposed system. We set the generation frequency of database fingerprint to once a day, which means the middleware produces the fingerprints for block, transaction and balance using the respective daily data. As shown in Fig. 7a, we record the time of database verification after the blocks are generated in the blockchain each day. When the middleware layer has constructed databases based on the blockchain for 180 days, the verification time of block databases for a miner is 242.4s and that of transaction databases is 75.6s. The balance databases take the least time, i.e., 2.98s for 180 days, since the size of involved information is quite small. With more daily databases generated by the middleware, the database verification time increases. We can see that there is a fluctuation in the transaction database between 50 and 100 days. This is because the amount of transactions in these days suddenly grows, which leads to more verification time. Thus, our proposed system is able to efficiently verify databases constructed in the middleware layer and applicable to practical blockchain systems.

4.2.7 Database Size

Considering the storage space efficiency, we also test the size of databases to be verified in the middleware layer during the database verification process. We record the size of each database for block, transaction and balance as the blocks are generated in the blockchain. As shown in Fig. 7b, when the middleware layer has constructed databases for 500,000 blocks in the blockchain, the size of transaction database stored in the middleware layer reaches about 4GB while the size of block database is around 500 MB. We can observe from the figure that the size of the transaction database increases notably twice due to the large amount of transactions in some blocks. The balance database always occupies the least storage and its size is only 12MB even when the number of blocks reaches 500,000. Thus, our proposed system can efficiently store the databases constructed in the middleware layer to provide query services and database verification.

4.2.8 Proof Cost in MPT

The cost of simplified query result verification is dominated by the communication overhead incurred by Merkle proof. The size of Merkle proof is mainly decided by the number of layers in MPT. The deeper the leaf node locates in MPT, the longer its search path becomes. Thus, we measure the size of proof that the middleware layer returns for each database fingerprint. In our evaluation, we employ SHA-256 hash function to generate the fingerprint for the database, thus the key to be stored in MPT has 256 bits. We insert 2,000 keys into the MPT and record the average length of Merkle proof that MPT provides by invoking the prove function for each key. As presented in Fig. 8a, the size of Merkle proof is only a few kilobytes and closely associated with the depth of key. The depth of the fingerprint is principally distributed between 7 and 13, and the proof size gradually increases as the depth grows, which conforms to our previous analysis. This is because Merkle proof is a list of nodes along the path and the RLP code of one node is about 100 bytes. Compared with the size of the block data needed in miner database verification, the overhead of giving the Merkle proof is practically negligible.

4.2.9 Storage Cost in MPT

Since the MPT for database fingerprint is updated by miners and will be synchronized to the middleware layer, it will cost storage space in both miners and the middleware layer. In order to show the storage cost of MPT with the amount of fingerprint increasing, we investigate the size of the LevelDB database files generated by the MPT when the total amount is 1,000, 5,000, 10,000, 20,000, 30,000, 50,000. Observing from Fig. 8b, we can see that the amount of fingerprint linearly promotes the storage cost, which indicates that MPT does not bring about much cost of extra storage space as the amount of fingerprint grows. The storage cost increases to 90 MB when the fingerprint amount reaches 50,000, which is relatively small compared with the size of

databases constructed in the miner database verification process. Therefore, the storage cost is acceptable to achieve our simplified query result verification scheme.

4.2.10 Throughput and Proof Size

In our simplified verification scheme, the middleware layer will return a Merkle proof for each query from users. Thus, we investigate how many verification requests the middleware is able to handle concurrently and how much overhead it costs to return a Merkle proof. The performance is presented in Fig. 8c, which includes the throughput and proof size under various number of fingerprints. We observe that the throughput of returning proofs decreases when the amount of fingerprints grows. This is because the MPT becomes larger when more fingerprints are stored, which leads to longer search time for each fingerprint. The middleware can support 3,000 verification requests per second with 50,000 fingerprints stored, which is acceptable as well. Meanwhile, we can also see that the average size of Merkle proof rises slowly when the number of fingerprints increases.

4.2.11 Depth Distribution

We further investigate the reason behind the proof size and observe that the distribution of fingerprint depth greatly change under different cases. Fig. 8d shows how fingerprint depth distributes under scenarios with different fingerprint amounts. When the amount of fingerprints is 1,000, the depth mainly distributes around 7 and the proportion of 7 exceeds 65%. As the amount increases, the majority of fingerprint depth rises slightly. The depth of 9 accounts for more than 60% of the whole fingerprints when the total amount reaches 20,000. In the scenario with 50,000 fingerprints, the proportion of depth 11 gradually grows to about 30%, leading to a higher average depth. Combining with the previous observation from the proof cost in Fig. 8a, the increasing average proof size conforms to the distribution of fingerprints. Compared with the size of database itself in the middleware, the proof cost in our simplified query result verification is relatively small.

5 Related Work

Within a wide number of works on the blockchain-based data query, our design is strongly related to the following research categories.

5.1 Blockchain

As the first successful application of blockchain, Bitcoin [16] has attracted much attention to the blockchain technique. It provides a new way to store transactions on the distributed ledger without the risk of tamper. Ethereum [18], as the successor of Bitcoin, expands the functions by introducing the design of smart contract, which enables more flexible operations on the cryptocurrency. Apart from cryptocurrency, the blockchain methodology contributes to other technologies as well. Provenance [21] establishes the auditable records behind all physical goods for suppliers. ChainSQL [22] combines blockchain with distributed databases to facilitate a decentralized, auditable and efficient application platform for database users. Other efforts have been done to improve the anonymity and security of the blockchain [23], [24].

5.2 Efficient Query

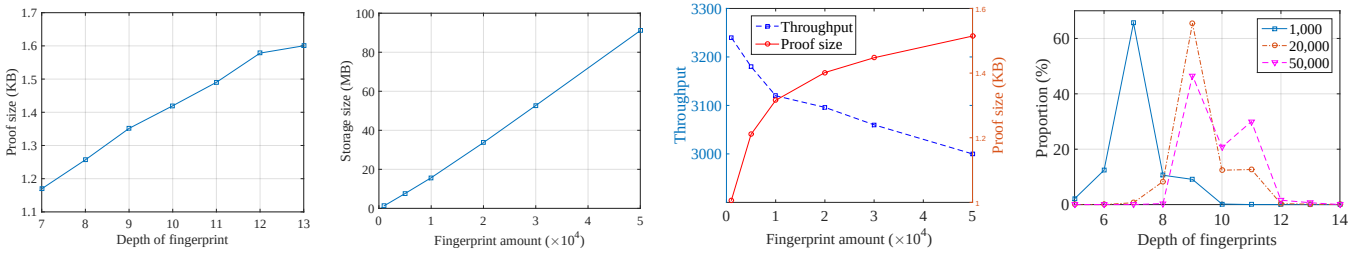
Much effort has been made to support various query and ensure a prompt response. Etherscan [25] is a block explorer and analytics platform where users can explore and analyze data from Ethereum blockchain. With the help of its virtual machine, Etherscan can also provide extra information like internal transactions and state changes in the smart contract. Project Toshi [8] is a fully implemented Bitcoin protocol and supported by PostgreSQL. It offers a RESTful API for large-scale web applications and blockchain data analysis. Blockchain.com [11] provides developers with RESTful service by encapsulating the blocks, transactions and address APIs in the Bitcoin. Aiming to support efficient and accurate queries for certificates, ECBC [9] utilizes a tree structure to facilitate the retrieval of historical transactions. EtherQL [10] employs the conventional database to provide efficient queries for blockchain data analysis. BlockSci [12] can support versatile analysis tasks for different blockchain systems by virtue of an in-memory and analytical database.

The explorers above contain rich information and enable users to explore blocks, transactions and accounts by providing basic interfaces, but the functions of these public APIs are limited. More complex queries (e.g., range queries) for blockchain data are not supported. Moreover, these systems do not provide verification functions to ensure the validity of the query result. In other words, those limitations are great obstacles to providing versatile queries and verifiable query services for blockchain systems.

5.3 Verifiable Query

Verifiable query technique that guarantees result integrity is also a hot research topic and has been extensively studied [26], [27]. These studies mainly focus on outsourced databases and can be categorized into two typical methods: circuit-based verifiable computation (VC) techniques and authenticated data structure (ADS). The VC approach like SNARKs [28] can support general queries over databases since it is able to verify arbitrary computation tasks from untrusted workers. But this method incurs a very high and sometimes unacceptable overhead. In addition, it requires a preprocessing step to hard code the data and query information into the proving key and the verification key, which also degrades the efficiency. To remedy this issue, Ben-Sasson et al. [29] propose a variant of SNARKs called zk-SNARKs, where the size of the output circuit depends on the upper-bound size of the query program. More recently, vSQL [30] provides publicly verifiable SQL queries for dynamic databases by utilizing the interactive-proof protocol. However, it is only restricted to the relational database scenario.

The ADS method employs data structures tailored to specific queries, thus it is generally more efficient in comparison. One of the ADS methods is digital signature scheme, which can be used to authenticate the content of digital messages using asymmetric cryptography. Pang et al. [31] present a verifiable B-Tree by adding signed digests to the B+-tree to authenticate query results based on digital signature. Merkle Hash Tree (MHT), which is a hierarchical tree, belongs to the ADS method as well. GSSE [32] utilizes Merkle Patricia Tree, which is a variant of MHT combining with the prefix tree, to enable verifiable and secure data search in cloud services. Xu et al. [14] design a framework named vChain that adopts the accumulator-based ADS scheme to achieve dynamic aggregation over various query attributes. However, this



(a) Proof cost with depth (b) Storage cost (c) Throughput and proof size (d) Depth distribution

Fig. 8: Performance of simplified query result verification.

approach entails great modification to the underlying blockchain structure and numerous preprocessing operations.

To the best of our knowledge, no research has been done to improve the efficiency of providing versatile query and public verifiability without radically changing the current blockchain systems. This paper is an extension of our previous work [33], adding details of the database verification scheme, the simplified query result verification scheme and abundant performance evaluation.

6 Conclusion

In this paper, we propose VQL, a cloud query service layer that can provide efficient and verifiable query services for the blockchain system. The proposed framework has a three-layer architecture, including the underlying blockchain network, the middleware layer and the application layer. To realize this system, first, the middleware layer extracts the data stored in the underlying blockchain and reorganizes them in databases to provide various query services efficiently for the upper application layer. Second, to prevent falsified data from being stored in the middleware, a cryptographic hash value, named as fingerprint, is calculated based on each constructed database. Finally, the database fingerprint is recorded in the blockchain after being verified by miners. In order to ensure the data integrity, we design the database verification scheme for miners and the simplified query result verification scheme for public users. We implement VQL on the cloud and conduct extensive experiments based on a practical blockchain system Rinkeby. The evaluation results demonstrate that VQL can effectively and efficiently support various data query services and guarantee the authenticity of query results for the blockchain system. Our proposed query service can be deployed on the cloud for practical applications and accessed by public users for efficient and versatile data queries.

Acknowledgement

This work was supported in part by the HK RGC GRF PolyU 15217321 and PolyU 15216220, the HK ITF ITS/081/18, and Guangdong Basic and Applied Basic Research Foundation 2020A1515111070.

References

[1] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symposium on Security and Privacy (SP)*, pp. 839–858, 2016.

[2] F. Tian, "An agri-food supply chain traceability system for china based on rfid & blockchain technology," in *Proc. of IEEE Service Systems and Service Management (ICSSSM)*, pp. 1–6, 2016.

[3] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "Medshare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. 5, pp. 14757–14767, 2017.

[4] M. Ali, J. C. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains.," in *Proc. of USENIX Annual Technical Conference (ATC)*, pp. 181–194, 2016.

[5] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *Proc. of IEEE PerCom*, pp. 618–623, 2017.

[6] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, "The log-structured merge-tree (LSM-tree)," *Acta Informatica*, vol. 33, no. 4, pp. 351–385, 1996.

[7] X. Wu, Y. Xu, Z. Shao, and S. Jiang, "LSM-trie: An LSM-tree-based ultra-large key-value store for small data items," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pp. 71–82, 2015.

[8] "Coinbase: Toshi project.." https://github.com/martindale/toshi.

[9] Y. Xu, S. Zhao, L. Kong, Y. Zheng, S. Zhang, and Q. Li, "ECBC: A high performance educational certificate blockchain with efficient query," in *International Colloquium on Theoretical Aspects of Computing*, pp. 288–304, Springer, 2017.

[10] Y. Li, K. Zheng, Y. Yan, Q. Liu, and X. Zhou, "EtherQL: A query layer for blockchain system," in *International Conference on Database Systems for Advanced Applications*, pp. 556–567, Springer, 2017.

[11] "Blockchain.com." https://www.blockchain.com/explorer.

[12] H. Kalodner, M. Möser, K. Lee, S. Goldfeder, M. Plattner, A. Chator, and A. Narayanan, "BlockSci: Design and applications of a blockchain analysis platform," in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2721–2738, 2020.

[13] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69–73, 2012.

[14] C. Xu, C. Zhang, and J. Xu, "vChain: Enabling verifiable boolean range queries over blockchain databases," in *Proceedings of the 2019 international conference on management of data*, pp. 141–158, 2019.

[15] N. Z. Aitzhan and D. Svetinovic, "Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams," *IEEE Transactions on Dependable and Secure Computing*, 2016.

[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[17] "Merkle patricia tree." https://github.com/ethereum/wiki/wiki/Patricia-Tree.

[18] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[19] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE Transactions on Industrial Informatics*, 2017.

[20] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein, "Bitcoin mining pools: A cooperative game theoretic analysis," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 919–927, International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[21] "Provenance." https://www.provenance.org/.

[22] M. Muzammal, Q. Qu, and B. Nasrulin, "Renovating blockchain with distributed databases: An open source system," *Future Generation Computer Systems*, vol. 90, pp. 105–117, 2019.

[23] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE Symposium on Security and Privacy (SP)*, pp. 459–474, 2014.

[24] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. of ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 3–16, 2016.