An Efficient Query Scheme for Hybrid Storage Blockchains Based on Merkle Semantic Trie

1st Qingqi Pei
State Key Laboratory of ISN
Xidian University
Xi'an, China
qqpei@mail.xidian.edu.cn

2nd Enyuan Zhou*
School of Cyber Engineering
Xidian University
Xi'an, China
eyzhou@stu.xidian.edu.cn

3rd Yang Xiao
State Key Laboratory of ISN
Xidian University
Xi'an, China
talentedx@163.com

4th Deyu Zhang
School of Cyber Engineering
Xidian University
Xi'an, China
303205844@qq.com

5th Dongxiao Zhao
State Key Laboratory of ISN
Xidian University
Xi'an, China
bigctime@163.com

***Abstract*—As a decentralized trusted database, the blockchain is finding applications in a growing number of fields such as finance, supply chain and medicine traceability, where large volumes of valuable data are stored on the blockchain. Currently, the mainstream blockchains employ a hybrid data storage architecture combining on-chain and off-chain storage. Real-time distributed search of mass data stored in this hybrid system is now a major need. However, previous fast retrieval schemes for the blockchain system are aimed only at on-chain data without considering their relevance to off-chain data, and thus fail to meet the requirement. In this paper, we propose an efficient blockchain data query scheme by introducing a novel Merkle Semantic Trie-based indexing technique without modifying the underlying database. A consensus on-chain index structure is constructed using the extracted semantic information of the off-chain data to create a mapping between the on-chain and off-chain data, thus enabling real-time data query both on and off the chain. Our scheme also provides multiple complex analytical query primitives to support semantic query, range query, and even fuzzy query. Experiments on three open data sets show that the proposed scheme has good query performance with shorter query latency for four different search types and offers better retrieval performance and verification efficiency than those available.**

***Index Terms*—blockchain, semantic extraction, distributed search, indexing**

I. Introduction

With the success of cryptocurrency systems such as Bitcoin [1] and Ethereum [2] in recent years, the blockchain as their core technology has attracted growing attention. As a distributed ledger technology, the decentralized tamper-proof blockchain guarantees high security and reliability, as well as data transparency [3]. At present, it has found applications in such fields as supply chain [4], financial infrastructure [5], and data sharing [6]. A common application scenario of the blockchain is the storage of massive valuable data owing to its safety and tamper-resistance characteristics. The mainstream data storage mode for now is a combination of on-chain and off-chain storage [7], the key to which is real-time data query. The user may search the data of interest, e.g. those containing certain keywords or in a certain range of size. Thus a blockchain system needs to provide real-time query capabilities.

Unfortunately, no blockchain systems available offer this real-time query. A few studies did explore in this aspect [8] [9] [10] [11], but focused only on querying the data stored on the chain, without considering the correlation between the on-chain and off-chain data in a hybrid on-chain/off-chain architecture.

We identify three major challenges to real-time query in such hybrid storage systems:

1. Existing systems adopt hash values for mapping between the on-chain data and off-chain data. This can only ensure that off-chain data are not tempered with, but unable to acquire from on-chain data valuable information such as the semantic and meta information of the off-chain data.

2. Blockchain systems were initially designed without query requirements in mind. On-chain data are stored as $< key, value >$ pairs in NoSQL databases, e.g. levelDB. The key value is often the only hash value for the transaction data and state data. The data can only be queried with corresponding hash values, which are random values calculated from the data by hash algorithms such as sha-256 and Keccak-256. It is impossible for the users to search with keywords of their choice, let alone more complex operations such as range query and fuzzy query.

3. The blockchain is an append-only data structure [12]. Blocks are connected by hash pointers and there is no other correspondence between data contexts. Therefore, data query and verification require traversing from the last block all the way back to the genesis block to guarantee that the search is complete. Since the number of random reads to the disk in the traversal process increases in proportion to the number of blocks, the time consumption may be unacceptable as more and more blocks are added. Furthermore, light nodes that only

* Enyuan Zhou is corresponding author.

2575-8462/20/$31.00 ©2020 IEEE
DOI 10.1109/SRDS51746.2020.00013

contain header information in the system need to connect to the full nodes that contain all data for query, and it is difficult for the light node to verify the results returned from the full node due to the absence of on-chain data information.

To address the above three challenges, we propose a real-time query scheme for the hybrid storage blockchain system. We employ an improved TF-IDF model [13] to extract semantic information of the off-chain data on a distributed peer-to-peer network. The extracted semantic information and the addressing structure of the data content are stored as transactions on the chain. An index structure called the merkle semantic trie (MST) is constructed based on the inverted list, hash pointers and B + trees, thus offering support for semantic (keyword) query, range query and fuzzy query, along with verification of the query results. Experiments show that our query scheme demonstrates good effectiveness and completeness.

Our contributions in this paper are as follows:

- We propose a decentralized semantic extraction (DSE) algorithm for extracting the semantic information of the off-chain data. The extracted semantic information and data block addressing structure are put into a meta data (MD) which is stored as a transaction on the chain. As a result, the on-chain data contain the semantic and location information of the corresponding off-chain data.
- We modify the on-chain data storage structure without changing the underlying database to provide a retrievable key value for each transaction stored on the chain, and build an index structure with the key values, thus enabling efficient semantic, range and fuzzy queries of on-chain data as well as good compatability with exsisting blockchain systems.
- We provide a verification object (VO) for the index instructure. With this VO, light nodes containing only block headers are able to verify query results returned from full nodes.
- We employ three data sets to test the performance of our scheme, covering the space and time overhead with the index structure added and the response time of four query types over the three data sets. A comparison with other blockchain query schemes is made, showing that our scheme enjoys better response time, lower query results verification cost and superior effectiveness and completeness.

In Section 2, related work is reviewed. In Section 3, we define the problems and the query mode. Our query scheme is described in Section 4, including the index structure and relevant query algorithms. In Section 5, we perform the experiments and analyze the experimental results by our scheme on three publicly available data sets in comparison with those by other schemes. Finally, we conclude this work and discuss our future researches in Section 6.

II. RELATED WORK

In this section, we introduce some related studies on blockchain storage schemes, query-related data structures, and some traditional index structures.

Blockchain Storage. The blockchain system does not store underlying raw data (typically transaction data in a certain format),but stores their hash values obtained by hash algorithms, generally not as plaintext. For example, Bitcoin uses a double sha256 hash function [14] to store original transaction records of arbitrary length into the block. Underlying most blockchain systems is the key-value database, which sacrifices read performance in exchange for improved write performance. Bitcoin and Ethereum adopt the levelDB database [2], while the Hyperledger Fabric uses a combination of the levelDB and couchDB [15]. Some efficient hybrid storage schemes have been proposed for better scalability of blockchain systems. The main approach is to store most of the data off the chain in a distributed peer-to-peer network (e.g. Bittorrent network [16] and Ipfs network [17]), with hash values of the entire data or partitioned data blocks on the blockchain. The on-chain stored hash values of the off-chain data ensure the consistency of on- and off-chain data, but contain no meaningful information such as the semantics of off-chain data.

Blockchain Structure. A blockchain is a data structure that is linked from one block to another. Each block contains two parts: the block header and the block body. And the block header contains the core data that determines the context of a block, called a hash pointer. Each block header has a previousBlockHash, which is a record of the hash value of the previous block. Due to the special nature of the chain structure, the previousBlockHash of a block header is considered to be related to all the previous blocks and contains time stamp information. This structure is believed to guarantee strong data security of the blockchain. However, its append-only attribute also means that a query of all the blockchain data requires traversing backward the entire chain to ensure that the search results are complete. Because this data structure is not friendly to light nodes containing only the block head, some blockchain systems have optimized the verification process of necessary data such as transactions. For example, based on the UTXO model, Bitcoin introduces a merkle hash tree (MHT) structure [1], which uses layers of hashing to ensure that the order of transactions in a block is not tampered with. The merkle root is stored in the block header. Light nodes can use the merkle root to verify the authenticity of transactions in the block. At its inception Bitcoin offers convenient authentication for light nodes, namely Simplified Payment Verification (SPV) [1]. To verify a transaction at a specific location within a block only requires the full node providing the transaction address and corresponding Merkle Proof. In fact, we only need $O(\log N)$ to the transaction within the time complexity of the location within the block is determined. Our query scheme uses a verification process similar to SPV which is detailed in Section 4. Unlike Bitcoin, Ethereum is an account-based model, known as a transaction-driven state machine [2]. Each block change can be considered as a state transition, which can be described as: $\sigma_{t+1} = \gamma(\sigma_t, T)$, where σ is the world state of Ethereum, γ is the state transition function of Ethereum, and T is a block. All Ethereum account states are stored as RLP codes in a state trie termed Merkle Patricia Trie (MPT)

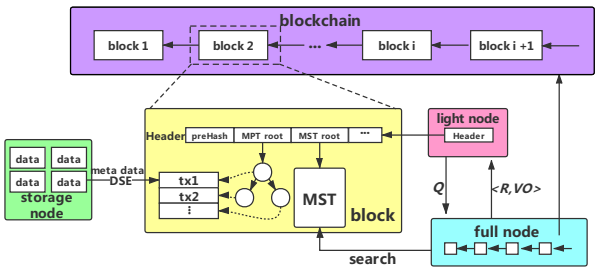


Fig. 1. System architecture.

[2]. Each MPT leaf node is an account state, while the branch stores the compressed prefix of the account address. The miner updates the states of the accounts in the MPT according to the transactions in the block. Therefore, the latest states of all accounts are stored in the latest block, with the root hash of the MPT in the block header.

Traditional Indexing Technologies. Full-text indexing [18] is the key technology for searching unstructured mass data, and it is also the core technology of search engines such as Google and Bing [19]. Unlike structured database systems, the mass data stored in the blockchain have no fixed structure, so the hash indexes [20] and B+ tree indexes [21] commonly employed in databases are not directly applicable in blockchain systems. Most full-text indexing technologies for unstructured big data are based on inverted indexes [22]. Inverted indexing technology is an inverse operation of data information and uses "keyword-document" mapping to support keyword queries of source data, which is the most effective index structure for modern information query. We adopt the inverted indexing for constructing the index structure in our scheme for hybrid storage blockchains, which is described in Section 4.1.

III. PROBLEM DEFINITION AND PRELIMINARIES

We propose in this paper a distributed query scheme based on indexing for the hybrid storage blockchain. The architecture of the system is illustrated in Figure 1. There are three types of nodes in our system: full nodes (accounting nodes), light nodes, and storage nodes. The storage nodes can also act in the system as accounting nodes. A light node may store only all the header information of the blockchain. A storage node needs to become a full node to participate in the consensus process. The query capabilities of our index structure enable fast retrieval, in which the light nodes only need to use their block headers to perform Simplified Retrieval Verification (SRV) on the search results provided by the full nodes to verify their correctness and completeness.

A. Query Models

Multi-keyword query is a common function of search engines, but existing blockchain systems do not support this function. It is our on-chain index structure that makes multi-keyword query possible on the blockchain. A typical keyword combination in multi-keyword query is in the form $< key1, key2, key3 >$ connected with connectives AND, OR, etc. It is sometimes referred to as a Boolean query. The user might want to search data containing several definite keywords or data containing one or more out of the keywords. For example,"$Blockchain$" $\wedge$ ("$Retrieval$" $\vee$ "$Search$") means the user wants to obtain results that contain Blockchain together with both Retrieval and Search or at least one of them. After optimizing the model, this article extends Boolean query to the Boolean model and unified vector model query. In our system, a user's input is a triplet $< \ searchtype, [Si, Sj], key1 \wedge (key2 \vee key3) \ >$, where $searchtype$ is the type of user query, such as semantic (keyword) query, range query and fuzzy query; $[Si, Sj]$ is the size range of the range query, which may be a time window or a space window, etc. The user's query statement may be "$all\ pdf > 20MB$" or "$0.99\$ < all\ prices < 1.99\$$", etc. The third part of the triplet is a conjunction normal form (CNF) composed of keywords.

B. Transaction Structure

Transaction is the most important way for us to map off-chain data to on-chain data. In the blockchain system, the processed transaction data is stored in the NoSQL database in the form of $< \ key, value \ >$ pairs. We add a meta data (MD) to the transaction to represent the location and semantic meta information of the data, thus establishing a mapping between the on-chain data and the off-chain data. With MD added, the general form of the transaction becomes tran = $< \ sender, recipient, amount, MD \ >$, a structure that includes the unique $marker_i$ of the data and a content abstract abs_i for the data, which contains the necessary semantic keyword information of the data. We define: MD = $< fhash, fname, datasize, abs >$,where $fhash$ and $fname$ constitutes a unique $marker_{md}$ for MD, satisfying: $\forall md, md' \in set\ MD, marker_{md} \neq marker_{md'}$.

C. Performance and Security

Due to the decentralized nature of the blockchain system, there is not a trusted node responsible for the unified administration of the entire system, so the completeness and correctness of the query are the major concerns of our scheme. A malicious miner may maliciously temper with the index on the chain, and the full node performing the query may also return incomplete or misleading results to the light node. The superiority of our system lies mainly in its query performance reflected in the following aspects:

- Response time: whether the system's time consumption is acceptable for various queries;
- Correctness: whether expected query results are returned;
- Completeness: whether all the query results are returned.

IV. METHOD

We focus primarily on real-time keyword query. As mentioned earlier, keyword query is realized mainly by full-text indexing. The cost of indexing mass data is very high. Therefore, we first extract the semantic information of the off-chain data, select a small number of highly representative

keywords according to the importance of the keywords, and finally construct an inverted index of these keywords and maintain a global search tree structure called merkle semantic trie (MST). We design an improved version of Aho-Corasick automaton [23] for keyword matching in the query process, realizing fuzzy query along with exact matching. We also introduce the B+ tree feature to MST to support range query.

The off-chain information extraction, on-chain indexing, and query and verification are detailed, followed by an optimization of the index structure for further support of range query.

A. Off-chain Information Extraction

First, we extract the semantic information of off-chain data to establish a mapping between off-chain data and on-chain data. We propose an improved TF-IDF model [13], which we call Decentralization Semantic Extraction (DSE). DSE improves the precision of semantic feature extraction by introducing decentralized word frequency, part-of-speech and position factors to address the lack of semantic understanding in traditional keyword extraction algorithms. On this basis, an inverted index of semantic keywords is constructed.

In the TF-IDF model, TF (Term Frequency) represents the frequency of occurrence of terms in the data. We use $TF_{i,j}$ as the frequency of keyword i in the data j, and the IDF (Inverse Document Frequency) indicates the importance of the entry, which is calculated by:

$$IDF_i = \log\left(\frac{N}{df_i + 1}\right), \tag{1}$$

where N is the total number of words, and df_i is the number of times the data containing the keyword i appears in N. In order to avoid the denominator being 0, Laplacian smoothing is used here to process df_i.

The traditional TF-IDF calculation result is the product of $TF_{i,j}$ and IDF_i, but TF-IDF can only calculate the weight of keywords by frequency. In off-chain data, the keywords have different contributions to data classification. We use the Decentralized Term Frequency Factor (DTFF) to decentralize the frequency of keywords by:

$$DTFF_{i,j} = 2^{N_{i,j} - N_i}, \tag{2}$$

where $N_{i,j}$ is the number of occurrences of keyword i in data j , and N_i is the average number of occurrences of keyword i in all data. The weight of exclusive words can be increased through the decentralization of DTFF.

Furthermore, we draw lessons from the scheme of speech classification priority based on statistical methods [24], which calculates the weight of part of speech by using a fuzzy hidden Markov model, using the method, we define the weights for keywords of different parts of speech as follows:

$$Pos_i = \begin{cases} 0.9 \text{ n.} \\ 0.8 \text{ v.} \\ 0.3 \text{ pron.} \\ 0.5 \text{ adj.} \\ 0.5 \text{ adv.} \\ 0.4 \text{ num.} \\ 1 \text{ idiom.} \end{cases} \tag{3}$$

On the other hand, the content of most data at the beginning and end are often of greater significance and should be given a higher weight. We arrange the positions where the keywords first appear in the data into a sequence with the number of words as the total length and 1 as the unit scale. The middle position of the sequence is set as the initial coordinate. The distances of other words from the initial coordinate are calculated. The longer the distance, the greater the weight. The Position Factor is defined as:

$$PF_{i,j} = \begin{cases} 1, & d < \delta \\ \varepsilon * \frac{d}{\delta}, & d \geq \delta \end{cases}, \tag{4}$$

where ε is an increased weight multiple, d is the distance of the word i from the initial coordinate, $\delta \in [0, \mathrm{L}/2)$ and L is the length of the sequence.

Then the weight of the feature word is:

$$W_{i,j} = TF_{i,j} * IDF_i * DTFF_{i,j} * Pos_i * PF_{i,j} \tag{5}$$

Subsequently, the miners maintain a global inverted index consisting of three parts: keyword, blockNumber, and W. Note that instead of using a data marker, we use BlockNumber here, because our index will eventually have a hash pointer to the block where the data is located, and BlockNumber is the major query content.

On-chain Index Structure. An index structure (IS) is established in the block body. This index structure is jointly maintained by the miners and is always up to date in the latest block. Since it does not depend on any specific underlying database, our index structure is compatible with various blockchain systems.

Index Structure Generation. In our scheme, the index structure is generated together with the genesis block, and changes as new blocks are added. We extract the semantic information of the off-chain data and construct an inverted index of keywords. This inverted index allows fast real-time query on a single keyword. In order to further support multiple keywords query, we construct a novel index structure called merkle semantic trie (MST) that integrates the inverted index, Merkle Tree, and hash pointer techniques. Prefix compression is employed to improve the efficiency of multiple keywords query. Due to the additional nature of the blockchain, the MST is dynamically constructed and updated every time a new block is generated.

There are three types of nodes in the MST: branch nodes, extension nodes, and leaf nodes. A node can only be of one type. Branch nodes only store index-related information and provide pointers to the next-level nodes; leaf nodes are at

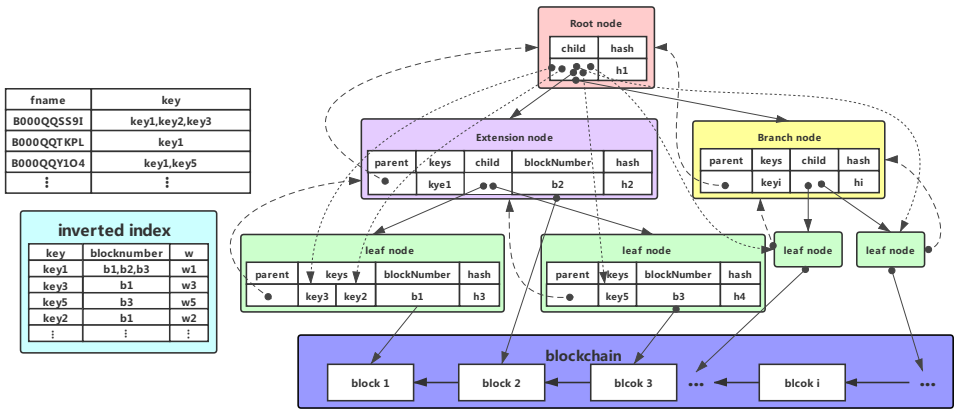


Fig. 2. MST structure.

Algorithm 1: MST Root Insert

Input : info as indexInfo(blockNumber and a list of keyword as $< key_m >$, m as the length of keyword list, $< key_m^n >$ as the combination C_m^n of $< key_m >$); InList as inverted index; R as the root node.

Output: MST mst

```
foreach key_m^i in < key_m^n > do
    Sort key_m^i by InList;
    true ← flag;
    forall child_j in < R.child > do
        if first key in key_m^i match first key in
         R.child_j.key_m^i then
            false ← flag;
            Insert <blocknumber, key_{m-1}^n > to
             R.child_i;
    if flag == true then
        Append new node of info into < R.child >;
return mst with R;
```

the bottom of the MST and represent the end of a search path. Query results are stored in leaf nodes. The result is a hash pointer to the block where corresponding transaction is located. An extension node contains not only a pointer to the next level, but also the query results. The root node is special, which serves as leaf node if there is only one node in the MST, and as branch node with child nodes present.

For the ordered keyword set in the inverted index, the miners will insert the $< blockNumber, \{key1, key2...\} >$ pairs into the MST in order of their weights W, and each time they need to be inserted from the root node by the following MST Root insertion algorithm.

Algorithm 1 shows inserting processing on the MST root. To explain this algorithm, we use an example of a data set to simulate an insert and query operation. Suppose the statement to be inserted is set_i : {"$blockchain$" $\wedge$ ("$query$" $\vee$ "$search$")}. The query may be {"$blockchain$" $\wedge$ "$query$"}or {"$search$" $\wedge$ "$blockchain$"}. This ambiguity can be interpreted as the sequence of keywords may be considered as different combinations of keywords. In our DSE model, this ambiguity is eliminated. We sort the input keyword combinations in the quertquery statement according to their weights with those of larger weights ranked ahead, thus preventing recursive insertion of the keyword combinations.

The storage of keywords in the MST borrows the common prefix concept of trie. Upon an insertion to the keyword list, the data with the same prefix keyword combination is saved to the same path. The Patricia Trie [2] ideas are also introduced to compress paths with only one child node in order to fully free up storage space, the specific structure of MST is shown in Figure 2.

Similar to a merkle tree, each node of the MST has a hash value. For a leaf node, this hash value is a hash operation on the keywords and the block pointers stored in the leaf node:

$$hash_{leaf} = \text{hash}(\sum_{l} key_l | \sum_{l} blocknumber_l) \quad (6)$$

For a branch node, this hash value is a hash operation performed on the hash value of the keyword and its child nodes:

$$hash_{branch} = \text{hash}(\sum_{b} key_b | \sum_{b} childhash_b) \quad (7)$$

For extension nodes, this hash value is a hash operation on the hash value of the keywords, block pointers, and child nodes:

$$hash_{ext} = \text{hash}(\sum_{e} key_e | \sum_{e} blocknumber_e | \sum_{e} childhash_e) \quad (8)$$

The parent node finds the child node by the hash value of the child node. The root hash value saved by the root node is stored in the MSTrootHash field of the block header.

In order to save storage space, we do not store the information related to the inverted index in the block, because the MST already contains all the keyword information. A newly added node can restore the inverted index locally by using the MST, or it can connect another full node, synchronize its inverted index, and then uses its own MST to verify.

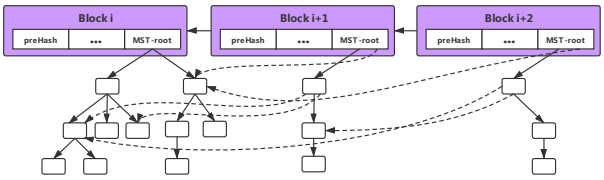


Fig. 3. Blockchain with added node pointers. The dotted lines represent the parent node pointers to the unchanged subtrees after the MST state changes.

Node pointer. Since MST contains all the index information on the blockchain, the index size increases as the index information is continuously added. But the miners in a block may only update part of the MST, for which we design the node pointer structure. If a part of the structure of the MST is not updated in a block, there is no need to save this part in the block. It suffices to add a node pointer to the parent node of the subtree of the part. The pointer points to the location of the last change in the subtree of this part changed. This method effectively reduces the storage cost within the block. A blockchain with added node pointers is shown in Figure 3.

B. Query Processing

After obtaining the user's search input, we need to match the keyword combination entered by the user with the index path in the MST, and return the matching result to the user. In the matching process, we used the Multiple keywords Aho-Corasick automaton (MKACA), which is similar to KMP [25]. But the public prefix in our system is not an identical string, but a set of identical keywords. First, we map the MST to the MKACA as shown in Figure 4. The double circles are the receiving states, which correspond to the extension nodes and leaf nodes in the MST. Our improvement to the MKACA here is that for a fuzzy query, if all keywords have been matched before the final state is reached, the state reached becomes a fuzzy receiving state, at which point the MKACA returns all the results after this state. Because ranking of the input keywords has been performed by the inverted index, no recursive matching occurs in the matching process. Before proposing a multi-keyword matching algorithm, we first propose the following related concepts:

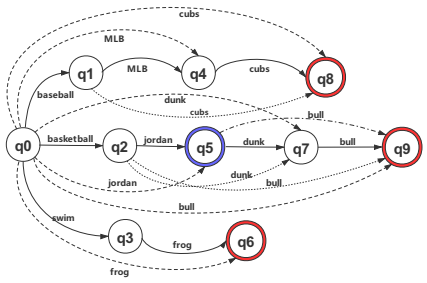


Fig. 4. Structure of Multiple keywords Aho-Corasick automaton.

Concept 1: The MKACA is a five-tuple $\mathrm{M} = (\mathrm{Q}, \Sigma, \delta, q_0, \mathrm{F})$.$\mathrm{Q} = \{q_1, q_2 \ldots q_m\}$ is a finite state set, corresponding to all nodes on the MST. Σ is a word list, i.e. the inverted index in our scheme. δ is the transfer function for transferring to next state at a new keyword input. q_0 is the MKACA initial state corresponding to the MST root node, while F is the halting state corresponding to the leaf nodes and extension nodes of the MST.

Concept 2: For the state $q_i \in \mathrm{Q}, \mathrm{pref}\,(q_i))$ represnets all the previous states, that is, $\mathrm{pref}\,(q_i) = \{q_j | \delta'\,(q_j, k_i) = q_i\}$, $\mathrm{suff}\,(q_i)$ represents the subsequent states of qi, that is, $\mathrm{suff}(q_i) = \{q_j | \delta'(q_i, k_i) = q_j\}$.

Algorithm 2: MKACA match

Input : MKACA $\mathrm{m}(\mathrm{Q}, \Sigma, \delta, q_0, \mathrm{F})$, MST mst, query list key= $< K >$
Output: result blockNumber $< R >$, VO

state be the initial state of m;
state $\leftarrow q_0$*;*
foreach k_i *in* $< K >$ **do**
  if δ (*state* , k_i) $\in$ Q **then**
    state $\leftarrow \delta$ (*state* , k_i)*;*
if *state* $\in F$ **then**
  node $\leftarrow$ *MKACAToMSTInSamePos(mst,m,state);*
  Add all node.value to R;
  Cerate merkleProofList;
  while *node is not the rootNode in mst* **do**
    pathNode $\leftarrow$ *node;*
    node $\leftarrow$ *node.parent;*
    foreach $node.child_i$ *except pathNode* **do**
      Add $< nodeInfo, nodeLevel >$ *to MerkleProofList;*
  VO $\leftarrow$ *merkleProofList+mst.rootHash;*
  return $< R, VO >$*;*
else
  return NULL

MKACA matching process is shown in Algorithm 2, the MKACA match algorithm, we get the keyword query result R and its verification object (VO), followed by the verification of the results. Firstly, the query results need to be verified by SRV, and the corresponding MSTrootHash of the query results can be calculated by using the results and VO, which is then compared with MSTrootHash in the local blockheader. A match indicates that the full node returns legitimate query results.

The SRV verification process is $SRV(R, VO) \rightarrow \{0, 1\}$. First, a cascade hash operation $\mathrm{hash}\,(\sum_i R | Node_R)$ is performed on the node information of R provided in the query result R and VO to obtain the hash result h(R). Then a recursive hash operation SRVresult = $hash(hash(hash(h(R)|\sum_i p(level_r^i))|\sum_i p(level_{r-1}^i))\cdots|\sum_i p(level_1^i))$ is performed on h(R) and the merkle proof provided by VO $\{\ p(level_1),\ p(level_2), \cdots p(level_r)\ \}$. Finally, the SRVresult is compared with the MSTrootHash in the local block header of the light node. If they are the same, the verification passes, and 1 is returned; otherwise 0 is returned.

C. Range Query Support

By adding the MST, our system already supports multi-keyword query and fuzzy query. However native support of range query is not available owing to its underlying levelDB database, which uses the Log Structured-Merge Tree (LSM) at the bottom. Because the MST index structure requires query from the root node to the leaf node, we introduce the B+ tree to reorder the MST nodes without modifying the underlying storage mode, so that each node has a B+ tree range key value while corresponds to a keyword key value. The MST with the B+ tree added is shown in Figure 5.

Different from the traditional B+ tree, our MST-B+ tree structure is strongly keyword-data size coupled and there is no need for frequent IO operations on the disk. The miners only need to sort for once the range of the file corresponding to the same keyword on each path.

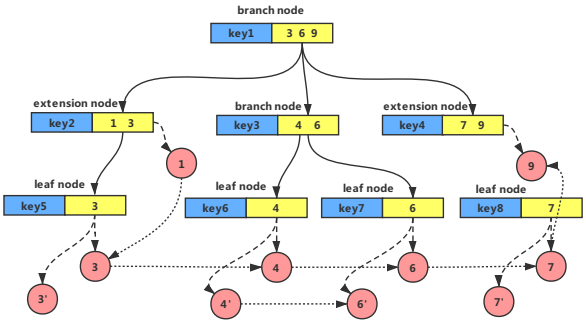


Fig. 5. MST-B+ tree structure. The chains 1 to 9 are the ordered B+ tree node chains for key1. 4 ' and 6 ' are the ordered B+ tree node chains for key3.

V. EXPERIMENT

In this section, we tested the performance of our query scheme with three different datasets in comparison with that of other blockchain query techniques.

Our experiments were performed using Ethereum's golang client geth1.8 on a 64-bit Linux server (Ubuntu 16.04) with Intel core i7-7700 CPU and 16GB of memory.

We first tested the time and storage cost of indexing. The response times for four of query types on the full nodes and the light nodes respectively are tested with the three data sets. Then we compared the query and retrieval performance before and after the inverted index and MST are added. The retrieval performance of our system was tested with different numbers of keywords, and the impact of the DSE algorithm parameters on the query relevance was also tested. Finally, a comparison of overall performance is made between our schemes and two blockchain proxy query schemes.

A. Dataset Description

- Amazon Johnson Music label data set (AM)
  AM is an Amazon-powered public data set of blues music meta information comprising approximately 305KB of meta data and 35MB of corresponding music information data. Its meta data use the json format.
- Steam bundle dataset (SB)
  SB is a user line data set on the game platform Steam, consisting of user-id, Game-title, bundle-name and value. We employ this data set for the range query test.
- Traffic data set (TAXI)
  TAXI is a public data set provided by the University of Hong Kong, containing 24-hour driving data of 4,000 taxis in the four cities of Rome, Shanghai, Bologna and Cologne. The data format is $< carID, time, longitude\&latitude, speed, whethercarrypassenger >$.

The sources of data of our system are not limited to blockchain-related transactions. With the development of smart cities, the blockchain in the future may also host data from edge IoT devices [26]. Therefore, the future data sharing requirements are taken into consideration in our experiments.

B. Indexing Cost

1) Time Cost: Unlike conventional blockchains, the most time-consuming part in our scheme is the generation of the index structure MST. The time cost can be expressed by:

$$Cost_{index}(\Delta t) = Cost_{insert}(\Delta t) + Cost_{update}(\Delta t) \quad (9)$$

In the time interval Δt, indexing time cost consists of the time spent on index insert operations and that on index updates. The index updates time cost comprises the time spent on MST updates and that on inverted index updates.

2) Storage Cost: Since the transaction volume may differ from block to block, we tested the space occupied by the blocks under different transaction data sizes.

In order to determine the impact on block generation of adding the MST index, the time and storage costs before and after the addition of the index were tested respectively. The results are shown in Figures 6 and 7.

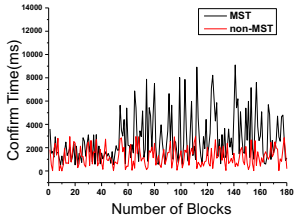


Fig. 6. Time for generating blocks with and without the MST.

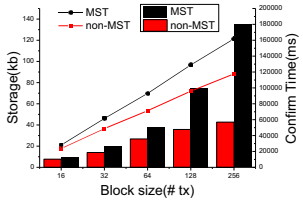


Fig. 7. Average storage cost and time for generating blocks with and without the MST.

Figure 6 shows the time for generating a block with and without the index under different numbers of blocks. The results show that the addition of the MST leads to an increase of about 0.5 seconds in the average block generation time, which is acceptable. Figure 7 shows the average size of blocks containing different numbers of transactions. It can be seen that there is a slight increase in storage in non-transaction-intensive blocks due to the addition of MST. In view of the

TABLE I
RESPONSE TIMES FOR FOUR QUERY TYPES.

Query approach	Basic query						Semantic (keyword) query					
dataset	AM		SB		taxi		AM		SB		taxi	
node type	light node	full node	light node	full node	light node	full node	light node	full node	light node	full node	light node	full node
Mean(ms)	2	2	2	2	2	2	5	4	6	5	-	-
Worse case(ms)	4	3	4	3	4	3	106	33	25	23	-	-
Query approach	Range query						Fuzzy query					
dataset	AM		SB		taxi		AM		SB		taxi	
node type	light node	full node	light node	full node	light node	full node	light node	light node	full node	light node	light node	full node
Mean(ms)	-	-	19	17	21	20	11	10	13	11	13	12
Worse case(ms)	-	-	156	145	273	251	29	26	33	29	42	38

current throughput of Ethereum, no significant increase in the average block size will result from adding the MST.

C. Query Performance

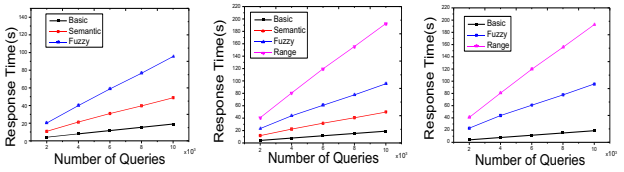
The effectiveness of the system is mainly tested by the system's response time, precision, and recall, which correspond to the efficiency, correctness, and completeness we mentioned earlier.

1) Response Time: In the response time experiments, we tested the average query time and worst case query time for four query types (basic query, semantic keyword query, fuzzy query, range query) under 3 different data sets (AM, SB, TAXI). The basic query covers getBalanceByAddress, getBlockByHash, and getTransactionByHash of Ethereum. Since the light node adopts the method of proxy query, it needs to communicate with the full node on the network and verify the query results. So the light node and the full node are tested separately in the response time experiments.

TABLE II
AVERAGE VERIFICATION TIME OF LIGHT NODES FOR SEMANTIC (KEYWORD) QUERY, RANGE QUERY, AND FUZZY QUERY.

Query approach	Semantic (keyword)	Range	Fuzzy
Mean(ms)	0.656	1.613	1.103
Worse case(ms)	8.791	13.263	3.389

From Table I, we can see that the average response time of our scheme is within 20ms for all four query types. It can be found in Table II that in the case of a proxy query, the query result verification time of the light node is quite short, which suggest good verification efficiency of our proxy query mechanism. We obtained the cumulative response times on three datasets for different queries under different query sizes, and the statistical results are shown in Figure 8.

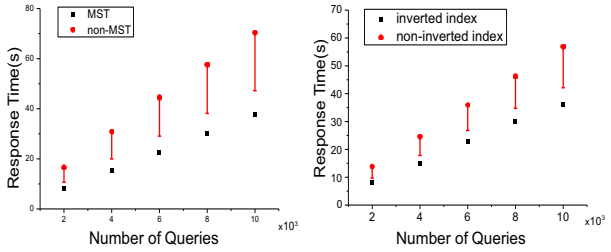


(a) AM (b) SB (c) TAXI

Fig. 8. Response times on three different data sets.

2) MST and Inverted Index Performance: To verify the significance of the MST and the inverted index in our scheme, we tested the system keyword query performance on the data set AM by the variable control method. The keyword query response time was tested with and without the MST respectively. Since Ethereum does not natively support keyword query, we wrote a smart contract to test Ethereum's keyword query performance without the MST by taking a circular scan. A second experiment was performed to test cumulative response time under different numbers of queries with and without the inverted index. If there is no inverted index, it is necessary to randomly sort the keywords extracted from the off-chain data by the DSE algorithm, and thus the keyword combinations inserted into the MST need also to be fully permuted.

The results are shown in Figure 9 . We can see in the first chart that after adding the MST structure, the average query time drops by more than 50% compared to that of the simple Ethereum. The second chart shows that in the absence of an inverted index, the MST itself offers only a limited query performance improvement. Therefore, for optimal query performance, both the inverted index and the MST need to function properly.



(a) Response times with and without MST. (b) Response times with and without inverted index.

Fig. 9. Results of MST and Inverted Index Performance in response time.

3) Precision, Recall, and Ranking Performance : To test the correctness and completeness of our scheme, we use precision to represent correctness and use recall to represent completeness, additionally, we use mean reciprocal rank(MRR) to test the ranking performance of DSE algorithm. We first tested the query precision and recall with 1 to 5 keywords. To trade off precision and recall, we use the harmonic average of the two,

namely the F value, which is calculated by:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-a)\frac{1}{R}}, \tag{10}$$

where P represents the precision, R represents the recall, and α is the proportion of P and R. Here, we set α to 0.5 because we deem the precision and the recall equally important. The test results are given in Table III.

TABLE III
PRECISIONS, RECALLS, AND F-VALUES AT DIFFERENT NUMBERS OF KEYWORDS.

Keywords	P	R	F
1	0.733	0.549	0.627
2	0.809	0.588	0.681
3	0.863	0.665	0.751
4	0.889	0.773	0.826
5	**0.909***	**0.802***	**0.852***

TABLE IV
PRECISIONS, RECALLS, AND F-VALUES WITH AND WITHOUT THE INVERTED INDEX.

	P	R	F
inverted list	0.815	0.640	0.717
non-inverted list	0.615	0.261	0.366

From the results in Table III, we can find that with the increase in the number of keywords, the precision improves by gradually smaller margins, while the recall also increases first by rising margins and then by decreasing margins. In general, both precision and recall of the system improves as the number of keywords increases. It is worth mentioning that results in Table IV show that the removal of the inverted list from the system leads to a drop in precision and particularly in recall.

Subsequently, we used the correlation metric MRR to test the performance of our DSE algorithm on keyword ranking. MRR represents the mean value of the inverse ranking of multiple queries, which is is defined as:

$$MRR = \frac{1}{|N|}\sum_{\mathrm{i}=1}^{|N|}\frac{1}{rank_i}, \tag{11}$$

where $rank_i$ represents the ranking of first related document of the ith query.

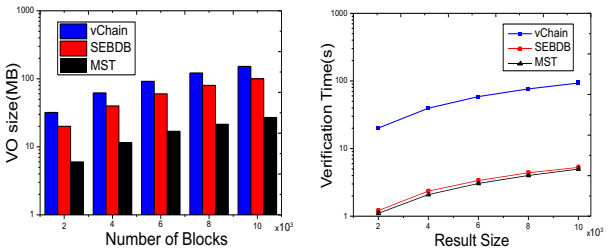
We selected two cases of ϵ = 1 and ϵ =2 for $\delta = L/2^n (n = 1, 2, 3, 4)$ four different values for MRR statistics.The statistical results are shown in Table V.

According to Table 5, as δ gets farther away from the initial coordinate, the keyword ranking performance of DSE first rises and then decreases, while peaking at $\delta = L/8$.

Additionally, it is worth noting that false positives are a problem that cannot be ignored, the equation of the probability

TABLE V
EFFECT OF δ VALUE ON KEYWORDS RANKING PERFORMANCE OF DSE ALGORITHM WHEN ϵ EQUALS 1 AND 2, RESPECTIVELY.

$n(\delta = L/2^n)$	ϵ@1 MRR	ϵ@2 MRR
1	0.502	0.502
2	0.611	0.636
3	0.677	0.692
4	0.652	0.661



(a) VO Size (b) Verification time

Fig. 10. Overall Perfomance of our MST system compared to those of vChain and SEBDB.

of a false positive f in the MST-based search method is defined as follows:

$$f \approx \left(1 - e^{-np/q}\right)^n, \tag{12}$$

where p is the the number of layers of MST, q and n are the number of keywords and hash functions separately.

Overall Perfomance. The retrieval performance of our scheme was tested in comparison with vChain [12] and SEBDB [27]. All three schemes support proxy query and non-proxy query, and provide a verification object (VO) for query results. We focus on the proxy query in the experiment with 18,000 keyword queries and range queries performed on the data set SB. We tested the storage space occupied by VO under different numbers of blocks and the cumulative verification time for VO for different numbers of search results, and then compared the VO size and VO verification time of the three schemes. The test results are shown in Figure 10 .

The VO size of our scheme is approximately 1/5 that of vChain and about 1/3 that of SEBDB. The verification time of our scheme is significantly reduced compared with that of vChain and slightly less than that of SEBDB. It can be concluded that for keyword query, our scheme promises smaller VO size and less verification time for VO than vChain and SEBDB .

VI. CONCLUSION

In this paper, we realize for the first time a mapping between on-chain and off-chain data by indexing for efficient query on the blockchain. We propose a new semantic keyword extraction algorithm for extracting the semantic keywords from off-chain data, which are ranked in accordance with their importance to build an inverted index. Based on this index, we

design an on-chain index structure called merkle semantic trie (MST). The MST provides keyword query, range query, and fuzzy query capabilities with its own prefix matching feature coupled with MKACA and B+ trees and is readily applicable to various types of blockchain underlying databases. The proposed scheme is implemented and the system performance is tested. Comparison of response times for four query types with available blockchain retrieval schemes shows that our scheme offers better query performance.

Having realized efficient query for hybrid storage blockchains, we will turn to the optimization of data query capabilities and performance. Deep learning, the convolutional neural network (CNN) and the deep trust model will be employed to process on-chain and off-line data for complex data processing and retrieval, ultimately realizing fully automatic data analysis.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program of China under Grant 2018YFE0126000, the Key Program of NSFC-Tongyong Union Foundation under Grant U1636209, the National Natural Science Foundation of China under Grant 61902292, and the Key Research and Development Programs of Shaanxi under Grant 2019ZDLGY13-07 and 2019ZDLGY13-04.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin. org/bitcoin. pdf," 2008.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[3] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang, "Fine-grained, secure and efficient data provenance on blockchain systems," *Proceedings of the VLDB Endowment*, vol. 12, no. 9, pp. 975–988, 2019.

[4] K. Francisco and D. Swanson, "The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency," *Logistics*, vol. 2, no. 1, p. 2, 2018.

[5] A. Dell'Anna, "OHLCV, social and blockchain cryptocurrency data analysis and price forecasting," Ph.D. dissertation, Politecnico di Torino, 2019.

[6] J. Zhou, F. Tang, H. Zhu, N. Nan, and Z. Zhou, "Distributed data vending on blockchain," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1100–1107.

[7] J. Eberhardt and J. Heiss, "Off-chaining models and approaches to off-chain computations," in *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, 2018, pp. 7–12.

[8] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, "Bigchaindb: a scalable blockchain database," *white paper, BigChainDB*, 2016.

[9] 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*. Springer, 2017, pp. 556–567.

[10] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, and R. Ramamurthy, "BlockchainDB: a shared database on blockchains," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1597–1609, 2019.

[11] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1085–1100.

[12] 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*, 2019, pp. 141–158.

[13] J. H. Paik, "A novel TF-IDF weighting scheme for effective ranking," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, 2013, pp. 343–352.

[14] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," 2014.

[15] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–15.

[16] S. J. Arulmozhi, K. Praveenkumar, and G. Vinayagamoorthi, "BITCOIN IN INDIA: A DEEP DOWN SUMMARY," *Advance and Innovative Research*, p. 28, 2019.

[17] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[18] H. Bast and B. Buchhold, "An index for efficient semantic full-text search," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 369–378.

[19] A. Van den Bosch, T. Bogers, and M. De Kunder, "Estimating search engine index size variability: a 9-year longitudinal study," *Scientometrics*, vol. 107, no. 2, pp. 839–856, 2016.

[20] G. R. Mitchell and M. E. Houdek, "Hash index table hash generator apparatus," Jul. 29 1980, uS Patent 4,215,402.

[21] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006, pp. 121–132.

[22] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel, "Compression of inverted indexes for fast query evaluation," in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, 2002, pp. 222–229.

[23] S. Dori and G. M. Landau, "Construction of Aho Corasick automaton in linear time for integer alphabets," *Information Processing Letters*, vol. 98, no. 2, pp. 66–72, 2006.

[24] I. N. Yulita, H. L. The, and adiwijaya, "Fuzzy Hidden Markov Models for Indonesian Speech Classification," *Journal of Advanced Computational Intelligence & Intelligent Informatics*, vol. 16, no. 3, pp. 381–387.

[25] R. Baeza-Yates and G. H. Gonnet, "A new approach to text searching," *Communications of the ACM*, vol. 35, no. 10, pp. 74–82, 1992.

[26] J. Feng, F. R. Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative Computation Offloading and Resource Allocation for Blockchain-Enabled Mobile Edge Computing: A Deep Reinforcement Learning Approach," *IEEE Internet of Things Journal*, 2019.

[27] Y. Zhu, Z. Zhang, C. Jin, A. Zhou, and Y. Yan, "SEBDB: Semantics empowered blockchain database," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1820–1831.