*sustainability*

MDPI

*Article*

# Implementation of a Smart Contract on a Consortium Blockchain for IoT Applications

**Ting Lin, Ziyi Huan, Yongcan Shi and Xu Yang ***

School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China; linting@bit.edu.cn (T.L.); 3220200891@bit.edu.cn (Z.H.); 2220170681@bit.edu.cn (Y.S.)
* Correspondence: yangxu@tsinghua.edu.cn

**Abstract:** Advancements in cryptography and computer science have given birth to blockchain technology. One of the most exciting evolutions of blockchain is the advancements in smart contract technology. Smart contracts can be used for a broad range of use cases, not just financial transactions. Smart contract technology on the public blockchain, represented by Ethereum, due to its public and opaque nature, is not a good choice for many scenarios that do not require full disclosure, such as many IoT applications. On the other hand, the existing blockchain smart contract system still has a strong connection with virtual currency, which also limits its application in non-financial scenarios. In order to solve the above problems and explore more of the potential of smart contracts for the IoT application domain, this paper mainly explores the construction of a smart contract system based on consortium blockchains associated with no virtual currency. Based on the smart contract system designed in this project, blockchain can be more easily applied in payment, product traceability, authority authentication, and other fields. Through a certain centralized way, the system is easier to manage, can reduce the management expenditure, and the power and other resource consumption is less, which is conducive to environmental protection. Results show that our smart contract system has the potential for IoT usage in the future.

**Keywords:** blockchain; smart contract; IoT



**Citation:** Lin, T.; Huan, Z.; Shi, Y.; Yang, X. Implementation of a Smart Contract on a Consortium Blockchain for IoT Applications. *Sustainability* **2022**, *14*, 3921. https://doi.org/10.3390/su14073921

Academic Editor: Ashutosh Tiwari

Received: 23 February 2022
Accepted: 21 March 2022
Published: 26 March 2022



## 1. Introduction

Transactions on the Internet require a trusted third party due to the digitization of currency and the opaque information of both parties to the transaction. However, the weakness of relying too much on the intermediary's credit led to the great restriction in non-rollback services and daily micropayments. Additionally, the problem of increased transaction costs and leakage of personal information have appeared.

In 1991, Stuart Haber and W. Scott Stornetta proposed an encrypted protection chain product [1,2]. After that, Ross J. Anderson and Bruce Schneier and John Kelsey were published in 1996 [3] and 1998 [4] respectively.

In 1998, Nick Szabo proposed Bit Gold. Bit gold is a theoretical solution for the decentralization of electronic money. In 2000, Stefan Konst proposed a complete set of implementation plans for the encryption protection chain.

In 2008, a researcher with the pseudonym "Satoshi Nakamoto" published a paper describing the design of Bitcoin [5]. Bitcoin can realize the confidentiality of its owner's information through the random hashing of digital signatures. However, at the same time, it brings the problem that it is difficult for the transaction payee to check whether the owner has made multiple payments for the electronic currency. For this reason, a "timestamp server" is introduced, which confirms that specific data must exist at a certain time through a timestamp. Through special incentive measures, a certain amount of virtual currency rewards are provided to the nodes that generate blocks, thereby ensuring the vitality of the entire system. By using a Merkle tree [6] and discarding past data, old blocks are compressed and hard disk space is reclaimed.

In 2009, the digital currency "Bitcoin" was designed. The original intention of Bitcoin [5] is to realize a point-to-point electronic cash system, which realizes decentralized transactions and transaction proofs by publishing transaction information, timestamps, and incentives.

Smart contracts are any contracts that have been pre-programmed with a set of definitive rules and regulations that are self-executing, without the need of any intermediaries [7,8]. We could treat smart contracts in the blockchain as the rough equivalent of an Application Programming Interface (API) in a traditional web environment. The smart contract is what connects the decentralized blockchain database to the front-end application.

The concept of smart contracts is proposed by Nick Szabo—a legend in the field of computer science and cryptography—in 1994. He defined a smart contract as a set of conventions defined in digital form, including agreements on which contract participants can enforce these conventions. The basic idea of a smart contract is that a variety of contract terms can be embedded into the hardware and software we use, making it extremely hard for attackers to attack. His work laid the foundations for smart contract technology; a software program that appends layers of information onto digital transactions via the blockchain. Smart contracts are self-executing contracts that operate on an if–then premise, enabling transactions to be complete once the terms of the contract are met. The terms of the contracts are coded directly into the smart contract.

Ethereum is one of the earliest and most popular blockchain projects that is built specifically to support smart contract functionality. Since then, there are a wide variety of projects that focuses on implementing smart contract technology such as NEO, Lisk, and Waves. Smart contracts allow for much more complex transactions than just the exchange of digital currency for services or products. They can execute many other functions as well.

Smart contracts can be used for a broad range of use cases, not just financial transactions [9]. A smart contract can execute a financial or contractual agreement between two parties, or it can simply trigger the execution of functions in a blockchain-based application.

M. N. Islam et al. proposed a method to eliminate the privacy threat of remote information processing devices in smart homes that support the IoT through blockchain-based smart contracts [10]. Christopher K. Frantz et al. proposed a modeling approach that supports the semiautomatic conversion of general-readable contract representations to computational equivalence contracts in order to encode rules into verifiable and executable computing structures that reside within a blockchain [11]. They also proposed mapping using domain-specific language operations to support the contract modeling process and explored this capability based on selected examples. Z. Gao et al. proposed an extensible smart contract execution scheme that can run multiple smart contracts in parallel to improve system throughput [12]. They also used the data of the existing smart contract system for experiments to evaluate the scheme.

With the development of Ethereum, more and more problems of its smart contract have been exposed [13], known vulnerabilities include race conditions, under/overflow, transaction order assumptions, dependency on timestamps, short address attacks, denial of service attacks, etc.

Smart contract technology on the public blockchain, represented by Ethereum, due to its public and opaque nature, is not a good choice for many scenarios that do not require full disclosure. On the other hand, the existing blockchain smart contract system still has a strong connection with virtual currency, which also limits its application in non-financial scenarios.

In order to solve the above problems, this paper mainly explores the construction of a smart contract system on a non-public blockchain associated with no virtual currency. We propose our effort at the implementation of a smart contract on consortium blockchains for future potential IoT applications.

## 2. Materials and Methods

### *2.1. Literature Review*

#### 2.1.1. Blockchain Framework

Advancements of cryptography and computer science have given birth to the blockchain technology [14–16]. It is fast gaining traction as a revolutionary innovation with the potential of disrupting current systems and a wide range of industries. Actually, blockchain is applied to more and more fields, such as healthcare, energy, national and territorial development [17], etc.

Ayesha Shahnaz et al. [18] have presented a framework that could be used for the implementation of blockchain technology in the healthcare sector for electronic health records. The framework they proposed can provide secure storage of electronic records by defining granular access rules for the users. Al Omar et al. [19] use blockchain as storage to attain the privacy of patients. Meanwhile, pseudonymity is ensured by using cryptographic functions to protect patients' data. In their work, a peer-to-peer (P2P) network which enables the property of decentralization is used to keep the sensitive health data private.

Esther Mengelkamp et al. [20] used a private blockchain, which underlines the decentralized nature of local energy markets to present a market design and simulation of a local energy market between 100 residential households. The technology proposed can trade local energy generation without the need for a central intermediary. Zhetao Li et al. [21] have proposed a secure energy trading system using the consortium blockchain technology, which can be widely used in general scenarios of P2P energy trading getting rid of a trusted intermediary. They also designed an optimal pricing strategy using the Stackelberg game for credit-based loans, which supports fast and frequent energy trading.

#### 2.1.2. Smart Contract for IoT

The advancement of smart contract technology is one of the most exciting evolutions of blockchain. There are more and more reports about using smart contracts based on blockchain in the literature recently.

Vinayak Singla et al. have developed a smart contract architecture for a leave management system using Solidity and Ethereum and included smartphones as IoT devices to use the app [22]. Their smart contract app is coupled with an alternative centralized architecture, which is a classic client/server architecture with an underlying blockchain backend. A. S. Omar et al. presents a decentralized identity management framework to implement a smartphone anti-counterfeiting system, which eliminates the need for a central authority and provides the features of identity creation and transfer of ownership. It also has the ability to report stolen and lost devices quickly and securely which takes effect in the shortest time. This framework uses a set of solidity smart contracts deployed on a private Ethereum blockchain [23].

T. Q. Nguyen et al. proposed a smart contract built on the NEO blockchain that applies to weather-based index insurance. They have designed five functions that can be triggered if certain conditions are met. They also built a virtual oracles server [24]. D. Naothu et al. introduced a secure smart surveillance system based on microservice architecture and blockchain technology, where the video analysis algorithms are regarded as various independent microservices [25]. Blockchain technology securely synchronizes the video analysis databases among microservices across surveillance domains, and provides tamper-proof of data in the trustless network environment. Smart contracts enable access authorization strategy to prevent any unauthorized user from accessing the microservices. Additionally, a scalable, decentralized, and fine-grained access control solution for smart surveillance systems is provided by it.

R. Cheng et al. proposed Ekiden, a system designed to solve the problem of lack of confidentiality and poor performance of blockchains by combining blockchains with Trusted Execution Environments (TEEs) [26]. Ekiden leverages a novel architecture that

separates consensus from execution and achieves efficient TEE-backed confidentiality-preserving smart contracts and high scalability.

Saquib et al. [27] proposed a blockchain-independent smart contract infrastructure suitable for resource-constrained IoT devices. Dustdar et al. [28] proposed to apply their concept of elasticity to smart contracts and thus enable analytics in and between multiple blockchains in the context of IoT. They also propose a reference architecture for elastic smart contracts and evaluate the approach in a smart city scenario, discussing the benefits in terms of performance and self-adaptability of the solution. Wickstrom et al. [29] propose to utilize smart contracts on the Ethereum blockchain to enforce a security model that helps maintain distributed IoT networks in a healthy condition throughout their lifecycle.

However, according to our investigation, almost all existing smart contract systems are based on public blockchains [30,31]. Compared with the public blockchain, consortium blockchains [32,33] reduce the number of nodes that could produce block through some degree of centralization, with less complex structure and faster block-producing speed. Compared with the private blockchain, consortium blockchains ensure the fairness of the system and increase the availability of the system by allowing some nodes to participate in block producing process.

In this paper, we introduced our contribution to building a smart contract system based on consortium blockchains.

### *2.2. Implementation of Smart Contract System Based on Consortium Blockchains*

#### 2.2.1. Address/Account

In order to ensure the security and consistency of data, the write permission of blockchain is restricted by account and consensus system. Account and address need to have user identity authentication, uniqueness confirmation, digital signature, and other functions.

The account consists of three elements:

- Account information: such as account name, email, and so on;
- Public key: used for signature verification [34];
- Private key: maintained by users themselves.

The most important part of an account system is the asymmetric encryption algorithm. Good asymmetric encryption algorithms have to satisfy the following features:

- Confidentiality: ensure the privacy of private data.
- Authentication: ensure the identity of all parties attempted to access.
- Authorization: ensure that a party attempting to perform a function has the right to do so.
- Data integrity: ensure that objects are not illegally changed.
- Non-repudiation: ensure that one party rejects the data or communication they initiate.

RSA and ECC are the most popular asymmetric encryption algorithms. In order to implement the account system in this paper, considering the guarantee of identity authentication and transaction tamper-proof in the distributed system, the account system should design one or more asymmetric key signatures. When choosing an asymmetric key algorithm, the key generation, performance, network, and storage resource overhead should be considered synthetically.

We compare all aspects of the two asymmetric key signature algorithms under the same difficulty. The comparative results are shown in Table 1.

We can see from the comparison that ECC has some advantages over RSA in key length and key generation time. Because the key length of ECC is relatively short, the network bandwidth and storage space required by ECC are also relatively small. However, the RSA algorithm is faster than ECC in signature verification time, which has a certain advantage in scenarios where a large number of transactions need to be verified, and the RSA algorithm is simple and relatively mature.

Therefore, the account system in this paper adopts a combination of RSA and ECC in the implementation, so as to be flexible. Users can specify whether the public key type is

RSA or ECC by setting different public key header identification fields. When the system receives the transaction signed by the user, the verification algorithm is identified by the identification field.

**Table 1.** Comparison of RSA and ECC.

| Key Length (bits) | | Key Generation Time (s) | | Signature Generation Time (s) | | Signature Verification Time (s) | |
|---|---|---|---|---|---|---|---|
| **RSA** | **ECC** | **RSA** | **ECC** | **RSA** | **ECC** | **RSA** | **ECC** |
| 1024 | 163 | 0.16 | 0.08 | 0.01 | 0.15 | 0.01 | 0.23 |
| 2240 | 233 | 7.47 | 0.18 | 0.15 | 0.34 | 0.01 | 0.51 |
| 3072 | 283 | 9.80 | 0.27 | 0.21 | 0.59 | 0.01 | 0.86 |
| 7680 | 409 | 133.90 | 0.64 | 1.53 | 1.18 | 0.01 | 1.80 |
| 15,360 | 571 | 679.06 | 1.44 | 9.20 | 3.07 | 0.03 | 4.53 |

#### 2.2.2. Network Structure

Since the transaction discovery and block proposal of blockchain are all realized through broadcasting, it would lead to a large number of redundant packets in the network, resulting in great pressure for the network layer system. In this paper, message frames are used to implement the blockchain network system protocol.

The network layer in this paper uses a P2P structure to reduce the risk of data loss and data tampering by reducing nodes in network transmission. Different from the network structure with a central server, the nodes in the peer-to-peer network are not only the clients but also perform some functions of the server, such as service discovery, message forwarding, and so on. Any node can not directly find other nodes beside its adjacent nodes after connecting to the network. It must rely on the node group in the network for information exchange.

The blockchain in this paper uses fully distributed unstructured topological networks. The network structure adopts the way of a random graph, while the degree of node generally obeys the power-law rule [35,36]. Fully distributed unstructured topological networks [37] have high fault tolerance and adaptability to the dynamic changes of nodes in the network. Therefore, the flexibility is high.

In the network, each node is responsible for processing the node discovery request of the adjacent node. Every node pushes its own known node to the adjacent node and obtains the new node through the node discovery service provided by the adjacent node.

In the transmission layer, the TCP/IP protocol is used for data transmission. A flooding protocol is adopted based on the TCP/IP protocol. The basic idea of flooding protocol is to deliver data packets to all possible paths except the path where data comes when each node receives new data, and to discard the packet if redundant data is received. The data transmission process is shown in Figure 1. When a piece of new data arrives, node A broadcasts to neighboring nodes B and C through flooding. When B and C receive the data, they continue to broadcast to neighboring nodes D, E, F, and G (because the data is lazily sourced, it is no longer broadcast to A). Then on the way to continue broadcasting, since nodes E and G are adjacent nodes, E and G broadcast the data to each other. However, since E and G have previously received the data from B and C respectively, the broadcast data between the two nodes this time will be discarded and the propagation will be terminated, ensuring that it will not continue to propagate to the B and C nodes.

We have divided nodes in our system into three kinds:

- Central node: After initialization, all nodes would be connected to the central node, which would also partially be in charge of transaction verification.
- Miner node: These nodes will store new transactions and generate new blocks when the transaction reaches a certain amount.
- User node: These nodes are specifically responsible for generating transactions and proposing them to the miner node. These nodes serve as the interface layer between the user and the blockchain system.

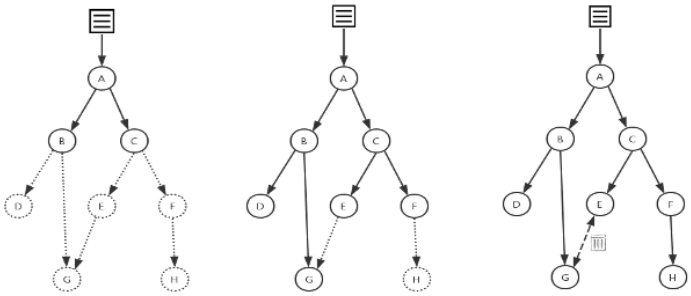

**Figure 1.** Data transmission and discarding under flooding protocol.

The packages used in our network system are presented below in detail.

Version Package

Version package is mainly used for node discovery and data synchronization. If a new node is added to the network, it will forward its version information to the node of its initial connection, which is the central node, according to our design. In this system, the structure of the Version package is shown in Table 2:

**Table 2.** Structure of Version package.

| Field | Meaning |
|---|---|
| Height | Length of the blockchain of the Version package sender |
| AddrFrom | Address of the Version package sender |

When the receiver receives a Version package, it records the sender's address and returns a Version package regardless of the content of that Version package. The returned Version package contains its own blockchain length and its own address. The Version package is equivalent to a two-time handshake protocol, which guarantees the synchronization of mutual discovery and information consistency between nodes. The process is shown in Figure 2:

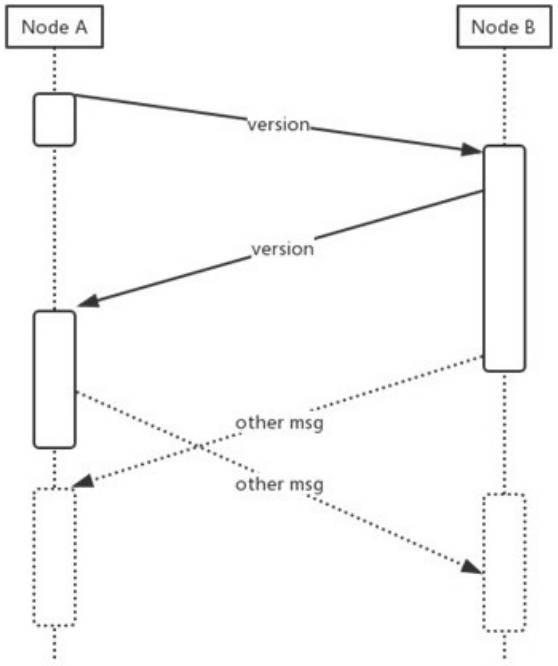


**Figure 2.** Send and receive of a Version packet.

When the node receives the Version package, it will not only return its own Version package. Instead, according to the sender's Height field in the Version package, it will check the status of its own blockchain data and determine whether there is data to be synchronized. If the sender's Height is higher than its own, it will enter the data synchronization process, and download the missing block. In particular, with the exception of the central node, all nodes need to have a node address as the receiver of the initial Version package when they are initially connected to the system. In our work, we set this initial node as the central node of the whole system.

Addr Package

In our system, each node is also responsible for the node push function, which means to inform the new node of the information of the known node. The Addr package is used to inform the new node of the information of the node known to the current node. The Addr package is sent in a very flexible way, usually sent along with the returned version package.

The structure of the Addr package is shown in Table 3.

**Table 3.** Structure of Addr package.

| Field | Meaning |
|---|---|
| AddrFrom | Address of the Addr package sender |
| AddrList | Known nodes of the Addr package sender |

GetBlocks Package

The GetBlocks package serves to inform the receiver of this package to send information about the block it currently owns back to the sender of the GetBlocks package. The package is mainly used for block data synchronization between nodes, and the structure is relatively simple.

However, let's assume a scenario where the receiver of the GetBlocks package has 10,000 blocks and each block's information is 2 KB, then the amount of data returned by a single GetBlocks request might reach 10 MB. This will cause huge network overhead and network congestion, which is obviously unacceptable. Therefore, we design the GetBlocks package to request only returning hash tables for all blocks.

The structure of the GetBlocks package is shown in Table 4.

**Table 4.** Structure of the GetBlocks package.

| Field | Meaning |
|---|---|
| AddrFrom | Address of the GetBlocks package sender |

Inv Package

The Inv package (Inventory package) is a special kind of package that contains all blocks and transactions held by the sender, but it does not directly contain all those data, just the hash value.

The structure of the Inv package is shown in Table 5.

**Table 5.** Structure of the Inv package.

| Field | Meaning |
|---|---|
| AddrFrom | Address of the Inv package sender |
| Type | Type of the Inv package (transaction or block?) |
| Items | Data |

Let us review the scenario we assumed previously. By using the Inv package (as shown in Figure 3), the data transferred can be compressed as the size of a block header,

which could greatly reduce the network overhead and reduce the network burden. Using the Inv package allows the receiver to choose the data to be pulled (through the GetData package) on its own, avoiding a large number of unnecessary data transfers. It also has the advantage that the receiver can know the information of which data it needs to pull from the sender and then pull the real data from different nodes. Therefore, it can improve node utilization and system scalability. However, using Inv also increases the difficulty of the receiver's data management strategy.

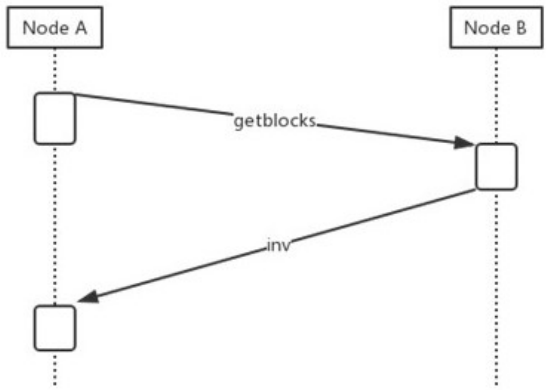


**Figure 3.** GetBlock and Inv pacakges.

GetData

When a node finds that it needs to synchronize data, it sends the GetData package to its adjacent nodes to obtain specific data. The structure of the GetData package is shown in Table 6.

**Table 6.** Structure of the GetData package.

| Field | Meaning |
|---|---|
| AddrFrom | Address of the GetData package sender |
| Type | Type of the GetData package (transaction or block?) |
| ID | Id of the data requested from the GetData package sender |

The processing flow of the GetData package is simple: if the block is requested, the block data is returned according to the block ID (the Hash value of the block). Similarly, if the transaction is requested, the transaction data is returned according to the transaction ID.

Block and Action Package

The Block package and Action package are the data when the GetData package requests a block or request transaction, respectively. The data structure of the two packages is similar (Table 7).

**Table 7.** Structure of the Block/Action package.

| Field | Meaning |
|---|---|
| AddrFrom | Address of the Block/Action package sender |
| Data | Data according to the requested ID |

According to the design of the system, the Data field here corresponds to the serialized data of the specific data. The sender needs to serialize the specific data before sending, and the receiver deserializes it after receiving it to obtain the real data. The serialization of data transfer helps extend and modify the data structure of the system, and improve the scalability and maintainability of the system.

#### 2.2.3. Consensus Algorithm

Public blockchains, such as Bitcoin and Ethereum, build their consensus system based on the PoW (Proof of Work) mechanism. Although PoW has key characteristics such as anti-forking and anti-tampering, in this work, using PoW is not a good choice. In this paper, we propose to construct a smart contract system based on consortium blockchains, where the nodes involved in the block production will be limited, that is, not all the nodes in the system have the authority to write in the ledger. Additionally, in consortium blockchains systems the number of nodes and the computational power of the whole system will not reach the level of the public blockchain at present. Thus building a consensus system based on PoW might lead to many issues, such as 51% computational power problems and resource consumption problems.

Although the proposal of proof of interest alleviates the defect of PoW to some extent, proof of interest is highly dependent on virtual currency. The blockchain system in this paper is designed to be a non-virtual-currency chain, so it is not applicable here.

The PoA (Proof of Authority) consensus system is a kind of consensus system that allows users to create new nodes and make transactions and produce blocks by using the node identity guarantee as "authentication".

The "verifier" node is the core of the PoA consensus system. When the transaction is broadcast to the whole network, the ordinary node will no longer verify the correctness of the transaction, but only as of the intermediate route of the verifier node. The verifier node will finally be responsible for verification and packaging. The management nodes select the verifier node by voting and polling.

The verifier does not need a large amount of computation overhead similar to the PoW, nor does it need the virtual currency guarantee of the PoI (Proof of Identity). The only requirement of a verifier is that it must be a known node that already exists in the system and has been authenticated. The verifier acquires block-producing rights by identification of other nodes.

If the verifier has malicious behavior, or if there is a malfunction, it can be eliminated by voting of other management nodes on the blockchain. In addition to the initial node group, any node that wants to be a new management node must be agreed upon by all management nodes. When a management node performs malicious behavior or actively exits and needs to be removed from the management node group, it is also subject to restrictions similar to when added. Although PoA may also have a similar 51% computational power problem like PoW, it is different in nature and can be fixed by flexible joint decision design.

The DPoS consensus system maintains the node order of the system by means of election voting, and each user has the same authority to decide on the addition and elimination of management nodes. Nodes are ranked by the votes of users. The top $n$ nodes with more votes become management nodes. The authority of each management node to confirm the blockchain transaction and write the distributed account book is equal.

Typically, management nodes select the "verifier" for each round by polling. Each verifier has the right to verify and produce a block for a certain period of time, and the produced block of a timed-out verifier will not be accepted by other management nodes. The fact that a verifier fails to produce a block within the last round block-producing time window will not affect the normal block production of the current round of verifiers.

In view of the advantages and disadvantages of the PoA and DPoS, the design of the blockchain system in this paper adopts a hybrid approach. In the dynamic expansion and reduction of management nodes, the PoA mechanism is adopted. While in the selection of management nodes, the DPoS mechanism is adopted.

In our system, the "block producer" role is played by the block producer node group, and the "verifier" is the responsibility of the central node. The "verifier" does not need to produce a block, but only needs to verify the block production of the block producer and sign the confirmation. The separation of "block producer" and "verifier" increases the risk of centralization, but it can simplify the complexity of the system and improve the system performance.

DPoS in our system is implemented using smart contracts. As the user initiates the DPoS transaction, the system runs and updates the block-producing node group through a smart contract system. When the block-producing node group accumulates a certain amount of transactions, a new block is produced and sent to the "verifier" node. After receiving the request, the "verifier" node will verify the validity of the new block and its source node according to the block data, the block producer node group table, and the current chain length. If the verification passed, the new block would be signed and broadcast to the whole blockchain.

PoA is also implemented using smart contracts. The PoA system authorization is improved to coordinate with the DPoS. When the block producing node group votes, it only determines the addition or deletion behavior but does not directly specify which node to add or eliminate. If the block-producing node group decides to add a node, the node to add is the node with the highest vote from the non-block-producing node group through DPoS voting. If the block-producing node group decides to delete a node, the node to delete is the node with the lowest vote from the block-producing node group through DPoS voting. After a valid block containing PoA transaction is verified and signed, all nodes will execute the block transaction and dynamically perform addition or deletion on the block node group table. This addition or deletion will affect the subsequent block-producing validation and will not take effect immediately.

#### 2.2.4. Data Storage

In order to improve system performance and scalability, the database in our blockchain system is implemented using the NoSQL database. For smart contract table data, block node group table, and other hot data, we use a memory database for storage.

For each smart contract, we need to support the transaction level operation of the smart contract. When the transaction is wrong, we need to be able to roll back the data in time. This requires that for each smart contract we have a separate transaction-enabled memory database to manage its data. Traditional memory databases, such as Redis or Memcache, can not support transactions. Because of the separation of storage and services, it can not flexibly store database table objects. As a result, in this paper, we have implemented a lightweight KV memory database based on the radix tree index.

The radix tree is spatially optimized on the basis of the prefix tree. In a radix tree, when a node is the only child node of its parent node, it merges with the parent node. As a result, the number of child nodes per node in the radix tree is at most the radix $r$. Unlike the common prefix tree structure, the edge of the radix tree can represent either a single element (such as a character) or an ordered set of elements (such as a string), which makes the radix tree more efficient in processing small data-set (especially if strings are long) or data-set of strings that share long prefixes.

In this work, the radix tree is used as the base index of our database. The insert, delete, and lookup operation based on the radix tree index is realized [38]. The time complexity of all these operations is $O(k)$, where the value of the length $k$ is determined by the maximum length of the keyword in the index tree, and the keyword length unit is determined by the number of radix digits of the radix tree [39].

The index tree structure in this work consists of the following two elements:

- Edge. The edges of the radix index tree consist of its pointing nodes and its corresponding prefix labels.
- Node. The radix index tree node consists of a value corresponding to its keyword and a set of values pointing to its child nodes. In particular, when the node is not a leaf node, its corresponding value is null.

When performing an insert operation, a new keyword is added to the index tree and attempts to minimize the storage. When inserting a keyword, we will first search with the keyword prefix until the search can not continue. There are two situations:

1. The remainder of the keyword has no common prefix with all edges with the current node as the root node. For this case, we just need to add an edge, as shown in Figure 4a.
2. The remainder of the keyword has common prefix with one of the edges with the current node as the root node. In this case, we split the edge into a common edge and two edges connecting the common edge, which ensures that the number of child nodes of any node does not exceed the radix of the index tree, as shown in Figure 4b.

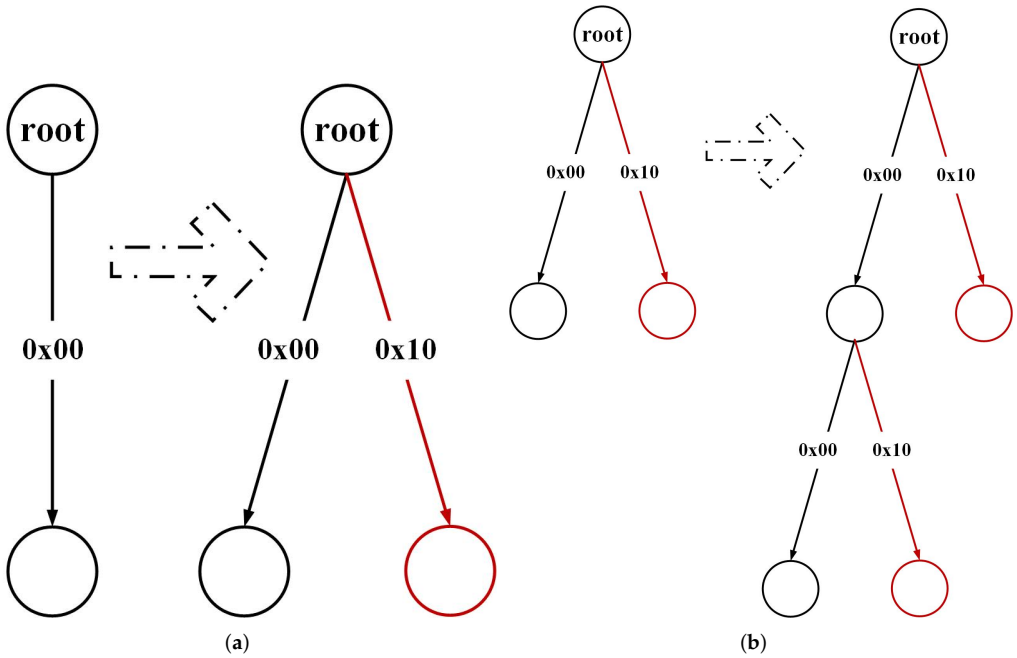


**Figure 4.** Illustration of insertion operation. (**a**) Insertion with no common prefix situation, (**b**) insertion with common prefix with one edge situation.

The process of delete operation is similar to insert operation. When performing a delete operation, the keyword is deleted, and it will also try to merge subtree indexes to minimize storage.

To delete a keyword $x$, you need to first find the leaf node representing the $x$. If you find it, delete the corresponding leaf node. If the parent node of that leaf node has only one other child node, the edge label of that child node will be attached to the edge label of the parent node and the child node will be deleted.

The lookup operation will query the data corresponding to the keyword.

In the smart contract system, sometimes the contract will fail or need to be rolled back. For example, in a multi-vote scenario, a user votes for multiple users at the same time, which involves a multi-table write operation. However, if one of the voting operations fails, the previous voting operation needs to be rolled back to ensure data consistency, which is the active rollback operation of the contract. There is also a situation where the contract is executed with irreparable errors (such as array crossing). The contract can not be executed in the normal process and the database needs to be rolled back. This is the passive rollback operation of the contract. In this work, we implement the transaction level structure of our database based on read–write lock and write cache, as shown in Figure 5.

For read-only operations, the read handle can be created directly because it is not destructive to the underlying data. The read handle contains database references, radix index tree references, read-only logos, etc. For the read-only handle, insert, modify, and delete operations are illegal operations. When looking up a read-only handle, it returns the result of the query directly through the underlying database index.

The read/write operation should be designed to avoid the destruction of the consistency of the underlying data. There are two typical scenarios:

1. Multiple read/write operations modify the same underlying data at the same time, destroying the consistency, isolation and persistence of transactions;
2. One write operation fails and needs to be rolled back after modifying multiple data, destroying the atomicity, consistency, and isolation of transaction, resulting in dirty data.

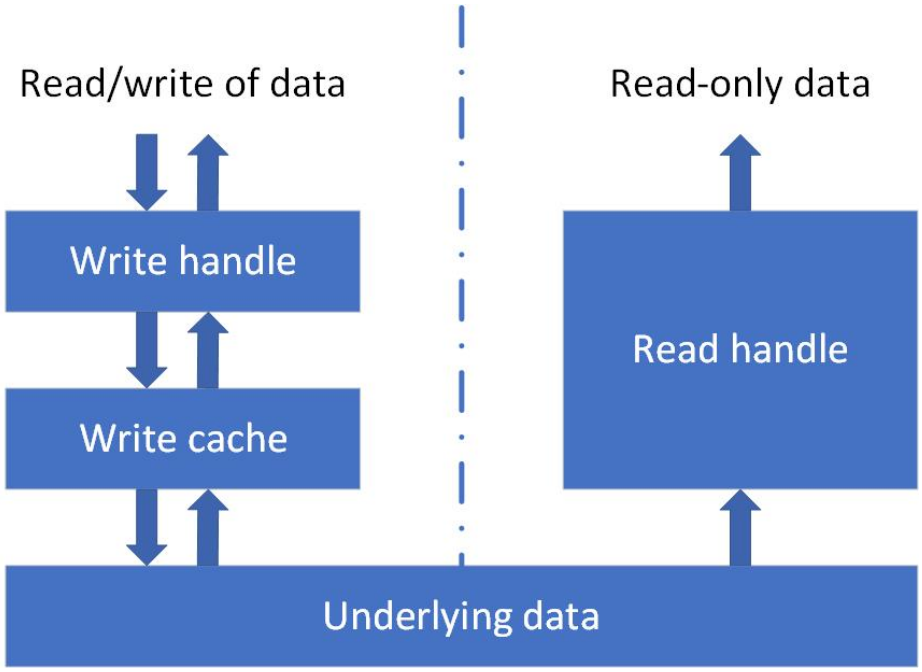


**Figure 5.** Transaction level structure.

For scenario 1, our database adopts the way of write lock, that is, for the same database object, at the same time can only have one read/write handle. This write lock differs from a write lock in a traditional database, such as MySQL, in that the write lock in a traditional database is an exclusive lock. Before a transaction is turned on, it keeps trying to get the write lock until it succeeds. When a transaction occupies a write lock, only the transaction can write to the object. The write lock only exclusive write operation, so other read handles can read the underlying data at the same time. Write locks are released when a transaction is completed or an error or exception causes rollback.

For scenario 2, our system adopts the way of a write cache. The two-level write cache is used to ensure that the write operation meets the four elements of the transaction while improving performance.

Level 1 cache is a transaction cache. In the write handle, a write cache based on the radix tree is set for each table that needs to be modified. When adding or deleting data, add or delete operations will be temporarily written to the cache, while the underlying data will not be overwritten. When the data is queried, the data written to the cache is first queried and the cache hit data is returned first. When the cache is not hit, it is queried from the underlying data. When the transaction is complete, the write handle writes the data in the cache to the underlying data. When the transaction fails or is abnormal, the write handle directly discards the changes in the cache to avoid dirty data.

Level 2 cache is a memory cache. Level 1 cache uses the radix tree as the cache data index, which will frequently request and free node memory when adding and deleting nodes. When the number of write operations is big, a large number of node memory requests and releases will cause unnecessary system overhead. Therefore, the memory pool is used to manage the cache temporary index nodes which are used frequently and have a relatively short living period. At the end of the transaction, the node released from the level 1 cache will enter the memory pool to wait for the next use, instead of directly releasing the memory, thus reducing the time overhead of the operating system memory allocation.

#### 2.2.5. Smart Contract

In this paper, a user-level smart contract system is implemented in the consortium blockchain system. Each user account can set up a smart contract system and several associated user-level tables.

Some important fields in the user-level table are shown in Table 8.

**Table 8.** Important fields in the user-level table.

| Field | Type | Meaning |
|---|---|---|
| Name | String | User account name (unique index) |
| Pubkey | BinaryArray | Public key of the user account |
| Table | StrucMap | Table information of the user account |
| Api | StrucMap | Interface for smart contract |
| Jscript | String | Code for smart contract |
| DB | DatabaseHandler | Handler for smart contract database |
| Schema | DatabaseSchema | Index type for smart contract database |

Both the definition and implementation of the user-level table are in the radix tree-based memory database of this system, and the Name field is used as the unique index. At the same time, for each user-level table, there is a database handle of the user and a database schema corresponding to the handle, which together form the basic data storage unit of the user account to store the data generated by the smart contract and provide the data for the smart contract running environment. The database handle and schema of different users are different, which ensures the data isolation of different contracts so that each smart contract can run independently.

For each smart contract, we specify three important fields:

1. API: The API field is used to define the interface type of the smart contract, which is a mapping table from string to array. The structure of the API field is shown in Figure 6. The API interface part specifies the name of its interface. Each interface name corresponds to a set of input parameters to provide the running environment of the interface. In particular, we can also support a zero-parameter interface;
2. Table: The Table field is used to define the database table structure for each smart contract, which is a tableName to tableHeader mapping table. The structure of the table field is shown in Figure 7. We specify three basic data types for each user-level data table: string, float, and int. Users can add, delete, and modify their data through a smart contract. In order to ensure the fairness of the contract, the database table read permission of the contract is open to all users. The Table field and the Schema field both define the header field of the contract database table, but they are different: the former represents the table structure of the contract and is used for the verification of the smart contract and the fast query of the database table structure, is open to the users; while the latter plays a key role in the index query of the database data and is not visible to other users.
3. Jscript: The Jscript field is the script field of the smart contract, which specifies the running process of the smart contract. We use Javascript as the scripting language of smart contract, otto as its interpreter.

When deploying a smart contract, a user needs to clearly define the deployed account name, contract interface, contract table structure, and contract script. After the contract is deployed, the original contract script can be modified and upgraded without affecting the table structure and data.

The process of the user calling a smart contract is a process of transaction. The transaction contains the account name, caller, interface name, and interface parameter of the deployment contract. By providing built-in APIs, it provides an interface for contracts to interact with databases and transactions. Those built-in APIs are automatically loaded

into the Javascript virtual machine environment before the contract runs, so the contract script can use those built-in APIs directly by a function call.

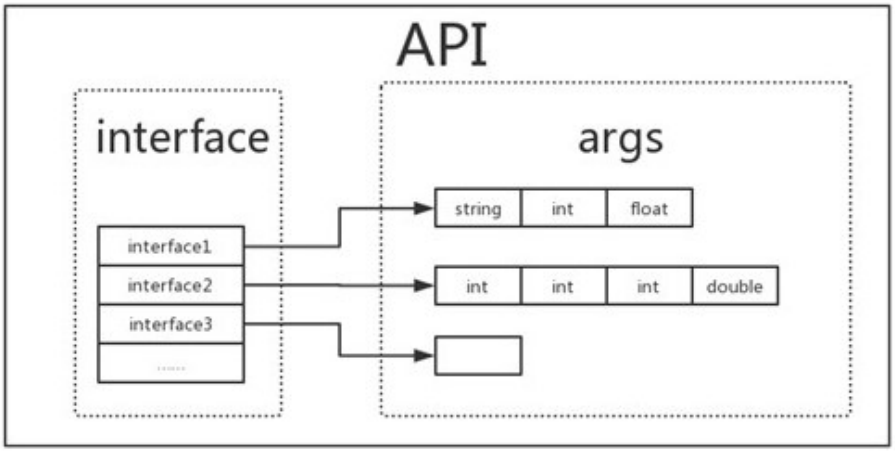


**Figure 6.** Structure of the API field.

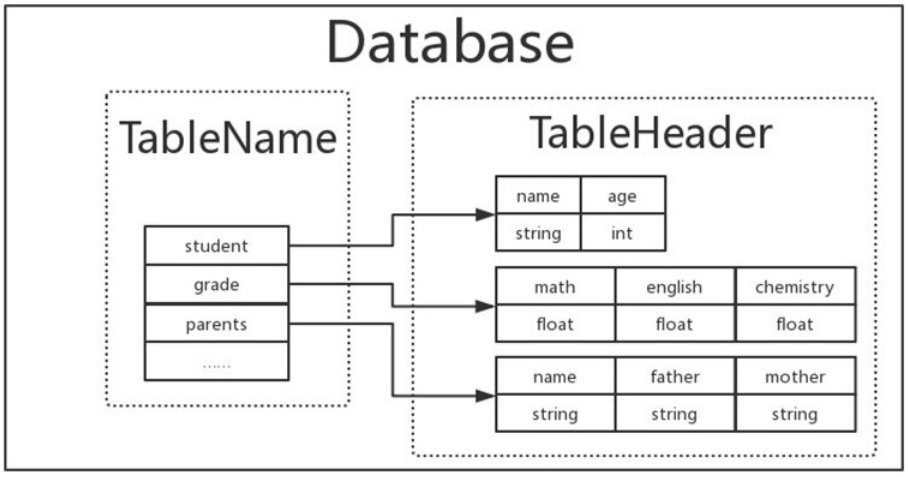


**Figure 7.** Structure of the Table field.

## 3. Results

### *3.1. Results of the Radix Tree Database*

In this paper, we have implemented a database system based on a radix tree. We will compare it with other different databases to verify its performance.

#### 3.1.1. Comparison with Redis

The test environment is configured with Intel 4 core 2.2 GHz processor, 8 G memory, Debian9 operating system, kernel version 4.4.0-33 (Santa Clara, CA, USA). The Redis server uses Redis4.0. The client uses go-Redis single routine write.

The experiment uses 40-bit UUID as the keyword, carries on the fixed number write operation to the radix tree database, the Redis database, and then reads all the written data to verify its write correctness. Finally, the total time consumption of write and read operation is recorded separately to compare the performance of the radix tree database with Redis.

The result is shown in Table 9.

**Table 9.** Comparison with Redis.

| Amount of Data | Time Consumed of Redis | Time Consumed of Radix Tree Database |
|---|---|---|
| 1 | 781.75 µs | 261.196 µs |
| 10 | 648.90 µs | 84.495 µs |
| 100 | 1.52 ms | 1.15 ms |
| 1000 | 90.60 ms | 15.34 ms |
| 10,000 | 647.27 ms | 89.47 ms |
| 100,000 | 4.81 s | 1.03 s |
| 200,000 | 10.32 s | 2.08 s |

As can be seen, the performance of the radix tree database is better than Redis.

#### 3.1.2. Comparison with MySQL and MongoDB

The test environment uses Intel 4 core 2.2 GHz processor, 8 G memory, Debian9 operating system, kernel version 4.4.0-33. The MySQL server uses MySQL5.7. The client uses a gorm single routine write. The MongoDB server uses mongo4.0.7. The client uses mongo-driver single routine write.

The experiment uses 40-bit UUID as the keyword, carries on the fixed number write operation to the MySQL, MongoDB, and the radix tree database, records the total write time separately, in order to compare the performance of the radix tree database with MySQL and MongoDB.

The result is shown in Table 10.

**Table 10.** Time consumed for data write compared with MySQL/MongoDB.

| Amount of Data | MySQL (Non-Transactional) | MySQL (Transactional) | MongoDB | Radix Tree Database |
|---|---|---|---|---|
| 1 | 783.65 ms | 730.86 ms | 22.31 ms | 34.23 µs |
| 10 | 882.76 ms | 970.67 ms | 23.51 ms | 106.92 µs |
| 100 | 5.21 s | 1.20 s | 32.49 ms | 1.23 ms |
| 1000 | 41.37 s | 2.02 s | 204.92 ms | 12.77 ms |
| 10,000 | 9 m 30 s | 7.89 s | 1.57 s | 159.71 ms |

As can be seen, when large-scale write operations are performed, the performance of MySQL non-transactional and transactional is quite different due to its table lock mechanism. For non-transactional conditions, the time-consuming of inserting a single for MySQL increases rapidly as the amount of data increases. Under transactional conditions, the time overhead of MySQL write operations is reduced, but still can not be compared with non-relational databases such as MongoDB. The radix tree database designed in this paper has obvious advantages over persistent storage databases in performing data write operations.

#### 3.1.3. CPU/MEM Usage

The test environment is Intel 4 core 2.3 GHz processor, with macOS 10.14.4 as the operating system. 1,000,000 write operations were performed using the radix tree database client, and then all written data were read to verify the correctness of their writes. CPU status is recorded during the write and read operations, and memory usage is recorded after the completion of operations.

CPU experimental results are shown in Figure 8. The total test case time consumption is 16.14 s. We can see that the pure time consumption (Insert and Commit) of the insertion operation of 1,000,000 data in the test case is 8.41 s, while the query operation takes only 0.36 s.

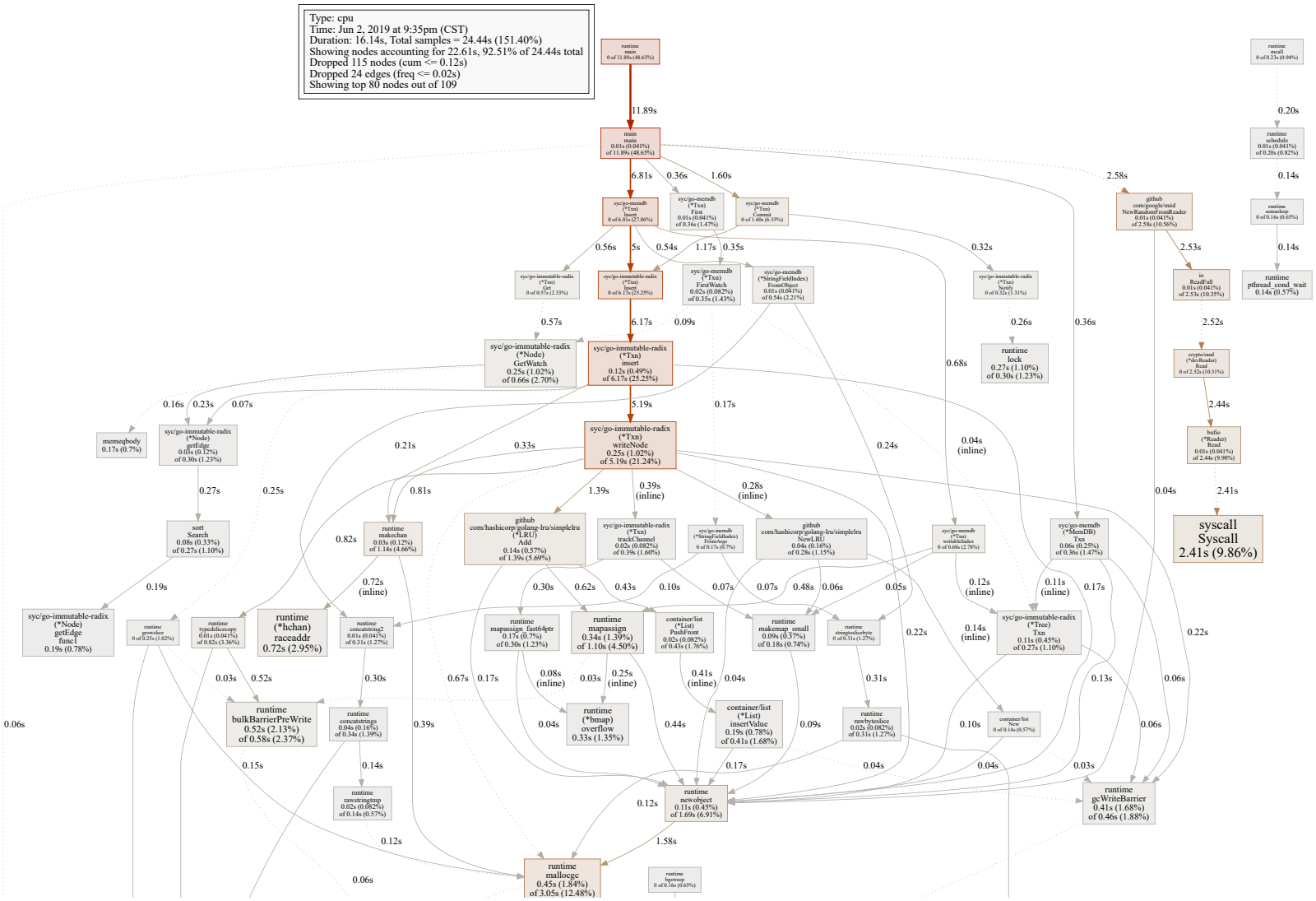


**Figure 8.** CPU usage result.

MEM experimental results are shown in Figure 9. Total test memory consumption is 513.69 MB. As can be seen, when the amount of data in a single table reaches 1,000,000, its memory consumption has reached nearly 300 MB. When the amount of data in the table increases, the overhead of maintaining the radix tree node will increase, resulting in the huge memory consumption of the radix tree database, which is consistent with the theory. On the other hand, it can be proved that there is still great room for improvement in the implementation of the radix tree database.

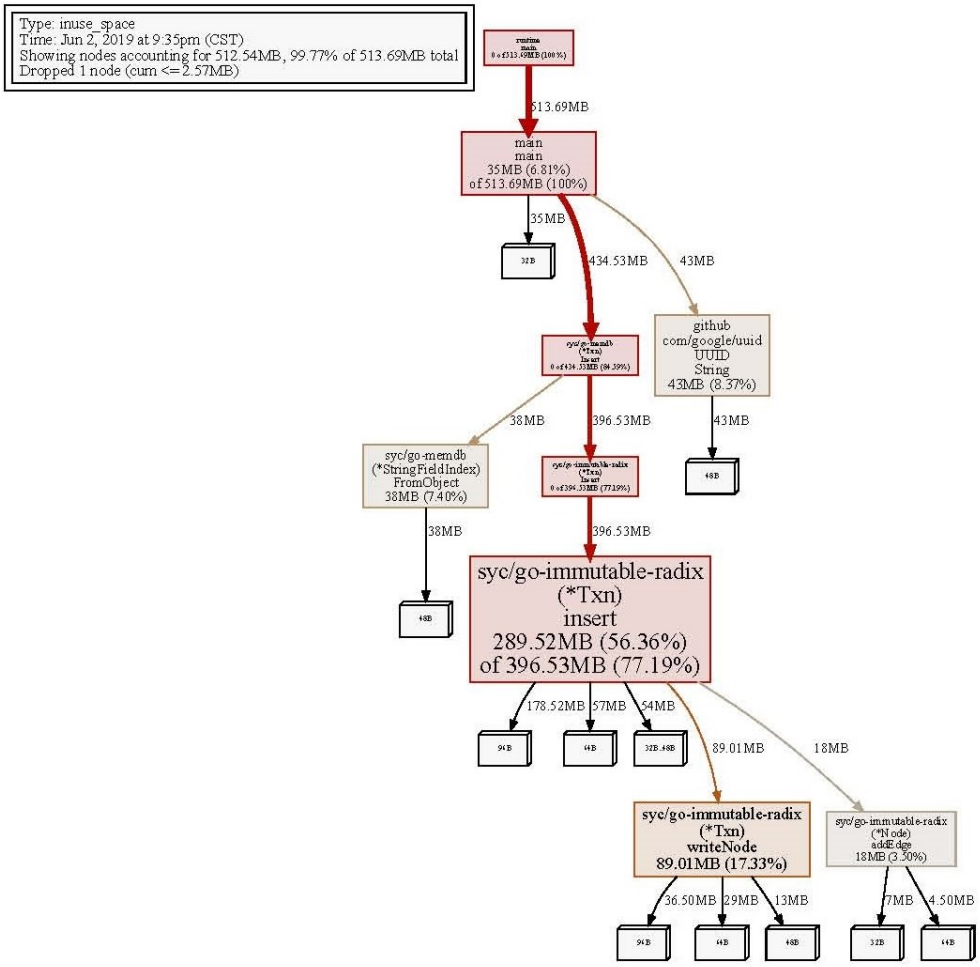


**Figure 9.** MEM usage result.

### *3.2. Performance and Stability of the Smart Contract System*

In this section, the smart contract system implemented in this paper is tested for different system configurations and compared to verify the performance and stability.

In a blockchain system, each block can contain multiple transactions. The system block-producing speed is negatively related to the maximum number of transactions contained in the block. The smaller the maximum number of transactions contained in the block, the smaller the system block-producing transaction threshold, the easier the block-producing and vice versa. The small transaction threshold is beneficial to the fast execution and broadcast of the transaction, but it will cause a certain network burden. We varied the maximum number of transactions in blocks and tested the contract. The key scripts of the contract were as follows:

```
......
    var data=getUserData(contract,"inc","inc";)
    if (data == null){
       data=["inc",1]
    }
    else{
       data[1]=parseInt(data[1]+1);
    }
    if(!setUserData(contract,"inc",data)){
       return SysErr("set inc failed.");
    }
......
```

This contract would count the number of calls to the blocks actually written. The experiment uses the Intel 4 core 2.3 GHz processor, the operating system is macOS 10.14.4, and the minimum interval time limit is removed in the system for increasing the QPS. The result is shown in Table 11 and Figure 10.

**Table 11.** The influence of the maximum number of transactions in block on system stability under large transaction requests.

| Maximum Number of Transactions in Block | Number of Transactions Sent | Number of Contract Execution (Inc Value) |
|---|---|---|
| 1 | 500 | 500 |
| 1 | 1000 | 1000 |
| 1 | 2000 | 1431 |
| 2 | 500 | 500 |
| 2 | 1000 | 1000 |
| 2 | 2000 | 1778 |
| 10 | 1000 | 1000 |
| 10 | 2000 | 2000 |
| 100 | 2000 | 2000 |
| 100 | 3000 | 2600 |
| 1000 | 3000 | 3000 |
| 1000 | 5000 | 5000 |

It can be seen from Table 11 that when the maximum number of transactions in the block is the smallest if the transaction volume is at the level of four orders of magnitude, the loss is obvious. The reason is that when the maximum number of block transactions is 1, block-producing requests are sent out frequently and in large quantities, resulting in serious network load. The large number of blocks produced and their subsequent requests occupy a large number of network resources, which is consistent with the experimental expectations.

Figure 10 shows that the transaction processing speed of this system is faster than that of Bitcoin (7 QPS) and Ethereum (15 QPS). At 20 QPS, the system can process transaction requests as expected, forming a linear relationship. However, when the QPS is increased to 100 or 500, the transaction requests actually processed by the system differ greatly from the expected transaction requests. Through further analysis of the log, it is found that because the system uses the way of short link when the QPS is large, the system spends a lot of resources on TCP connection processing. This is in line with the wavy relationship in the broken line diagram. Whenever the number of links reaches a certain order of magnitude, because the network resources are not released temporarily, in order to prevent the system from suffering excessive requests, some transaction requests will be discarded, which is in line with the experimental expectations.

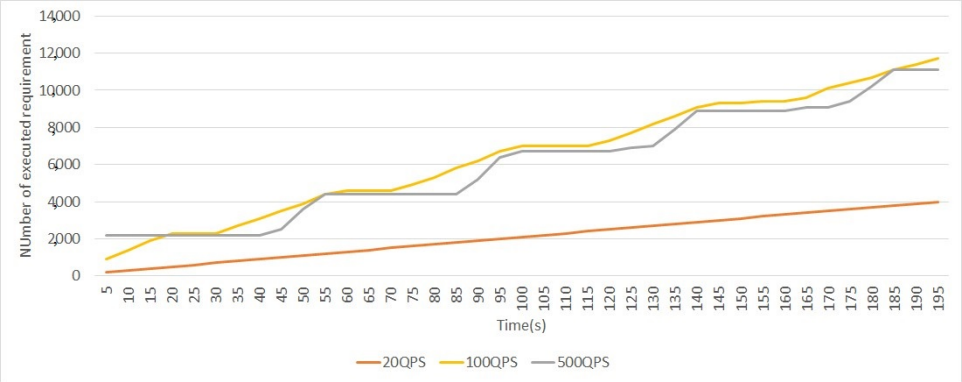


**Figure 10.** Effect of different transaction request QPS on system transaction execution performance.

## 4. Conclusions

In this paper, we have presented our effort at constructing a smart contract system based on a consortium blockchains system. We have presented our method including the designing of user accounts, network structure, consensus system, and database. We have designed a radix tree-based memory database in this work. Experiments have been done to verify the performance and stability of our smart contract system.

Results show that the main advantages of our solution are faster block-producing speed and lower computational cost, which is very attractive in many IoT applications. Current Ethereum systems usually experience issues such as race condition, under/overflow transaction order to assumptions, dependency on timestamps, short address attacks, etc. In this work, we have constructed our own code execution module and have not used the contract design language of Ethereum, so could avoid the issues associated with Ethereum. RSA and ECC have their own advantages, thus we believe that the combination design of them could bring an edge for many IoT application scenarios, and could help to increase the flexibility of the address/account system, without much limitation on that.

According to the results of our method, we believe it could meet the requirement of future potential usage of IoT applications. Based on the smart contract system designed in this paper, blockchain can be more easily applied in payment, product traceability, authority authentication, and other fields. Through a certain centralized way, the system is easier to manage, can reduce the management expenditure, and the power and other resource consumption is less, which is conducive to environmental protection. However, also according to the experimental results, a large number of links and release operations are caused by the use of short links between nodes to maintain communication. That is the main limitation of the current version of our method. In the future, we would want to improve this.

**Author Contributions:** Conceptualization, Y.S.; methodology, Y.S.; software, Y.S.; validation, T.L.; formal analysis, Z.H.; investigation, Z.H.; resources, X.Y.; data curation, X.Y.; writing—original draft preparation, T.L.; visualization, T.L.; supervision, X.Y.; project administration, X.Y.; funding acquisition, X.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is supported by the National Natural Science Foundation of China under Grant No. 91846303, and the Beijing Municipal Natural Science Foundation under Grand No.4212043.

**Conflicts of Interest:** The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## References

1. Haber, S.; Stornetta, W.S. How to Time-Stamp a Digital Document. *J. Cryptol.* **1991**, *3*, 99–111. [CrossRef]
2. Haber, S.; Bayer, D.; Stornetta, W.S. Improving the Efficiency and Reliability of Digital Time-Stamping. In *Sequences II: Methods in Communication, Security and Computer Science*; Springer: New York, NY, USA, 1993; pp. 329–334.
3. Anderson, R.J. The Eternity Service. In Proceedings of the Pragocrypt, Prague, Czech Republic, 30 September–3 October 1996.

4. Schneier, B.; Kelsey, J. Cryptographic Support for Secure Logs on Untrusted Machines. In Proceedings of the Seventh USENIX Security Symposium, San Antonio, TX, USA, 26–29 January 1998; pp. 53–62.
5. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. *Decentralized Bus. Rev.* **2008**, 21260.
6. Merkle, R.C. Protocols for public key cryptosystems. In Proceedings of the 1980 Symposium on Security and Privacy, Oakland, CA, USA, 14–16 April 1980; IEEE Computer Society: Washington, DC, USA, 1980; pp. 122–133.
7. Palmer, S. What Is a Smart Contract? Available online: https://www.shellypalmer.com/2018/05/what-is-a-smart-contract/ (accessed on 9 June 2020).
8. Jacob, A.; Klemens, S. What Are Smart Contracts? A Beginners Guide. Available online: https://www.bitpremier.com/smart-contracts (accessed on 9 June 2020).
9. Kakavand, H.; Sevres, N.; Chilton, B. *The Blockchain Revolution: An Analysis of Regulation and Technology Related to Distributed Ledger Technologies*; Social Science Electronic Publishing: Waltham, MA, USA, 2016.
10. Islam, M.N.; Kundu, S. Poster Abstract: Preserving IoT Privacy in Sharing Economy Via Smart Contract. In Proceedings of the 2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI), Orlando, FL, USA, 17–20 April 2018; pp. 296–297.
11. Frantz, C.K.; Nowostawski, M. From institutions to code: Towards automated generation of smart contracts. In Proceedings of the 2016 IEEE 1st International Workshops on Foundations and Applications of Self Systems, Augsburg, Germany, 12–16 September 2016; pp. 210–215.
12. Gao, Z.; Xu, L.; Chen, L.; Shah, N.; Lu, Y.; Shi, W. Scalable Blockchain Based Smart Contract Execution. In Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, China, 15–17 December 2018; pp. 352–359.
13. Beyer, S. Ethereum Smart Contract Security. Available online: https://medium.com/cryptronics/ethereum-smart-contract-security-73b0ede73fa8 (accessed on 29 January 2018).
14. Singh, M. Blockchain Technology for Data Management in Industry 4.0. In *Blockchain Technology for Industry 4.0*; Blockchain Technologies; Righi, R.R., Alberti, A., Singh, M., Eds.; Springer: Singapore, 2020.
15. Lee, S.-W.; Singh, I.; Mohammadian, M. *Blockchain Technology for IoT Applications*; Springer: Singapore, 2021.
16. Singh, D.; Kim, J.-H.; Singh, M. *Blockchain Technologies*; Springer: Berlin/Heidelberg, Germany, 2022 .
17. Calzada, I. 'Algorithmic nations': Seeing like a city-regional and techno-political conceptual assemblage. *Reg. Stud. Reg. Sci.* **2018**, *5*, 267–289. [CrossRef]
18. Shahnaz, A.; Qamar, U.; Khalid, A. Using Blockchain for Electronic Health Records. *IEEE Access* **2019**, *7*, 147782–147795. [CrossRef]
19. Al Omar, A.; Rahman, M.S.; Basu, A.; Kiyomoto, S. MediBchain: A Blockchain Based Privacy Preserving Platform for Healthcare Data. In *Security, Privacy, and Anonymity in Computation, Communication, and Storage, Proceedings of the SpaCCS 2017, Guangzhou, China, 12–15 December 2017* ; Wang, G., Atiquzzaman, M., Yan, Z., Choo, K.K., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2018; Volume 10658.
20. Mengelkamp, E.; Notheisen, B.; Beer, C.; Dauer, D.; Weinhardt, C. A blockchain-based smart grid: Towards sustainable local energy markets. *Comput. Sci. Res. Dev.* **2018**, *33*, 207–214. [CrossRef]
21. Li, Z.; Kang, J.; Yu, R.; Ye, D.; Deng, Q.; Zhang, Y. Consortium Blockchain for Secure Energy Trading in Industrial Internet of Things. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3690–3700. [CrossRef]
22. Singla, V.; Malav, I.K.; Kaur, J.; Kalra, S. Develop Leave Application using Blockchain Smart Contract. In Proceedings of the 2019 11th International Conference on Communication Systems & Networks (COMSNETS), Bengaluru, India, 7–11 January 2019; pp. 547–549. [CrossRef]
23. Omar, A.S.; Basir, O. Smart Phone Anti-counterfeiting System Using a Decentralized Identity Management Framework. In Proceedings of the 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), Edmonton, AB, Canada, 5–8 May 2019; pp. 1–5. [CrossRef]
24. Nguyen, T.Q.; Das, A.K.; Tran, L.T. NEO Smart Contract for Drought-Based Insurance. In Proceedings of the 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), Edmonton, AB, Canada, 5–8 May 2019; pp. 1–4. [CrossRef]
25. Nagothu, D.; Xu, R.; Nikouei, S.Y.; Chen, Y. A Microservice-enabled Architecture for Smart Surveillance using Blockchain Technology. In Proceedings of the 2018 IEEE International Smart Cities Conference (ISC2), Kansas City, MO, USA, 16–19 September 2018; pp. 1–4. [CrossRef]
26. Cheng, R.; Zhang, F.; Kos, J.; He, W.; Hynes, N.; Johnson, N.; Juels, A.; Miller, A.; Song, D. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden, 17–19 June 2019; pp. 185–200. [CrossRef]
27. Saquib, N.; Bakir, F.; Krintz, C.; Wolski, R. A Resource-Efficient Smart Contract for Privacy Preserving Smart Home Systems. In Proceedings of the 2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI), Atlanta, GA, USA, 18–21 October 2021; pp. 532–539. [CrossRef]
28. Dustdar, S.; Fernandez, P.; Garcia, J.M.; Ruiz-Cortes, A. Elastic Smart Contracts in Blockchains. *IEEE/CAA J. Autom. Sin.* **2021**, *8*, 1901–1912. [CrossRef]

29. Wickstrom, J.; Westerlund, M.; Pulkkis, G. Smart Contract based Distributed IoT Security: A Protocol for Autonomous Device Management. In Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 10–13 May 2021; pp. 776–781. [CrossRef]
30. El Ioini, N.; Pahl, C. A Review of Distributed Ledger Technologies. In On the Move to Meaningful Internet Systems. In Proceedings of the OTM 2018 Conferences, OTM 2018, Valletta, Malta, 22–26 October 2018; Panetto, H., Debruyne, C., Proper, H., Ardagna, C., Roman, D., Meersman, R. Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2018; Volume 11230.
31. Suciu, G.; Nădrag, C.; Istrate, C.; Vulpe, A.; Ditu, M.; Subea, O. Comparative Analysis of Distributed Ledger Technologies. In Proceedings of the 2018 Global Wireless Summit (GWS), Chiang Rai, Thailand, 25–28 November 2018; pp. 370–373. [CrossRef]
32. Li, S.; Xu, Q.; Hou, P.; Chen, X. Exploring the Challenges of Developing and Operating Consortium Blockchains: A Case Study. In Proceedings of the 2020 Evaluation and Assessment in Software Engineering, Trondheim, Norway, 15–17 April 2020; pp. 398–404. [CrossRef]
33. Dib, O.; Brousmiche, K.L.; Durand, A.; Thea, E.; Hamida, E.B. Consortium blockchains: Overview, applications and challenges. *Int. J. Adv. Telecommun.* **2018**, *11*, 51–64.
34. Koblitz, N. Public Key. In *Graduate Texts in Mathematics*; Springer: Berlin/Heidelberg, Germany, 1994; Volume 114, pp. 83–124
35. Zhou, B.; Meng, X.; Stanley, H.E. Power-law distribution of degree–degree distance: A better representation of the scale-free property of complex networks. *Proc. Natl. Acad. Sci. USA* **2020**, *117*, 14812–14818. [CrossRef] [PubMed]
36. Esquivel-Gómez, J.; Stevens-Navarro, E.; Pineda-Rico, U.; Acosta-Elias, J. A growth model for directed complex networks with power-law shape in the out-degree distribution. *Sci. Rep.* **2015**, *5*, 7670. [CrossRef] [PubMed]
37. Fletcher, G.H.L.; Sheth, H.A.; Börner, K. Unstructured Peer-to-Peer Networks: Topological Properties and Search Performance. In Proceedings of the Agents and Peer-to-Peer Computing, AP2PC 2004, New York, NY, USA, 19 July 2004; Moro, G., Bergamaschi, S., Aberer, K., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3601.
38. Leis, V.; Kemper, A.; Neumann, T. The adaptive radix tree: Artful indexing for main-memory databases. In Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE'13), Brisbane, QLD, Australia, 8–12 April 2013; pp. 38–49.
39. Wu, G.; Song, Y.; Zhao, G.; Sun, W.; Han, D.; Qiao, B.; Wang, G.; Yuan, Y. Cracking In-Memory Database Index A Case Study for Adaptive Radix Tree Index. *arXiv* **2019**, arXiv:1911.11387.