applied sciences

Article

OSCMS: A Decentralized Open-Source Coordination Management System Using a Novel Triple-Blockchain Architecture

Jiakai Chen [1], Yishi Zhao [1,*] and Xiao Chen [2]

1 School of Computer Science, China University of Geosciences, Wuhan 430078, China; xgcjk@cug.edu.cn
2 School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, UK; xiao.chen@ed.ac.uk
* Correspondence: zhaoyishi@cug.edu.cn

Abstract: Open-source systems help to manage the rapid development of software, while governing open-source systems properly can effectively promote software and software engineering. However, some significant problems, such as code controls, incentives, interaction and cooperation, automation, transparency and fairness of rights and responsibilities, cannot be properly solved by traditional methodologies. The decentralization, immutability, change in trust mode and smart contract programming of blockchain provide new solutions. In order to solve the problems of traditional centralized open-source governance, this paper proposes a decentralized open-source coordination management system using a novel triple-blockchain architecture. Through the analysis of traditional and blockchain-based research, the business and technical issues that need to be addressed in decentralized open-source governance systems have been emphatically studied. Combined with triple-blockchain architecture, smart contracts, oracles and continuous integration tools, we study the decentralization of open-source businesses and make them more trustworthy, automated and coordinated. An identity authentication mechanism is designed for permission control and inter-community collaboration. A decentralized open-source reputation is proposed for incentive and reference. We also improved the DPoS (Delegated Proof of Stake) consensus under triple-blockchain architecture to reduce repeated elections. By constructing the OSCMS prototype based on the proposed architecture model, many comparative experiments were conducted under different parameters and conditions and showed good feasibility, scalability, reliability and performance. The OSCMS not only solves the shortcomings of previous research but also provides a comprehensive and feasible reference for the decentralized practice of open-source governance.

Keywords: blockchain; open-source community; decentralized system; coordination management; multi-blockchain architecture; consensus protocol

check for updates

Citation: Chen, J.; Zhao, Y.; Chen, X. OSCMS: A Decentralized Open-Source Coordination Management System Using a Novel Triple-Blockchain Architecture. *Appl. Sci.* **2023**, *13*, 6580. https://doi.org/10.3390/app13116580

Academic Editor: Gianluca Lax

Received: 4 April 2023
Revised: 21 May 2023
Accepted: 26 May 2023
Published: 29 May 2023

1. Introduction

As the scale and complexity of software are increasing rapidly, it is inevitable to rely on open-source (OS) ecosystems for further development [1]. Many successful and active OS projects or communities, such as Linux, Apache and GitHub, benefit from OS code contributed by the public. The OS community is a network platform that publishes source codes according to the corresponding OSS license agreement. People communicate, practice, help each other and make contributions in the community to aid the progress of individuals or organizations and promote software engineering. For instance, GitHub is an OS code warehouse and version control system used all over the world. It provides tools for collaborative software development, social networking, reputation and project and access management and has formed a large community of OS projects and developers [2]. Constructing, managing and maintaining open-source communities properly and steadily can help software and software engineering flourish.

Although the introduction of open-source software (OSS) can improve development efficiency and save costs, there are still some problems in OS project management and OS ecosystem governance. Due to the dispersion of OSS and participants, the problems to be solved in OS ecosystems include code control [3,4], incentive [5], interaction and cooperation [6,7], automation [8,9], transparency and fairness of rights and responsibilities [10–13]. Traditional solutions and architectural models focus on perspectives such as communities, projects, OS participants, enterprises and organizations but are limited to centralization. It is easy to overlook one aspect and lose sight of the other, especially with issues of trust and fairness.

The software field has begun to advance through decentralized methods such as software supply chain management, outsourcing management, collaborative development, etc. Promoting the transformation of OS community ecosystems to decentralization can further solve the above problems. As a decentralized and shared distributed ledger [14], blockchain has the advantages of being tamper-proof, having trusted traceability, the automatic execution of smart contracts, etc. It can not only integrate various incentive policies and promote multi-party collaboration, but also solve the problems of transparency, fairness and trust. Fully engaging dedicated OS users in community governance is the way forward.

Related research based on blockchain rarely pays attention to all five problems. For OS project management and OS ecosystem governance, which of the requirements are common to OS also needs to be considered. This paper proposes a decentralized open-source coordination management system (OSCMS) based on blockchain to fully engage OS users in governance. The OSCMS supports a variety of OS scenarios in a secure, sustainable, coordinated, automated, transparent and fair environment. The main contributions of the proposed work are as follows:

- Using smart contracts, oracles and continuous integration tools, the OSCMS meets OS requirements under decentralization and provides relevant management functions, such as OS information recording, citation, feedback, rating, sponsorship and OS reward. These are all important components of OS and can promote the OS community. The new goals, organic combination and feasibility of these functions are emphatically studied. This work improves the traceability and trustworthiness of OS code and makes OS business and system management more automated.
- A novel triple-blockchain architecture is proposed to further coordinate and optimize the OSCMS, which is composed of three blockchains that cooperate with each other to maintain the system. For permission control, an identity authentication mechanism is designed, which allows for inter-community collaboration. A decentralized open-source reputation stored in the form of a Merkle Patricia tree is designed, which has incentive and reference values. Based on Ethereum, a triple-blockchain DPoS consensus mechanism is added, in which the same delegated miners are shared after a round of voting. Finally, a system prototype is built by modifying the Ethereum source code and creating relevant smart contracts.

The remainder of this paper is organized as follows. Preliminaries are presented in Section 2. In Section 3, the proposed methodology is introduced, including the architecture and functions of the OSCMS and an improved consensus mechanism. Then, experiments and results are discussed in Section 4. Finally, Section 5 concludes the paper and lists potential ideas for future work.

2. Preliminaries and Related Work

2.1. Blockchain Technology

Blockchain [14] is a technical solution to collectively maintain a reliable append-only ledger in a decentralized way, and it can provide complete scripts to support various business logics. In a typical blockchain system, data are stored in blocks, which are connected in a chained data structure in chronological order. All nodes participate in the consensus of a blockchain system, including data verification, storage and maintenance.

Blockchain is an ordered, reverse-linked list of transaction blocks based on hash pointers. As the basic structural unit of blockchain, a block is composed of a block header η containing metadata and a block body β containing transaction data. The block body records various parameters of each transaction tsi in detail within time interval $[\tau_{pre}, \tau_{pre} + T_{interval}]$, including the index of tsi in the block, the brief information of the block bi, the information of participants pi and the subject information of tsi. There are generally three types of information stored in the block header: the information maintaining the relationship between blocks re, such as the hash value of the parent block and uncle blocks; a set of metadata describing the production of blocks and block restriction me, such as the hash value and number of the block, the timestamp, the miner's signature and the data about consensus or packaging blocks; and the root structure of the data storage, usually the Merkle root ro.

$$\begin{aligned}
&Transaction_{index} \longleftarrow (index, bi, pi, tsi)\\
&\beta_i \longleftarrow \sum_{\tau_{pre}}^{\tau_{pre}+T_{interval}} Transaction\\
&ro \longleftarrow Merkle(Hash(\textstyle\sum Transaction_{index}))\\
&\eta_i \longleftarrow (re, me, ro)\\
&Block_{(miner\rightarrow i)} \longleftarrow (\eta_i, \beta_i), \quad i = prenumber + 1
\end{aligned} \tag{1}$$

A Merkle tree is used to quickly verify data within the time complexity of $O(\log_2^n)$. The digital signature can ensure that each record comes from the user with a specific private key. Instructions on the simplified verification of blocks and transactions can be found in the Bitcoin white paper [14] and the Ethereum project yellow paper [15]. Figure 1 is a schematic diagram of a block.

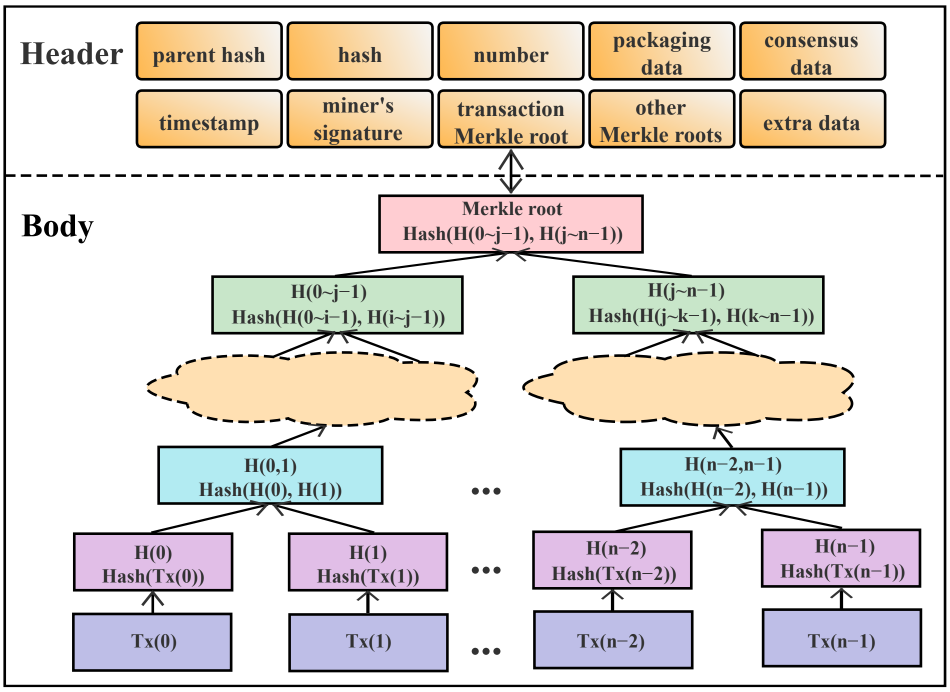

Figure 1. Block structure diagram.

There are three main types of blockchain systems. For public blockchains with open network policies, such as Bitcoin [14] or Ethereum [15], any node can participate in producing blocks and the consensus. It is highly transparent and decentralized, but not suitable for scenarios requiring authentication and control. Introducing varying levels of permission, private blockchain and consortium blockchain are proposed. Consortium blockchain, which still has the sufficient advantage of decentralization, is now widely adopted by various industry alliances. Some mature frameworks in this field include Hyperledger Fabric [16], R3 Corda [17] and Quorum [18] (based on Ethereum). A private blockchain is a strictly controlled blockchain belonging to individuals or a centralized organization. Ethereum and Hyperledger Fabric also provide the development of private blockchain.

A consensus algorithm usually solves the problems of decision-making and consensus-reaching in a distributed system [19]. In a blockchain system, the blocks are validated, shared and synchronized across nodes via a P2P and decentralized consensus mechanism. The first and most popular consensus algorithm is PoW (proof of work) [14], applied in Bitcoin, in which miners need to solve a calculation puzzle. Aiming at mitigating the high cost of PoW, PoS (proof of take) [20] has been proposed. Miners' assets, not their computing power, are used as the criterion. DPoS (delegated proof of stake) [21] is a consensus algorithm that requires stakeholders to vote for delegates. It counteracts the sizeable power of stake-holders that PoS systems entail. In addition, PoS is simplified to obtain PoA (proof of authority) [22], in which predetermined miners pledge their reputation based on their real identity. PoA assumes that these miners are trusted, so it is more suitable for private blockchains. Security blockchain typically uses off-the-shelf algorithms, such as PBFT (practical byzantine fault tolerance) [23], to solve classical byzantine faults. However, PBFT is not effective in terms of dynamics or scalability.

A smart contract [24] is a modular, reusable and automatically executed script running on blockchain that can realize a series of functions, such as data processing, value transfer and asset management. A smart contract not only provides innovative solutions for the financial industry but also plays an important role in the management of information, assets, contracts, supervision and other affairs in the social system.

A smart contract runs in the virtual machine and can only access data on the blockchain. Therefore, a closed blockchain system cannot directly interact with the external environment. Naturally, an open practical problem referred to as oracle has been raised, which is defined as how real-world data can be transferred into/from the blockchain [25], as shown in Figure 2. Centralized oracles are efficient but need to take more risks. In contrast, decentralized oracles (based on consensus), such as Chainlink [26], are realized by aggregating data from multiple independent data sources, so are more reliable.

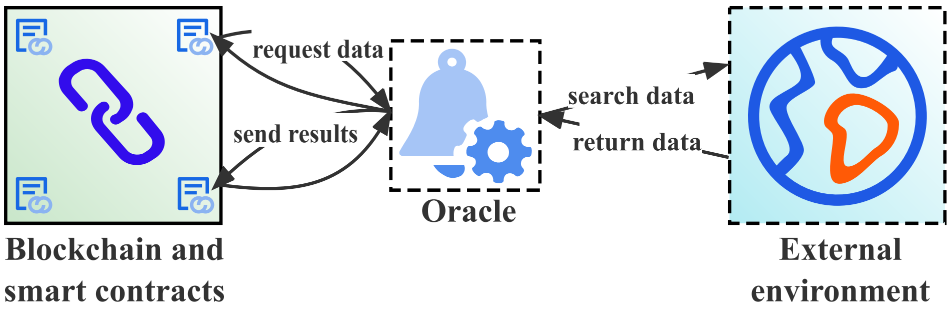

Figure 2. Oracle connects the blockchain with the real external environment.

2.2. Related Work

2.2.1. Traditional Methodology

Many researchers have put forward traditional solutions to address OS issues of code control, incentive, interaction and collaboration, automation, transparency and fairness of rights and responsibilities. Although these solutions do not utilize blockchain to achieve decentralization, it is still important to analyze their achievements and shortcomings.

The technologies involved in code control exhibit diversity. The rapid development of OS has made code referencing complex, and common traceability technologies can be divided into three directions: benchmark databases, signatures and machine learning. By studying the largest corpus of publicly available source code, the Software Heritage archive, Guillaume et al. [4] benchmarked different data models for capturing software provenance information. However, the pressure on the centralized operation and maintenance of such a massive amount of data is enormous. Based on the code database, Woo et al. [27] studied the identification of modified OSS reuse and adopted a signature-based clone detection method. However, they did not discuss reliable traceability. Machine-learning-based code traceability has better generalization and transfer capabilities. Michael et al. [28] presented the spojitR toolset and workflow, which can assist developers in creating trace links between the commits of version control systems and issues. Yet, spojitR can only establish project-wide traceability, and sufficient data are needed for machine learning.

According to the study on incentive factors [29–31], community experience (policy, innovation and communication) and harvest (knowledge, reputation and donation income) are significant. In OS ecosystems, there are many behaviors that damage the incentive mechanism [27], involving infringement, unfair competition and using loopholes to damage the value system. Good experience and reliable harvest come from the consensus of most participants, which also requires research on the credibility of the processes and results.

Liao et al. [32] explored OSS management from problems' feedback, but the perspective was single and did not consider the historical tracing of problems and related modifications. The authors of [33] studied the collaboration and communication of OS developers from a higher-level perspective, which can improve efficiency, but did not consider trust. Zhang et al. [34] summarized the common structure and evolutionary characteristics of OS community developer collaboration models and provided relevant management suggestions based on sociology. However, there was insufficient consideration for the joining and exiting of personnel.

There are many automation tools in the field of software engineering, such as DevOps, which reshape software processes through automation [35]. Automation problems in OS are often related to development, testing and delivery. Some scenarios unrelated to software engineering in OS governance can also be automated, such as payment automation. In [36], automation was considered a good strategy to govern OS. Although lots of OS or commercial toolsets are available to drive automation, transparency and trustworthiness are always ignored.

In [37,38], some existing organizational forms of OSS community governance were discussed. There are two main leadership modes: top-down, such as sponsor-based Redis Labs and tolerant dictatorship-based Linux in the past; and bottom-up, such as business alliance-based GENIVI Alliance and Eclipse Foundation-based PolarSys. Many OS communities at different stages actually have mixed modes. However, in these so-called democratic governance modes, the decision-making process lacks transparency and fairness, and the participation of ordinary members is very low. This is limited by the inability to fully achieve decentralization. The existence of elites is conducive to the rapid development of OS ecosystems, but their rights and responsibilities must be transparent and supervised.

Although the above research attempts to address some OS issues from multiple perspectives, there are still potential problems, including difficulty collecting information, transparency, fairness and trust. Promoting the transformation of OS community ecosystems to decentralization can further solve the above problems. As a decentralized and shared distributed ledger [39], blockchain has the advantages of being tamper-proof, having trusted traceability, the automatic execution of smart contracts, etc. It can not only integrate various incentive policies and promote multi-party collaboration but also solve the problems of transparency, fairness and trust. Fully engaging dedicated OS users in community governance is the way forward.

2.2.2. Blockchain-Based Methodology

Using blockchain can protect the source code from tampering and provide reliable traceability of records. Bose et al. [40] proposed an extensible framework based on standard provenance model specifications and blockchain technology for capturing, storing, exploring and analyzing software provenance data. In [41], a public malware verification ecosystem based on blockchain was proposed to provide decentralized, transparent and immutable software audit management. However, experienced members are required to participate in the verification or the audit, with a high workload and cost.

The value drivers supported by blockchain (a reputation-value system, data ownership, verification and tracking) [42] meet the incentive needs of OS governance. Zeng et al. [43] designed a blockchain-based OS contribution protection system that realizes the conversion of developers' contributions to intellectual properties and records their cumulative interests in the form of tokens. In [44], a distributed autonomous software organization model was proposed in combination with blockchain, which provides a continuously running virtual organization for the software community and users, improving incentives and the community experience through financing, citation, reputation, etc. There are few discussions on contribution, evaluation or inter-organization. The evaluation vulnerability may lead to the loss of credibility of the converted token or reputation.

Blockchain has reshaped the trust model in the organization, trustlessness; thus, interaction and collaboration are easier. Canidio et al. [45] developed a fair bug bounty market based on blockchain to complete bug submissions and reward feedback without mutual trust. Combined with blockchain and open intercloud architecture, Qayum et al. [46] designed a self-evolving and coordinated OS community architecture, including organization management and team coordination. However, they ignored blockchain permissions and identity authentication in actual OS ecosystems. Too strict restrictions will affect the activity, and too relaxed policies will lead to the proliferation of malice.

Blockchain and related technologies, such as smart contracts and oracles, can improve the transparency and trustworthiness of automation in software development, testing and delivery. Singe et al. [47] proposed a system framework using blockchain and smart contracts to capture development activities and automatically evaluate compliance with governance policies. Michal et al. [48] proposed a blockchain-based platform for outsourcing software among parties that distrust each other that can perform automatic tests and reward payments. However, it did not effectively solve the free rider behavior in the software reward. Requirements such as automatic testing and automatic delivery of OS rewards also exist in OS scenarios. OS code is public, so each process of OS reward must be coordinated to prevent others from stealing codes and obtaining the reward.

The inherent decentralization of blockchain drives the rights and responsibilities in the governance process to be transparent and fair. In [49], a blockchain-enabled governance framework was proposed to build trustworthy software. Recording, monitoring and compliance analyses are mostly performed by pre-agreed smart contracts, which bring transparency and auditability. However, this framework is oriented to regulatory organizations and does not discuss fairness. Li et al. [50] proposed a blockchain-based crowdsourcing framework that replaces managers through smart contracts and realizes various policies with enhanced trustworthiness and fairness. Although the arbitration committee is randomly selected from the miners to resolve the dispute over evaluation, the consensus process and the credibility of the selected miners were not discussed.

From the perspective of architecture, some studies tend to give too many functions to a single blockchain. Not only does this limit the performance, but the consideration of relationships and coordination is also lacking. Multi-blockchain architecture is proposed to solve some defects of single-blockchain architecture and problems in practical applications. Many researchers apply multi-blockchain architecture in agriculture [51], medical treatments [52], smart cities [53], etc. As far as we know, there are few studies on the application of multi-blockchain architecture in OS.

3. Proposed Methodology

3.1. Overview of Proposed System

This paper designs an open-source coordination management system (OSCMS) using a novel triple-blockchain architecture for decentralized OS governance. To address the current and future issues of OS project management and OS ecosystem governance, the OSCMS provides a comprehensive range of solutions for the decentralized transformation of an OS ecosystem and is also a fair and transparent community platform. The triple-blockchain architecture is located at the upper layer of the underlying blockchain modules, directly adopting mature underlying technologies such as state databases, virtual machine engines and communication protocols. The form of consortium blockchain is selected to avoid unlimited low-cost account creation, organize major OS communities and introduce necessary control. In the triple-blockchain architecture, OIChain (open-source information chain) is used to record OI (open-source information) of various versions, CFChain (citation–feedback chain) is used for citations, feedback and ratings, RSChain (reward–sponsorship chain) is used for transactions involving funds such as sponsorships and rewards. These decentralized OS businesses are controlled by smart contracts, which are tamper-proof, traceable and more automated.

The above three blockchains jointly maintain the system. They are not only related to each other but also need to conduct necessary interoperability with the external environment, that is, different OS communities and their related tools. On the one hand, different blockchains bind and cooperate with each other only through specified queries. This process is referred to as pre-verification and will be detailed using the specific case in Section 3.3.2. The pre-verification proposed in this paper is a blockchain interoperability scheme based on the specific multi-blockchain architecture, which no longer requires complex or expensive cross-chain technologies. On the other hand, the OS community is the external environment with which OSCMS interacts through an oracle. They have large code warehouses and provide some functions that OSCMS also needs, such as authentication and continuous integration.

Oracle servers are not strictly part of OSCMS. There are already some mature and reliable oracle services collaborating with major blockchain platforms and providing services, such as Oraclize and Chainlink. Regardless of the specific service provider, they are collectively referred to as oracle in this paper. The oracle server is connected to the three blockchains in the OSCMS through pre-deployed oracle smart contracts. When a blockchain needs to perform external queries, this contract will be called. When the oracle server receives the oracle contract calls, it performs the specified queries and returns the results. The interoperability between triple-blockchain architecture and the external environment includes calling community APIs to query information about the community account and code warehouse and calling the API provided by the automated test tool in the code warehouse to query the test report. For the latter, some continuous integration tools support the automatic testing of the code in the warehouse, such as Travis CI, Jenkins, CodeShip, etc. Continuous integration is one of the software development practices. Before the code is integrated into the backbone and a new software version is created, it must pass automated testing [54]. Different OS code hosting platforms may use different tools to support continuous integration. For instance, Travis CI is a common system for CI and deployment. Of the continuous integration projects in GitHub, more than half use Travis CI to automatically build and run tests. In the OSCMS, the code reward service needs to query the automatic test report to the API provided by the continuous integration tool in the code warehouse through an oracle. Continuous integration tools are not strictly part of the OSCMS. The specific tool used depends on the community platform.

An improved DPoS consensus is used in triple-blockchain architectures, optimized for content such as elections, multi-blockchain architectures and incentives. The consensus mechanism is elaborated in Section 3.5.1.

Figure 3 shows the overall schematic structure of the OSCMS.

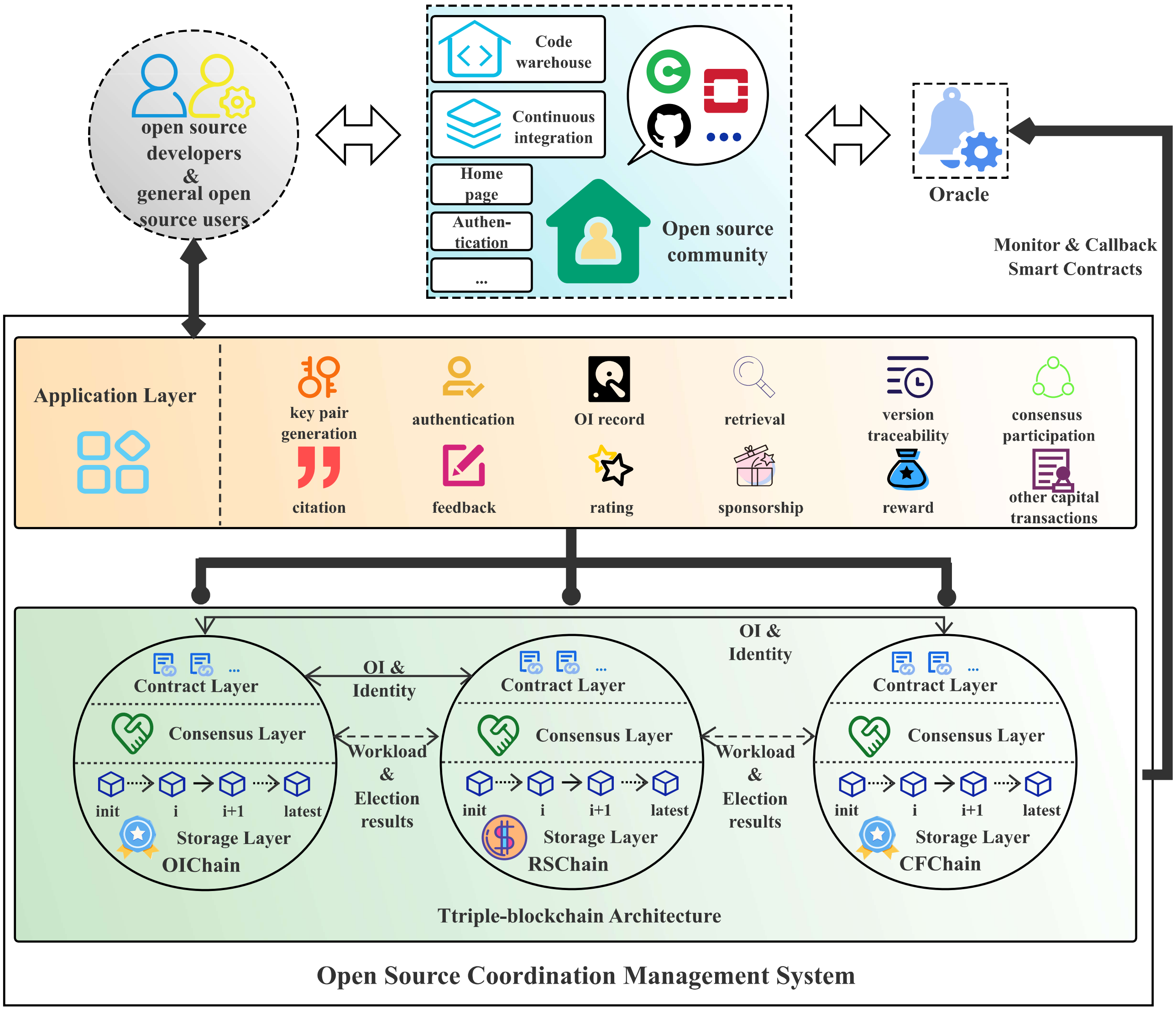

Figure 3. The overall schematic structure of open-source coordination management system.

The advantages of adopting a triple-blockchain architecture are as follows. Firstly, different businesses require different permissions and store different information, so dividing them by type can facilitate management. Secondly, compared with a single blockchain, the triple-blockchain architecture can significantly improve business processing efficiency. Meanwhile, different blockchains can be configured with different parameters to adapt to different business throughputs. Finally, the three blockchains interact only through interfaces to better cope with risks and possible future evolution. Users can choose to maintain one or two blockchains locally to save resources.

3.1.1. Identity Authentication

Due to the need for node access, functions or consensus permissions in the OSCMS, it is necessary to design a reasonable identity authentication mechanism. In traditional OS community and project governance, email/SMS or third-party identities, such as a Google account, are commonly used for identity authentication. Meanwhile, users can further upgrade their identity by meeting certain conditions or simply not register their identity and only browse public content. Although this mainstream identity authentication method cannot completely avoid identity forgery, it can reflect the true identity of users to a certain extent and has certain creation costs. In a blockchain system, asymmetric encryption is adopted. The user randomly generates a pr_k (private key) and maps it to generate a unique Pu_k (public key) according to asymmetric encryption, such as elliptic curve cryptosystems [55]. The pr_k must be kept confidential. The key pairs are used for identity management and account security. Furthermore, for a consortium blockchain system, identity authorization is typically achieved through a CA (certificate authority) using PKI (public key infrastructure) technology.

OSCMS is an OS platform based on consortium blockchain and cannot use a central server for identity authentication, similar to traditional communities. The blockchain system itself is anonymous, and key pairs can be generated in large quantities at a low cost. Even if certified by some internationally renowned CA institutions, the relationship with OS code repositories or some permissions in OSCMS cannot be customized. The OSCMS does not have a central manager and therefore cannot establish its own CA. Even if a new CA is established, it cannot be trusted.

OSCMS provides the local generation of key pairs to prevent them from being caught during transmission. Additionally, OSCMS maintains the mapping relationship between the blockchain account (Pu_k) and major community accounts. There is no need to change or create any institutions to obtain usable identities, and it provides a basis for interoperability between OSCMS and multiple major OS communities. During the initial release of OSCMS, it is necessary to collaborate with major communities and utilize facilities such as the code warehouse.

There are three OS identities in the authentication mechanism of OSCMS: *developer*, *general user* and *unidentified user.* If the users have passed the developer authentication of OSCMS, their projects that have been uploaded to OS communities can be recorded in OIChain in the form of digests. The users who have only passed the general authentication of OSCMS can only perform actions—such as make various queries, cite, give feedback, rate, sponsor, post an open-source reward, etc.—that are irrelevant to development. Users who fail to pass the authentication are unidentified and can only query some information. A user needs to put the Pu_k in the OS community profile and register in the OSCMS through a community authentication link. The link can prove that the user has passed the authentication of a community, at least the general user identity corresponding to this community can be obtained in the OSCMS. The user who obtains the developer authentication of this community or provides a certain amount of code there can further obtain the developer identity corresponding to this community in the OSCMS. The OSCMS provides authentication for a variety of OS communities by different $CommunityID$. A user can authenticate in the OSCMS many times to bind the Pu_k with all community accounts. However, a community account that has been bound in the OSCMS cannot be used for authentication again. For example, a user has successfully authenticated the developer identity of community 1 and the general user identity of community 2 in OSCMS but failed to authenticate community 3:

$$Pu_k \Leftrightarrow \begin{cases} 1 & \Leftrightarrow & \{authenticationLink_1, dev\} \\ 2 & \Leftrightarrow & \{authenticationLink_2, gen\} \\ 3 & \Leftrightarrow & \{null, null\} \end{cases} \tag{2}$$

This process will be completed through a smart contract and an oracle, as shown in Figure 4.

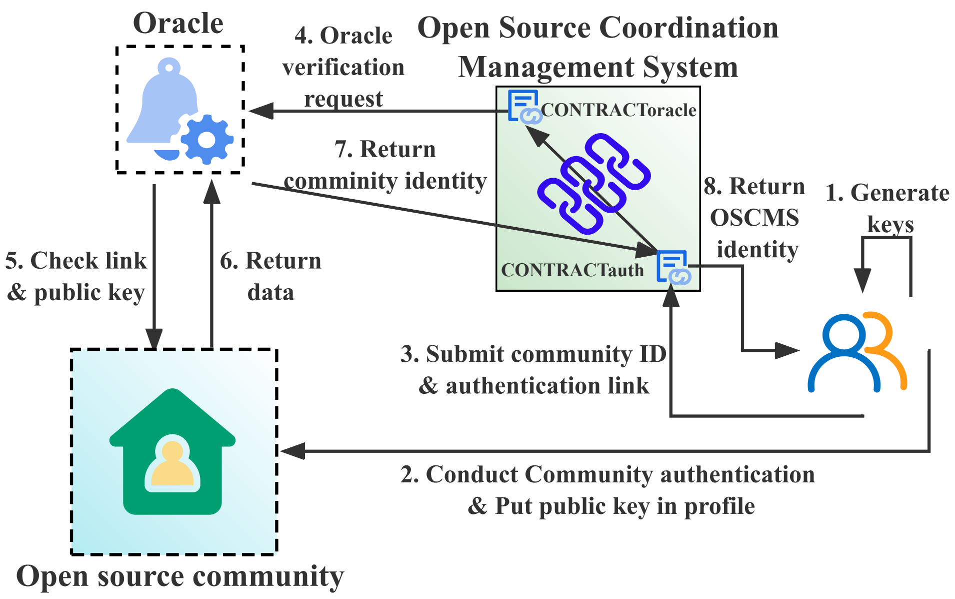
Oracle
4. Oracle verification request
Open Source Coordination Management System
CONTRACToracle
7. Return comminity identity
8. Return OSCMS identity
1. Generate keys
CONTRACTauth
5. Check link & public key
6. Return data
3. Submit community ID & authentication link
2. Conduct Community authentication & Put public key in profile
Open source community

Figure 4. The process of user authentication in OSCMS.

Algorithm 1 describes how to register an account in OSCMS and cooperate with the OS community through a smart contract and an oracle to authenticate identity. The authentication contract is only deployed in OIChain.

Algorithm 1 Register and authenticate in OSCMS

```
Input: $a\,random\,seed, CommunityID, user's\,community\,authentication\,link$
Output: $OSCMS\,authenticationResult : developer, general\,user\,or\;\;null$
1: $pr_k \leftarrow seed$
2: $Pu_k = pr_kG, E_p(a,b)$
3: $CONTRACT_{auth} : FUN_{auth} \longleftarrow CommunityID\,\&\,link$
4: if $boundLink[link] == true$ then
5:     return $null$;
6: end if
7: $CONTRACT_{Oracle} : FUN_{auth} \longleftarrow Pu_k\,\&\,CommunityID\,\&\,link$
8: external: Oracle verification
9: $CONTRACT_{auth} : FUN_{callback} \longleftarrow verificationResult$
10: $Record$ mapping relationships between $Pu_k$ & $[CommunityID, link, authenticatedidentity]$
    in $CONTRACT_{auth}$
11: return $authenticationResult$;
```

Algorithm 2 describes how an oracle service assists with a smart contract for verification.

3.1.2. Computing and Storage of Open-Source Reputation

The decentralized reputation is a quantitative metric to measure a user's contribution to open source and plays a role in incentives and references. In the OSCMS, voters can refer to it when electing miners, which is mentioned in Section 3.5. Communities or organizations can also refer to it to give developers varying degrees of privilege and reward.

The reputation indicator R_T is composed of the quantity of OI submitted by the user to the OIChain and the rating of each submitted OI harvested in the CFChain. The specific process is mentioned in Sections 3.2 and 3.3.

Each time an OI is submitted to an OIChain, the user's development reputation R_{dev} will increase by 1 point:

$$R_{dev} = n_s \times 1, \quad n_s \in N \tag{3}$$

Algorithm 2 Oracle assists with smart contract for verification

```
Input: $Pu_k$, $CommunityID$, $user's\ community\ authentication\ link$
Output: $verificationResult : developer, general\ user\ or\ \ null$
1: while Oracle service is running do
2:     $Monitor\ CONTRACT_{Oracle}$
3:     if $CONTRACT_{Oracle} : FUN_{auth}$ is called then
4:         $Query$ by $CommunityID$ & $link$
5:         $Get\ communityIdentity$ & $Pu_k^{'}$ in the profile
6:         if $communityIdentity == null \parallel Pu_k^{'} = null \parallel Pu_k^{'} \neq Pu_k$ then
7:             return $null$ to $CONTRACT_{auth} : FUN_{callback}$;
8:         else if $communityIdentity == developer$ then
9:             return $developer$ to $CONTRACT_{auth} : FUN_{callback}$;
10:        else
11:            return $general\ user$ to $CONTRACT_{auth} : FUN_{callback}$;
12:        end if
13:    end if
14: end while
15: return ;
```

Users are allowed to rate a submitted OI in the CFChain, which can be 1 or -1. The initial OI rating r_{OI} of an OI is 10 points, and r_{OI} cannot be lower than 1 point. Then, each r_{OI} is transformed into an OI reputation R_{OI} by a logarithmic function. Furthermore, all the R_{OI} are added up to obtain a user's total reputation R_{OItot} about an OI rating. Such a design encourages developers to submit high-quality OS projects to obtain a higher rating. For example, if an OI obtains 90 r_{OI} points, it will be equivalent to the reputation of submitting a new OI.

$$r_{OI} = \begin{cases} 10 + \sum_0^{n_p} 1 - \sum_0^{n_n} 1 & , \quad (10 + \sum_0^{n_p} 1 - \sum_0^{n_n} 1) \geqslant 0 \\ 1 & , \quad (10 + \sum_0^{n_p} 1 - \sum_0^{n_n} 1) < 1 \end{cases}, \quad n_p \in N, n_n \in N \tag{4}$$

$$R_{OI} = log_{10}^{r_{OI}} - 1 \tag{5}$$

$$R_{OItot} = \sum_{0}^{n_s} R_{OI}, \quad n_s \in N \tag{6}$$

$$R_T = R_{dev} + R_{OItot} \tag{7}$$

where n_s is the quantity of the OI submitted by the user, and n_p and n_n are the positive and negative ratings of an OI, respectively.

It is necessary to effectively limit the low-cost growth of reputation. Firstly, the OSCMS requires that only the developer can submit the OI of a project stored in the corresponding community, and the same OI cannot be submitted repeatedly. If the project is copied from others or the quality is low, it has a high probability of having a lower rating in the CFChain, offsetting the reputation of submitting the OI. Secondly, one user can rate an OI at most once, and unidentified users cannot rate it. Since a community account can only be bound once in the OSCMS, only by authenticating more community accounts can an OI be rated multiple times, but this process has costs. In the future, it will be possible to detect similar code or low-quality code through enhanced technology. Due to the inherent characteristics of open source, it is recommended to combine the transformed reputation with other factors for reference, such as democratic elections.

After collection, verification and transformation through related smart contracts and blockchain system codes in the OIChain and CFChain, respectively, the reputation is stored in the form of a Merkle Patricia tree (MPT). The MPT combines the advantages of a Merkle tree and a prefix tree and can compress and store key-value pair data of any length, quickly calculate the hash of the maintained dataset, provide fast state rollback and provide Merkle proof to simply verify data.

The structure of reputation MPT is shown in Figure 5. The key-value pair is $\langle Pu_k, reputation\rangle$. The time complexity of retrieving, inserting and modifying MPT is $O(\log(n))$.

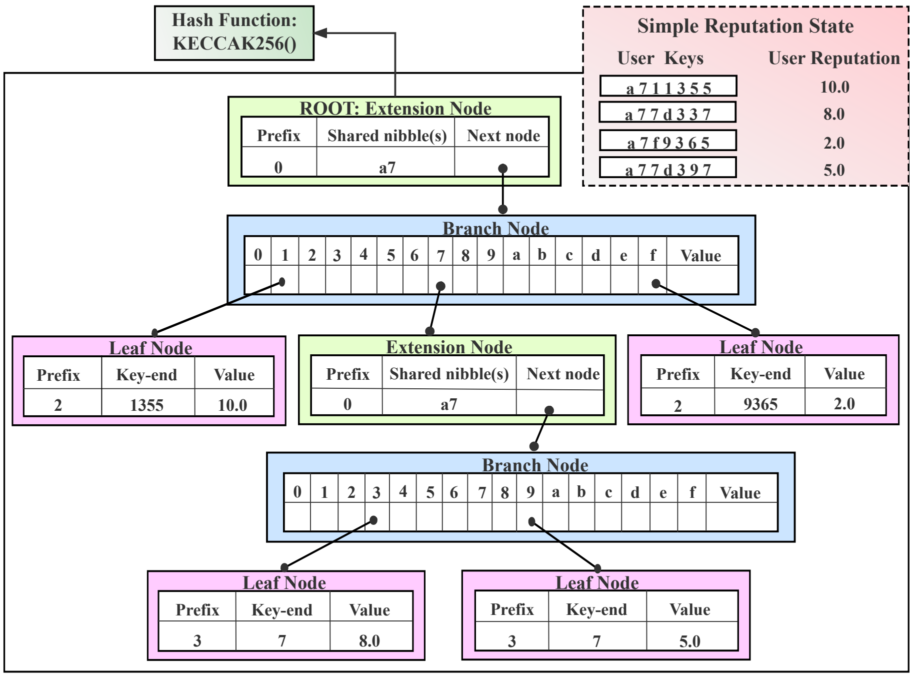

Figure 5. Structure diagram of reputation MPT tree.

3.2. OIChain

In triple-blockchain architecture, the OIChain is the core blockchain for managing open-source information.

3.2.1. Contents of the Stored Data in OIChain

An OIChain is used to coordinate and manage massive projects stored in the warehouses of major OS communities and uses the hash algorithm to obtain the digital digest of them as verifiable information. The hash algorithm maps binary plaintext strings of arbitrary lengths to shorter hash values. Different plaintexts are difficult to map to the same hash value. SHA3 or other secure hash algorithms are recommended.

$$Digest_{(author \to project)} \longleftarrow SHA3(project) \tag{8}$$

In addition to the digest, an OI submitted to an OIChain also needs a $CommunityID$, the author, the link of the warehouse (as an ID that will not be repeated in the global), the version number, the link ID of the previous version (0 if there is no previous version), the timestamp submitted to the warehouse and some other description. It is also necessary to provide numbers $num_{(Pu_k \to OI)}$ for all the OIs submitted by a user in order. This user's Pu_k will be recorded automatically. If the verification passes, the mapping relationship will be formed and stored:

$$\{CommunityID, Digest, author, version, prelinkID, timestamp, Others\} \longrightarrow linkID \longrightarrow \{Pu_k, num_{Pu_k \to OI}\} \tag{9}$$

$$OIstruct = \{\underline{\underline{Pu_k}, num_{(Pu_k \to OI)}}, \underline{linkID}, CommunityID, Digest, author, version, prelinkID, timestamp, Others\} \tag{10}$$

where underlines mark the keys that can be used for retrieval.

An OIChain also needs to store the identity authentication results and development reputation.

3.2.2. Module Design of OIChain

Users submit OI through the pre-deployed OI management contract. It needs to be verified through a smart contract and submitted to an oracle for further verification. This process is shown in Algorithm 3.

Algorithm 3 The entire process of OI submission by the user

```
Input: $OIstruct$
Output: $true$ or $false$
1: $CONTRACT_{OI} : FUN_{submit} \longleftarrow OIstruct$
2: if $identity[Pu_k, CommunityID] \neq developer \parallel prefix(linkID) \neq prefix(authenticationLink_{CommunityID}) \parallel existLink[linkID] == true \parallel prelinkID \neq 0$ && $CONTRACT_{OI} : FUN_{queryOI}(prelinkID) == null$ then
3:     return $false$;
4: else
5:     $CONTRACT_{Oracle} : FUN_{OI} \leftarrow OIstruct$
6:     external: Oracle verifies $linkID, Digest$ & $timestamp$
7:     $CONTRACT_{auth} : FUN_{OI} \leftarrow verificationResult$
8:     $Record\ OIstruct$ & various mapping relationships
9:     $Update\ R_{dev}$
10:    return $true$;
11: end if
```

After identity authentication, a user has maintained the corresponding relationship between the identity in communities and Pu_k in the OIChain. Therefore, it is simply needed to compare the prefixes of the links submitted twice (generally community domain name + user name) after confirming the corresponding developer identity. The contract also checks whether a $linkID$ is repeatedly submitted and whether the OI to which the non-zero $prelinkID$ points is meaningful (the map of some programming languages cannot determine the existence of the value through the existence of the key). With the assistance of the oracle, the last verification is to check the correctness of the $linkID$, the digital digest and the timestamp submitted to the warehouse. These checks can prevent the same project in a warehouse from being reused or used by others.

Any user can query the total OI amount and the last submitted $OIstruct$ of a developer through Pu_k. An $OIstruct$ submitted for the $num_{(Pu_k \to OI)}$th time can be queried through $< Pu_k, num_{(Pu_k \to OI)} >$. After the $OIstruct$ is queried through a $linkID$, users can continue to query the previous version through the $prelinkID$ in this $OIstruct$. The OSCMS provides one-click version traceability, which is shown in Algorithm 4.

Algorithm 4 Version traceability process of OI

```
Input: $linkID$
Output: $\Theta$ //set of $OIstruct$
1: $OSCMS \longleftarrow linkID$
2: $CONTRACT_{OI} : FUN_{queryOI}(linkID)$
3: $\Theta \longleftarrow OIstruct_{linkID}$
4: while $OIstruct.prelinkID \neq 0$ do
5:     $CONTRACT_{OI} : FUN_{queryOI}(prelinkID)$
6:     $\Theta \longleftarrow OIstruct_{prelinkID}$
7: end while
8: return $\Theta$;
```

3.3. CFChain

A CFChain is an extension blockchain for some basic open-source activities, including citation, feedback and rating, which are designed to be decentralized.

3.3.1. Contents of the Stored Data in CFChain

Software development processes or papers often cite some OS projects. Yet, detailed records are inconvenient to trace, and these cited projects cannot be informed. Citation-related data stored in an CFChain include: the person who cites it, the $linkID$ of the cited OI, the current cite number of this person and this $linkID$, the citation status and other information.

$$CIstruct = \{\underline{\underline{CIPu_k}, num_{CIPu_k \to CI}}, \underline{\underline{linkID}, num_{linkID \to CI}}, CIstatus, Others\} \tag{11}$$

An OS project may have some places that need to be corrected or improved or some things that are difficult to understand. A CFChain provides users with a feedback channel. Feedback-related data include: the person who gave the feedback, the $linkID$ of the OI being fed back, the current feedback number of this person and this $linkID$, the feedback content and other information.

$$FEstruct = \{\underline{\underline{FEPu_k}, num_{FEPu_k \to FE}}, \underline{\underline{linkID}, num_{linkID \to FE}}, FEcontent, Others\} \tag{12}$$

For each OI in an OIChain, users can choose to give ratings through related functions in an CFChain. Rating-related data include: the person who gave the rating, the submitter of this rated OI, the $linkID$ of the OI and the rating result.

$$RAstruct = \{\underline{RAPu_k, OIPu_k, linkID}, RAresult\} \tag{13}$$

CFChains also store each user's OI rating reputation.

3.3.2. Module Design of CFChain

In CFChains, there are pre-deployed contracts to support citations, feedback and ratings.

For users participating in producing blocks and transaction verifications, all three blockchains must be maintained locally. So, another blockchain should not be regarded as the external environment. Since an CFChain only needs to query the OI and identity of the OIChain for verification, it does not need high-cost cross-chain solutions or oracles.

Before the contract verification, the OSCMS adds a pre-verification stage when collecting transactions, which is executed locally without changing the blockchain state. Each verification node needs to call the RPC interface of the OIChain locally to obtain the required OI and identity. If the pre-verification fails, the transaction will be directly discarded and not executed. Since a verification node can trust all blockchains maintained locally, cross-chain verification can be quickly and reliably realized at a low cost. The schematic diagram is shown in Figure 6.

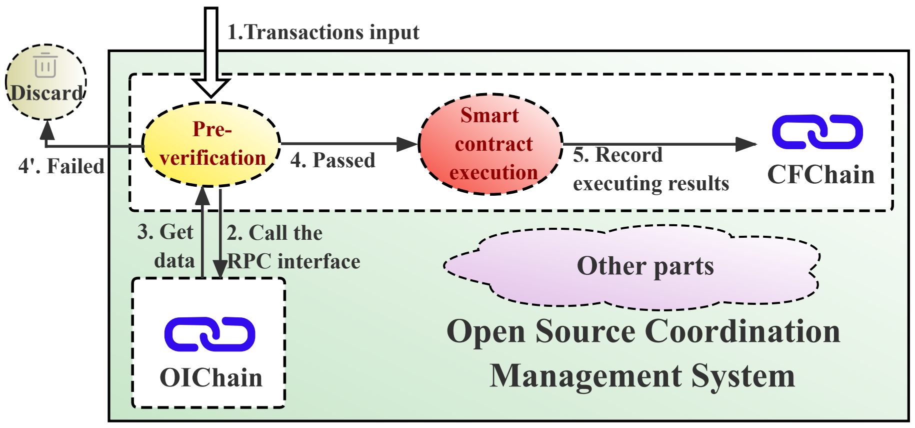

Figure 6. Transaction execution process with pre-verification stage.

The general submission processes of citations, feedback and ratings are similar. Algorithm 5 describes them.

Algorithm 5 The whole process of submitting citations, feedback or ratings

```
Input: $CIstruct \parallel FEstruct \parallel REstruct$
Output: $true$ or $false$
1: if $CONTRACT_{CI} : FUN_{cite}$ is called then
2:     $CONTRACT_{CI} : FUN_{cite} \longleftarrow CIstruct$
3:     pre-verification:
4:         $RPC_{OIChain} \longleftarrow linkID$, $RPC_{OIChain} \longleftarrow < CIPu_k, OIstruct.CommunityID >$
5:         $Get\ OIstruct$ & $identity_{CIPu_k}$
6:         if $OIstruct == null \parallel identity_{CIPu_k} == null$ then
7:             $Discard$ the transaction
8:             return $false$;
9:         end if
10:    $Record\ CIstruct$ & various mapping relationships
11:    return $true$;
12: else if $CONTRACT_{FE} : FUN_{feedback}$ is called then
13:    ... //the whole process is basically the same as citation
14: else
15:    $CONTRACT_{RA} : FUN_{tare} \longleftarrow RAstruct$
16:    pre-verification: ... //in addition to checking whether $OIstruct.Pu_k$ is $OIPu_k$, other
       processes are the same
17:    if $rated[RAPu_k, linkID] == true$ then
18:        return $false$;
19:    end if
20:    $Record\ RAstruct$ & various mapping relationships
21:    $Update\ r_{OI}$ & $R_{OItot}$
22:    return $true$;
23: end if
```

Unidentified users cannot submit any citations, feedback or ratings. Each $RAPu_k$ can rate the same $linkID$ once, but there is no such restriction on citation and feedback transactions. In addition to storing submitted information, various mapping relationships are also stored through smart contracts, which provide different query methods or restrictions. Rating transactions are special and ultimately affect R_{OItot}. The query methods for citation and feedback include: querying how many citation or feedback records an OI corresponding to $linkID$ has and querying a certain item; and querying how many records a user corresponding to $CIPu_k$ or $FEPu_k$ submits and querying a certain item. As for ratings, users can query whether they have rated a $linkID$ and query r_{OI} or R_{OItot}. Authentication is not required for querying.

3.4. RSChain

RSChain is an extension blockchain for security fund transfer business in open-source scenarios, such as decentralized reward and sponsorship. There is no reputation stored in RSChain, but tokens are provided for trading.

3.4.1. Contents of the Stored Data in RSChain

In addition to storing normal fund transfer transactions, sponsorship and rewards are realized in line with OS communities through smart contracts.

Sponsoring excellent OS projects can motivate developers and promote communities. Sponsorship-related data include: the sponsor, the sponsored developer, the $linkID$ of the sponsored OI and the sponsorship amount.

$$SPstruct = \{\underline{SpoPu_k}, \underline{DevPu_k, linkID}, Svalue\} \tag{14}$$

Nowadays, many individuals or organizations use OS code in projects. Posting an OS reward has become a major demand. The OS reward requires multi-party participation. OS demanders provide OS code requirements, test requirements, deadlines and reward amounts. OS developers need to provide the *linkID* of the solution, the timestamp recorded in the OIChain and the test report link. The reward contract records solutions that pass tests and updates the winner faithfully after the deadline.

$$\begin{aligned} R_{dem}struct &= \{DemPu_k, cReq, tReq, deadline, Rvalue\} \\ R_{dev}struct &= \{\underline{DevPu_k}, linkID, timestamp, testlink\} \\ REWARDstruct &= \{R_{dem}struct, []R_{dev}struct, winnerPu_k\} \end{aligned} \tag{15}$$

3.4.2. Module Design of RSChain

Sponsorship and token withdrawal transactions are controlled by the same smart contract, which records the total historical sponsorship amount of an OI or a sponsor and the remaining amount not withdrawn by a developer. Through <$DevPu_k$, $linkID$>, the sponsorship tokens can be deposited into the sponsorship contract. Sponsored developers can withdraw their own tokens from the contract. For the sponsorship transaction, the pre-verification is the same as the rating transaction in the CFChain. Then, the contract verifies whether various variables exceed the numerical range. The withdrawal transaction only needs to be verified when the remaining amount of the user is greater than 0. Any user can query multiple sponsorship mappings recorded in the contract. The decentralized sponsorship process is completely transparent. Once sponsored, only the sponsored user can withdraw it.

There were some studies on code rewards using blockchain. However, there were few simple and practical methods to solve the problems of tampering, plagiarism, insufficient automation, etc. With the assistance of OIChain, RSChain provided a decentralized way for OS rewards that can prevent malicious competitors and murky operations without complex cryptography technology. The schematic process is shown in Figure 7.

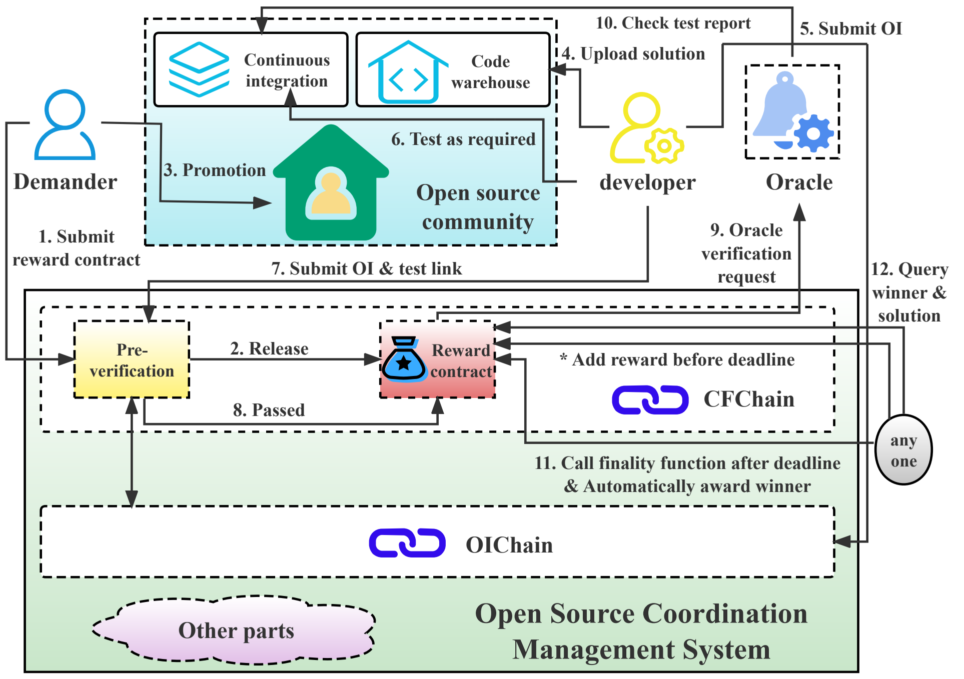

Figure 7. The process of open-source reward.

The demander generates a reward smart contract according to the provided template. Then, it is submitted to the RSChain with the code requirements, test requirements, deadline and reward tokens. Creating a smart contract is also a transaction that needs to be pre-verified to confirm if it conforms to the template and if the identity of the demander is not null. Reward tokens are locked into the contract before the deadline. The demander can upload the contract address to the community for promotion.

Each developer interested in it uploads the solution to the code warehouse and then submits relevant OI to the OIChain. Before completing the solution, the developer does not need to declare their intention to join and the development progress will not be exposed. Therefore, even if it is copied by malicious competitors later, the timestamp submitted to the warehouse will have been recorded in the OIChain and can be regarded as the development completion time within a strict chronological judgment. In addition, OIChain records can prevent codes from being tampered with.

Before submitting the solution (*linkID* and timestamp recorded in the OIChain) and test report link to the reward contract in the CFChain, each developer needs to use Travis CI or other tools for testing in the code warehouse according to requirements. The solution submission transaction in the CFChain needs to be pre-verified for the correctness of the OI and its author. Then, the reward contract will request the oracle to verify the test report link, including whether it corresponds to the *linkID* and whether the test scheme is consistent with the requirements and passes. The result is returned by the callback function. Users can submit a solution or add reward tokens to motivate developers before the deadline.

After the deadline, anyone can call the finality function. The reward contract chooses the solution with the earliest timestamp that passes the tests as the winner and pays the reward to the developer. If no solution passes the tests, the reward is withdrawn according to the source of the tokens. Users can query the reward amount, the number of solutions that have passed tests, whether their solution has passed the tests and the winning solution.

Demanders and developers only need to pay attention to their own reward content or development content; other processes are coordinated automatically. In this decentralized OS reward, the development process is not open, but the flow of tokens and the competition process are transparent. The pseudo code of the reward smart contract template is shown in Algorithm 6.

Algorithm 6 Brief pseudo code of reward smart contract template

```
contract Reward {
    uint $deadline, reward$; string $codeReq, testReq$;
    address $demander, winner$, address[] $passSolutions$;
    mapping (address ⇒ bytes) $dev\_linkID, dev\_testlink$;
    mapping (address ⇒ uint) $dev\_timestamp, investor\_reward$;
    function Reward() {...//Initialize some variables and deposit reward}
    function addReward() {...//Before deadline}
    function commitSolution($linkID, testlink, timestamp$){
        ...//Commit the solution
        $Oracle.requestTest(msg.sender, this, linkID, testlink, testReq)$;
    }
    function finality() {
        $require(block.time > deadline \&\& winner \neq 0x0)$;
        ...//Update the winner and transfer the reward
    }
    function qDemand() {...//Query reward demand}
    function qReward() {...//Query reward amount}
    function qPass() {...//Query how many solutions pass}
    function qIfPass() {...//Query if your solution passed}
    function qWinner() {...//Query the winning solution after the deadline}
    function _ callback($\_requestID, \_result$) {
        require ($msg.sender == Oracle.address()$);
        ...//update passSolutions
    }
}
```

3.5. Consensus Mechanism in OSCMS

PoW is a mature and widely used consensus mechanism that has been fully tested by bitcoin and other blockchain projects for a long time. However, its high cost and requirement that each node verify transactions are not desired by OSCMS. DPoS requires stakeholders to vote for delegates, who are then responsible for verifying transactions and producing blocks. OSCMS chooses this fair and democratic consensus manner, whose advantages are further evaluated in Section 4. However, there are still some limitations.

3.5.1. Limitations and Solutions

Insufficient reference for election. In conventional DPoS, tokens are used to vote for delegates based on the real-world credibility or commitment of candidates, which is often not intuitive or easy to obtain. The decentralized open-source reputation provided by OSCMS can supplement the reference of voting. Users with high reputations have inherent needs to maintain the system and produce blocks.

No support for multiple-blockchain architecture. It is not necessary for all blockchains to carry out the whole process of DPoS. Since they share a set of account identities and have the same epoch cycle, elections can be held once each epoch in an RSChain, while other blockchains try to obtain results. The elected delegates must maintain three blockchains at the same time. The consistency of results is based on the fact that DPoS can ensure the blocks that meet the conditions are irreversible. This design not only saves the cost of repeated voting, but also enables each blockchain to focus on its own work.

Strapped for incentive. In DPoS, the delegates are pre-allocated the time window (round robin) for producing blocks. If a slothful delegate does not produce a block, transactions cannot be verified and processed in time until the time window of the next delegate starts. Using an exponential function reward instead of a linear reward can better encourage delegates to produce as many blocks as possible.

The classic DPoS process [43] is adopted, but it is implemented in the OSCMS through a smart contract to better realize the above solutions.

3.5.2. Delegate Elections

The P2P network, composed of all nodes in the blockchain system, constitutes the whole network environment. The set of all nodes is expressed as N, whose size is $n(n > 0)$, each node is expressed as $N_i(i \in [1, n], N_i \in N)$ and all but unidentified users can be voters. We use N^C to represent the set of all candidate nodes, with its size as $c(c \leqslant n, N^C \in N)$. N^D represents delegate node sets with size $d(d \leqslant c, N^D \in N^C)$, who are miners in this epoch. DPoS requires a $3f + 1$ fault tolerance [56]. We assume the majority of miners (i.e., more than 2/3) are honest and will produce blocks in their time windows (except for when obtaining election results or counting rewards), meaning that the network can operate securely. In the triple-blockchain architecture of the OSCMS, each blockchain carries out a DPoS consensus independently, only utilizing the irreversibility of blocks in DPoS to share election results, without changing the Byzantine fault tolerance and robustness of DPoS. The white paper [21] analyzes the working principle and robustness of the DPoS consensus algorithm, which can be fully utilized by the consensus mechanism in the OSCMS.

Only RSChain needs to set tokens to solve fund transfer transactions, as delegates are elected through tokens in it. The election process includes voting and selecting the top d nodes as delegates. We use a voting smart contract to implement voting, which is pre-deployed in the RSChain. The brief pseudo code is shown in Algorithm 7.

Miners in the first epoch are recorded in three genesis blocks. Without losing generality, we discuss the election process in the k-th epoch, which can be analogized to other epochs.

Algorithm 7 Brief pseudo code of voting smart contract

```
1:  contract Voting {
2:      struct VoteInfo {//Voting information struct of each candidate
3:          uint totalVotedToken;
4:          address[] voters;
5:          mapping (address ⇒ uint) voter_token;
6:      }
7:      struct Record {//Voting information struct of each candidate
8:          address[] candidates;
9:          mapping (address ⇒ voteInfo) candidate_voteInfo;
10:     }
11:     Record currentRecord;//Voting record in the current epoch
12:     mapping (uint ⇒ record) epoch_Record;//Historical records
13:     function Voting() {...//Constructor: Initialize some variables}
14:     function applyCandidate() {...//Deposit fixed tokens as a candidate}
15:     function exitCandidate(){...//Exit candidates, return tokens corresponding to candidates and voters}
16:     function vote(candidate, tokens) {...//Deposit tokens to vote for existing candidates}
17:     function withdrawVote(candidate) {...//Withdraw the vote for a candidate and get the token back}
18:     ...//Some other query methods
```

Step 1: Node $N_i (N_i \in N^C)$ calls the applyCandidate method with a fixed deposit amount. This transaction is broadcast to the blockchain network.

Step 2: Current miners pre-verify it, that is, call the RPC interface of the OIChain locally to query whether N_i has the developer identity. If this transaction is not discarded due to a failure of pre-verification, it will be executed. If N_i is already a candidate or the deposit is not enough, the call fails. After passing, this candidate will be added to currentRecord and epoch_Record.

Step 3: Node N_i can call the exitCandidate method to exit the candidate.

Step 4: If N_i is one of the current miners, the transaction will be discarded by the miners. We avoid the same node producing blocks in two consecutive epochs, so N_i can exit in the next epoch. If N_i is not the candidate, the call fails. Otherwise, the deposit tokens of N_i and relevant voters will be returned, and the candidate and voting records will be deleted.

Step 5: For candidate N_i, any node with identity can call the vote or withdrawVote method repeatedly, and the total deposit of a voter cannot exceed a certain value. Before that, voters can refer to the decentralized reputation of candidates in the OSCMS.

Step 6: If the above transactions are executed successfully, currentRecord and epoch_Record will be updated. Historical records can be used for reward distribution. These four methods are prohibited from being called during the first 3*d* blocks and the last *d* blocks of each epoch. The total amount of tokens voted by all voters for N_i will be counted in totalVotedToken:

$$V_i = currentRecord.candidate\_voteInfo[N_i] \tag{16}$$

$$totalVotedToken = \sum_{V_i.voters} V_i.voter_token[voter] \tag{17}$$

Step 7: When the miner produces the checkpoint (the *d*-th block from bottom in the *k*-th epoch), according to the descending order of totalVotedToken, *d* candidates excluding current miners will be new delegates N^D of the next epoch and recorded in this block header. If subsequent miners obtain the same N^D, they will continue to produce blocks until the epoch switches and new miners start working. The block witnessed by more than two-thirds of the miners is irreversible, so a round of consensus on new delegates can be completed. How to reach a consensus on blocks is introduced in the next section.

Step 8: The previous steps are only repeated in the RSChain. In the checkpoint of the OIChain and CFChain in the k-th epoch, instead of obtaining new delegates by voting contracts, miners try to query them from the checkpoint of the RSChain in the k-th epoch. Before querying this checkpoint, the last block in the k-th epoch must be queried and exist in the RSChain, which ensures that the checkpoint is irreversible. New delegates will be recorded in the checkpoint of the OIChain and CFChain. When a checkpoint block needs to be produced, if the current miner cannot query the last block in the time window, the next miner will try it.

3.5.3. Reaching a Consensus on Blocks

The delegates (miners) in the current epoch should not only take turns verifying transactions and producing blocks, but also verify the content and order of all blocks to reach a consensus. For each blockchain, the module for reaching a consensus can be described as follows:

Step 1: The miner with the current time window collects some transactions in this blockchain network and verifies them. The current block is produced by this miner according to the previous block. If no block is produced in this time window, the miner will be skipped and the next will try to produce it.

Step 2: The current miner broadcasts the signed block to the network, which can be received by all normal nodes.

Step 3: Miners must verify these transactions. Other nodes can choose whether to verify them. In addition, each node needs to verify the validity of the block structure and whether the producer matches the time window. If the verification passes, the current blockchain maintained by this node will temporarily accept the block. Otherwise, the block is rejected.

Step 4: The above steps are repeated. If a miner approves all the previous blocks, it will produce a block pointing to the last one. After d blocks, the earliest block will be witnessed and approved by more than two-thirds of the miners. At this time, the majority of miners have reached a consensus on it, which is irreversible. In fact, this process only requires more than $(2/3)d$ blocks. For the convenience of description, d is still used later. How the blocks in DPoS are irreversible is analyzed in white paper [21].

3.5.4. Incentive Tactic

A general incentive tactic is that each valid block brings fixed rewards. However, the number of blocks produced by a miner does not affect the rewards brought by each block, which is not conducive to promoting miners to produce as many blocks as possible. Using an exponential function reward can improve it.

For each blockchain, the current number of blocks produced by each miner in the k-th epoch is recorded in each block header:

$$miner\_workload = \{\{miner_1, number_1\}, \{miner_2, number_2\}, \{miner_3, number_3\}, ..., \{miner_d, number_d\}\} \quad (18)$$

It is necessary to count each miner's workload, which is the sum of *miner_workload* recorded in the endpoint (the last block in the k-th epoch) of three blockchains. According to Sections 3.5.2 and 3.5.3, the election result recorded in a checkpoint is irreversible after the endpoint and can be used securely. As shown in Figure 8, before the checkpoints in the OIChain and CFChain are produced, it must be confirmed that the endpoint in the RSChain exists. Similarly, after d blocks in the $(k+1)$-th epoch, the *miner_workload* recorded in the previous endpoint can be reached by a consensus of the new miners and used securely. When the $(2d+1)$-th block in the $(k+1)$-th epoch is produced (countpoint) in the RSChain, it must be confirmed that the d-th block in the $(k+1)$-th epoch exists in the OIChain and CFChain. In countpoint of RSChain, the total workload in three blockchains can be counted and recorded. Only when a valid block is produced in each time window will the corresponding relationship between blocks and time windows be as shown in Figure 8. Otherwise, there will be some synchronization waiting, which does not affect the above steps. The mutual confirmation of the block heights keeps the blockchains synchronized during epoch switching, so that the shared election results are within the timelines.

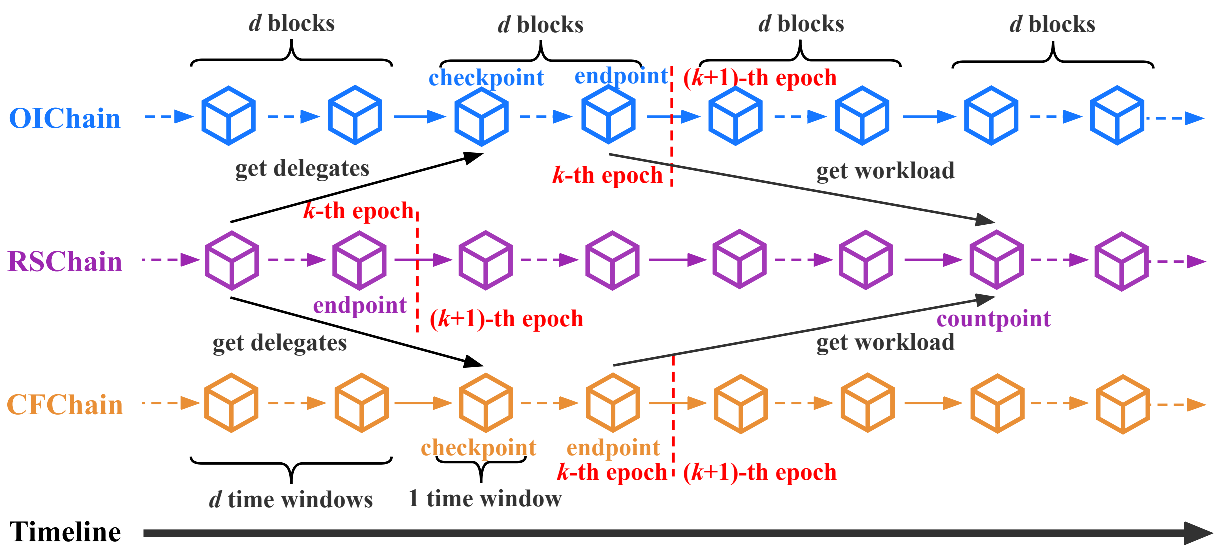

Figure 8. Mutual confirmation among three blockchains during epoch switching.

After a round of consensus on the countpoint of RSChain, that is, after the $3d$-th block in the $(k+1)$-th epoch, tokens are awarded exponentially:

$$reward(m) = \alpha^m \tag{19}$$

where m is the total number of blocks actually produced by a miner in three blockchains in the k -th epoch. α can be calculated by:

$$\alpha^{m_{max}} = \beta m^{max} \tag{20}$$

where β is rewards for a valid block in conventional linear reward schemes. If each miner has produced a valid block in each allocated time window, m will be m_{max}.

The voters of the miner will obtain 20% of its rewards according to the proportion of votes.

Delegates can vote for a malicious or too slothful delegate in blocks they produce and collect votes from other blocks. Once a node has $(2/3)d$ votes in a blockchain, the number of this miner in *miner_workload* recorded in the current block header is -1 and no time window will be allocated to it thereafter. This will reduce some invalid waiting. Although it is only removed from delegates of this blockchain, miners with -1 record will not be rewarded and lose the deposit when counting in countpoint.

4. Experiments and Results Discussion

4.1. Experiment Settings

We conducted a secondary development based on the Ethereum source code of go-ethereum-1.9.25, and established a prototype of the OSCMS in the environment of Go 1.14.2. Based on the modified Ethereum code and solidity 0.4.17, we implemented a consensus and all the functions introduced in this paper, and simulated an oracle service with node.js 10.19.0 to monitor and respond to requests from smart contracts. The prototype and experiments ran on four Ubuntu 20.04.1 (64-bit) virtual machines, an Intel Xeon 5117 @2.0 GHz processor with 14 cores and 32 GB RAM. Each node independently operated the protocol of Ethereum and our system on Docker.

Our experiments explored the feasibility, scalability, reliability and performance superiority of the OSCMS under different parameters, architectures and consensus algorithms. The consumption and steps of different transactions are different. For example, the submission of OI is divided into two steps. After the submission transaction is processed and exists in a valid block, the oracle will submit the verification result in a later block through the callback function, thus updating the OI status. Other transactions involving oracle verification have similar two steps. Our simulation oracle service calls the callback function around 3 s after listening to related events in the network. The speed of real oracle is affected by many factors, which are beyond the scope of our experiments. We select representative submission transactions or query transactions of each blockchain according to a certain ratio. The settings of some conditions and parameters are shown in Table 1.

Table 1. The settings of some conditions and parameters.

Conditions or Parameters	Description	Default
S	Block size, measured by Ethereum's gas	7,000,000 gas
bpi	Block production interval/yime window size	3 s
BPN/TN	The number of block-producing nodes/total nodes	15/150
r	Offline rate of block-producing nodes	0%
epoch	Number of interval blocks for adjusting BPNs	300
A:B:C:D:E	Ratio of submission transactions: A is to submit OI in OIChain (two steps: 101,495 + 31,054 gas); B is to submit feedback in CFChain (32,504 gas); C is to submit rating in CFChain (76,260 gas); D is to sponsor in RSChain (40,063 gas); E is to commit reward solution in RSChain (two steps: 98,517 + 29,802 gas)	1:3:3:2:1
a:b:c:d:e	Ratio of query transactions: a is to query OI from OIChain; b is to query a feedback item from CFChain; c is to query the r_{OI} of a developer's OI from CFChain; d is to query query the total historical sponsorship amount of an OI from RSChain; e is to query reward amount from RSChain	4:2:2:1:1

4.2. Performance Evaluation under Different Parameters

Through a series of experiments under normal throughput or excess throughput, we tried to explore the response time, confirmation time and throughput of submission transactions with different parameter settings. We also tested the change in the average query delay as the number of valid blocks increased. We measured these performance metrics by:

$$\begin{aligned} T_{subResponse} &= \frac{\sum_0^{TN} T_{remove}}{TN} - T_{input} \quad (s) \\ \overline{T_{subResponse}} &= \frac{\sum_0^{N_{tx}} T_{subResponse}}{N_{tx}} \quad (s) \end{aligned} \tag{21}$$

$$\overline{T_{confirmation}} = \overline{T_{subResponse}} + T(\frac{2}{3}BPN\, subsequent\, blocks\, are\, accepted) \quad (s) \tag{22}$$

$$TPS = \frac{Count(tx\ in(t_i, t_j))}{t_j - t_i} \quad (txs/s) \tag{23}$$

$$\overline{T_{query}} = \frac{\sum_0^{N_{tx}} (T_{return} - T_{input})}{N_{tx}} \quad (s) \tag{24}$$

The response time of the submission transaction is the time gap from input to removal (when a single-step transaction or the callback transaction of a two-step transaction last exists in the transaction pools of nodes). The last removal indicates that a transaction and its corresponding block (response block) are accepted by a node after verification. We calculated the average response time for N_{tx} transactions. The confirmation time is the time gap from input to the irreversible response block, which means that $(2/3)BPN$ blocks need to be accepted by nodes after the response block. Throughput is the number of submission transactions that are responded to in a unit of time. When the maximum throughput is exceeded, the response time and confirmation time will be seriously affected. The average query delay is the average of the time gap from inputting the query to obtaining the results of N_{tx} query transactions.

Without exceeding the throughput, we evenly and continuously submitted a set of submission transactions within each bpi. From Figures 9 and 10, we can see that if the bpi is at the second level, the response time or confirmation time is only related to the step of transactions. For transactions with the same number of steps, the average confirmation time tends to increase linearly. However, the impact of the bpi on the average response time is not linear until it is greater than 9 s. Through analysis of specific data, even if the transaction submission frequency is very low, some transactions may not be packaged into the current block. This situation becomes more serious when the bpi is shorter.

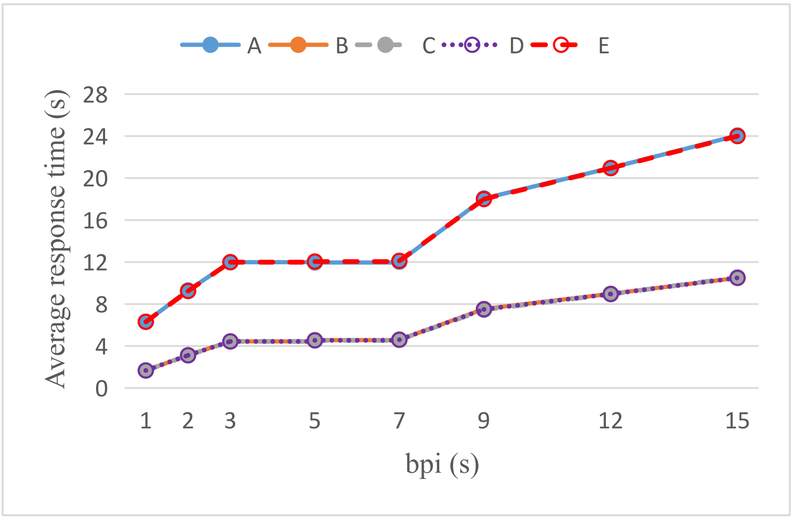

Figure 9. Average response time with variable block production interval.

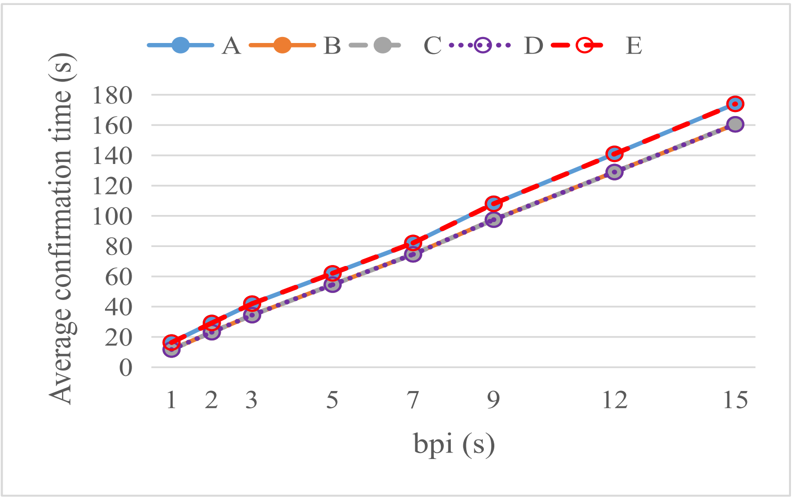

Figure 10. Average confirmation time with variable block production interval.

By submitting a large number of transactions, A, B, C, D and E, at one time, Figure 11 shows that the throughput decreases as the bpi increases. Different transaction consumption leads to different throughput of each blockchain. We can find that appropriately shortening the bpi effectively reduces the average response time and confirmation time and increases the throughput.

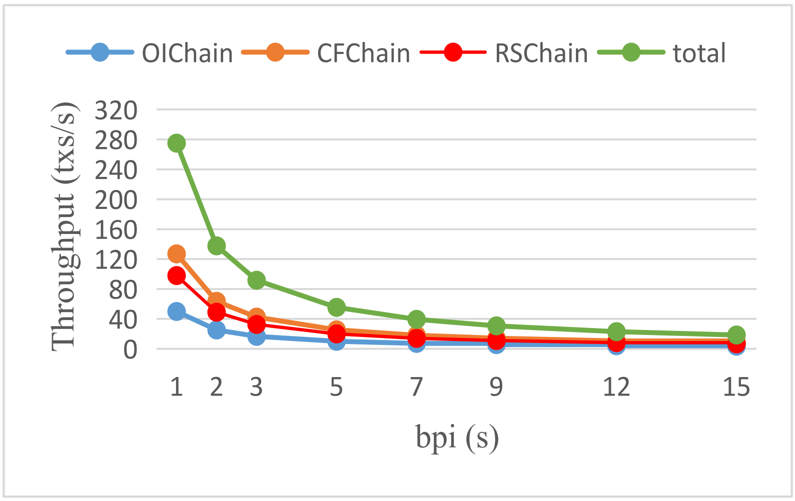

Figure 11. Submission throughput with variable block production interval.

For an appropriate bpi, the average response time and confirmation time should not be too long, and the throughput should not be too low. However, the block size will also affect the throughput, which should not be increased by excessively shortening the interval. Too short an interval is not conducive to the necessary steps, such as verification, execution, signature, network propagation, etc. Therefore, we recommend setting the block production interval to 3 s.

By submitting a large number of transactions, A, B, C, D and E, at one time, Figure 12 shows that the throughput can be linearly increased by expanding the block size. Ethereum can control the block size through a unique dynamic algorithm. In short, if the current block size exceeds the threshold, then the maximum limit will be increased; otherwise, it will be reduced. The initial block size of the Ethereum main network is far from the current size. If this dynamic control algorithm is used, we do not have to set a larger block size at the beginning. To control the variables in subsequent experiments, we fixed the block size to 7,000,000 gas.

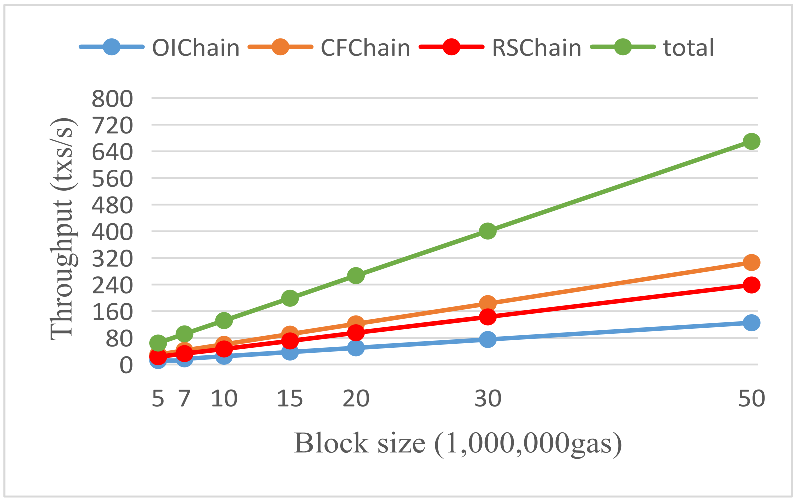

Figure 12. Submission throughput with variable block size.

Without exceeding the throughput, Figure 13 shows that the average response time is negligibly affected by the number of BPNs or total nodes, whether for one-step transactions or two-step transactions. Although the network propagation delay increases with the increase in nodes, even in the Ethereum main network, the average network propagation delay is less than 0.3 s. However, the average confirmation time increases linearly with the number of nodes. This is due to the increase in BPNs (delegates), which leads to a longer consensus time for a block in DPoS. Even with 21 delegates, the average confirmation time is shorter than that in PoW (Bitcoin takes one hour and Ethereum takes six minutes [56]). The DPoS consensus is partially scalable for the node network scale, and there can be a lot of ordinary nodes, but the number of delegates is not recommended to exceed 21. Since only a few delegates are required to verify transactions and produce and confirm blocks without competition, the DPoS consensus is scalable for the transaction scale and the throughput is not affected by the total number of nodes.

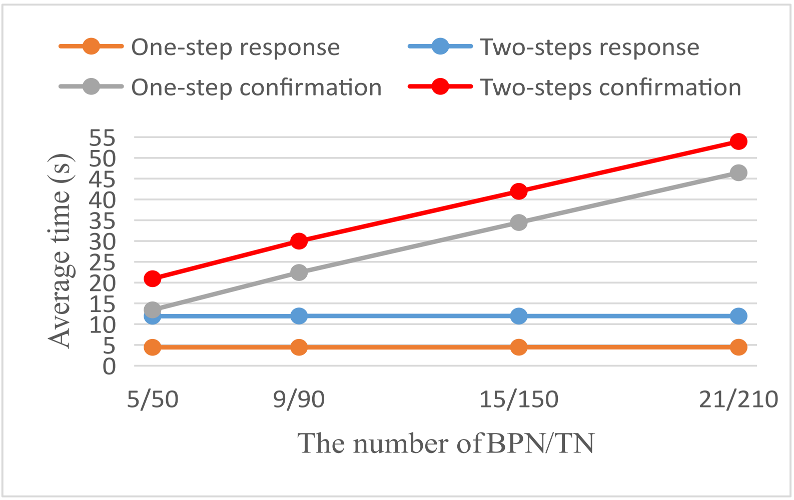

Figure 13. Average response time and confirmation time with variable number of nodes.

To explore the impact of block-producing nodes offline, zero to four of them are taken offline in turn (the system will no longer be secure when more than one-third of the BPNs are offline). By submitting a large number of transactions, A, B, C, D and E, at one time, Figure 14 shows that the throughput decreases linearly with the increase in offline BPNs. In the worst case, it decreases by a quarter. Without exceeding the throughput, the test results of the average response time and confirmation time are shown in Figure 15. These times all increase with the increase of offline BPNs, and the increasing trend is also rising. If more transactions are performed, the trend will be faster. Because transactions committed within the time windows allocated to offline BPNs cannot be processed in time, active BPNs are allowed to remove lazy BPNs from delegates by voting, thereby reducing these effects. Before the completion of the voting adjustment, the performance is still acceptable and better than that in PoW when a few BPNs are temporarily offline.

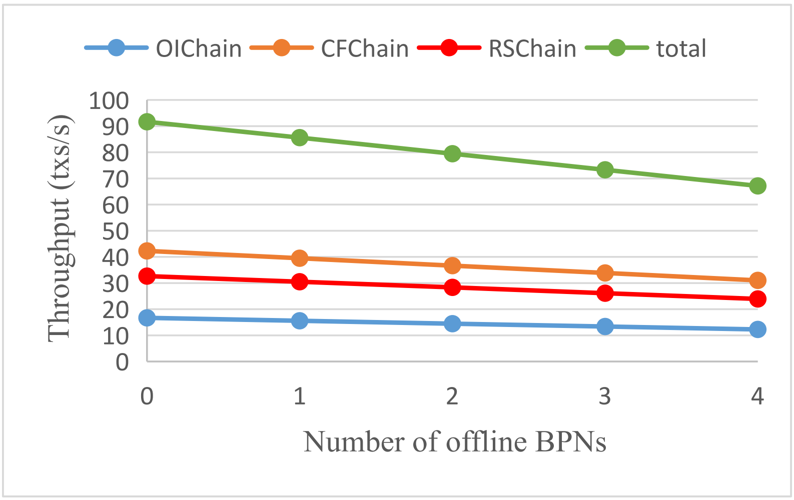

Figure 14. Submission throughput under different offline rates of BPNs.

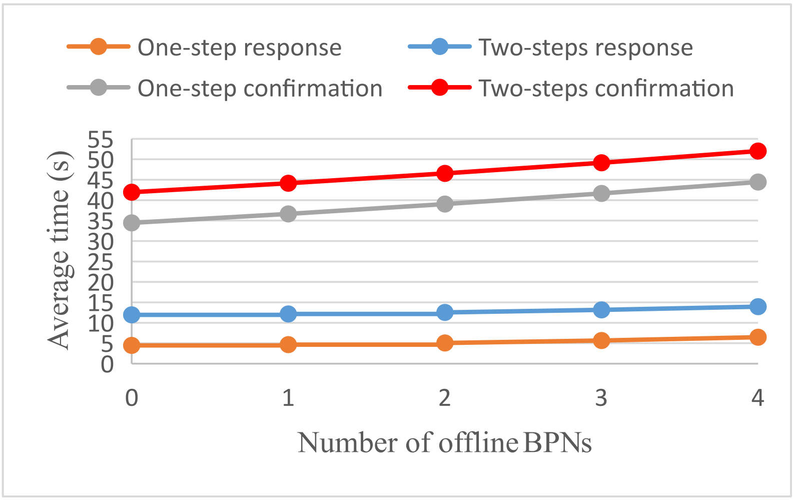

Figure 15. Average response time and confirmation time under different offline rates of BPNs.

Query transactions are not recorded in the blockchain, so the query delay is very short. Figure 16 shows that the average query delay will not change significantly as the block height increases, no matter which query transaction. Traversal and traceability retrieval are not included because they are implemented through multiple calls to query interfaces. Hardware or network factors may lead to the fluctuation of the single query delay or the average query delay over a period of time, but through a number of repeated tests, the average query delay is stable between 480 ns and 500 ns.

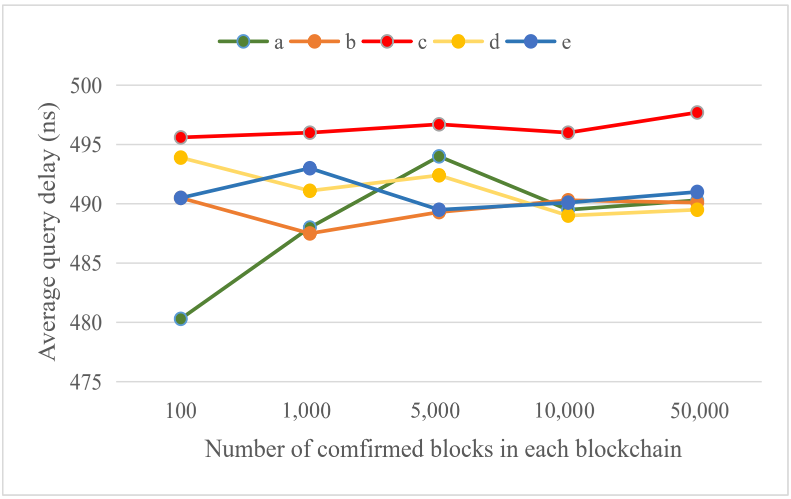

Figure 16. Average query delay as block height increases.

4.3. Comparison of Different Architectures

In addition to the scheme in this paper, three other common or possible architectures are given for performance comparison:

Architecture I: Single-blockchain architecture based on PoW consensus. The average bpi of PoW in our experiments was 12–13 s, referring to the Ethereum main network.

Architecture II: Single-blockchain architecture based on DPoS consensus (one election per epoch).

Architecture III: Triple-blockchain architecture based on DPoS consensus (each blockchain holds elections separately).

Architecture IV: Scheme in this paper (three blockchains share the same election results).

By submitting sufficient OS transactions (A, B, C, D and E) and necessary election transactions (applyCandidate, exitCandidate, vote and withdrawVote) in an epoch, Figure 17 shows the OS transaction throughput under each architecture. We simulate a case where each node submits two election transactions to the blockchain that holds elections in an epoch. Although there are no election transactions in PoW, the OS transaction throughput in DPoS-based blockchains is higher. Compared with single-blockchain architecture, triple-blockchain architecture can increase throughput. Architecture IV only retains elections in the RSChain, which further increases the OS transaction throughput. It seems that there is not much improvement because there are only 150 nodes in the experiment, but the number of active users in the OS ecosystem is far more than this. Therefore, in the real world, Architecture IV can reduce a lot of unnecessary election costs. Although multi-blockchain architecture is highly scalable, the OSCMS does not provide too many services at present, and unnecessary blockchain splitting will increase the time cost and resource consumption of cross-chain.

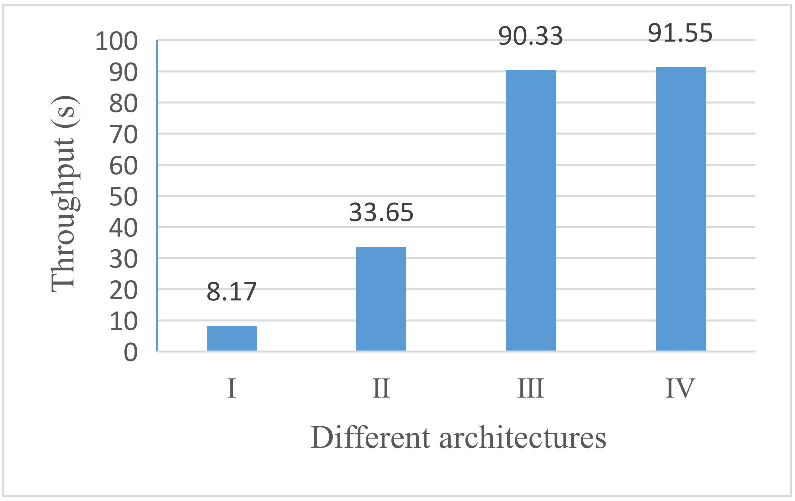

Figure 17. OS transaction throughput under four architectures.

4.4. Comparison of Different Consensus Algorithms

The top three consensus algorithms most widely used in cryptocurrencies are PoW, PoS and DPoS [52]. PoW is highly secure and scalable for the node network scale, but it has high energy consumption and loses scalability on the transaction scale. PoW-based blockchains take considerable time to confirm transactions by accumulating blocks. Peercoin [20] is the first cryptocurrency (blockchain system) to formalize the notion of PoS, which utilizes coinage and limited search space to reduce energy consumption. So far, many other PoS consensus algorithms have been put forward that have better performance in throughput and delay. DPoS is an extension of conventional PoS, which fixes the rich's problem of becoming richer in PoS through representative democracy. DPoS provides higher scalability, faster confirmation speed and lower consumption at the cost of limiting the number of delegates. According to the principle of Peercoin, we added a PoS consensus module to Ethereum, and comparative experiments of PoW (ethash of Ethereum), PoS and DPoS were carried out.

The fork means that different BPNs have temporarily accepted different blocks at the same height, and they will reach a consensus after a period of time. Although the fork has little effect on the throughput, it will affect the reliability of the transaction response. Figure 18 shows the average number of temporarily forked blocks for each BPN in an epoch (300 blocks). PoW has more forks than PoS, and DPoS has almost no forks. For each block in PoW and PoS, each BPN will compete for production through computing power or stake, which leads to different BPNs potentially temporarily accepting different valid blocks. Therefore, even if a transaction responds, it will likely take a certain time before it can be considered reliable. In DPoS, each BPN produces blocks in turn according to the agreed order. Regardless of dishonest BPNs and temporary network failures, DPoS can almost maintain zero forks, which means that transactions without irreversible status are considered reliable with 99.9% certainty [21]. The next section will further analyze security.

Under the same number of BPN/TN and transactions, the average CPU utilization and the average P2P network traffic of each bpi are shown in Figure 19. Compared with PoW, PoS greatly reduces computing power requirements, while DPoS is lower. We can also find that DPoS consumes fewer network resources because, in DPoS, there is only one BPN propagating the produced block in any bpi. On the other hand, as the number of nodes increases, the number of BPNs in DPoS is fixed, while in the other two, all qualified nodes can become BPNs, and the network overhead will be higher.

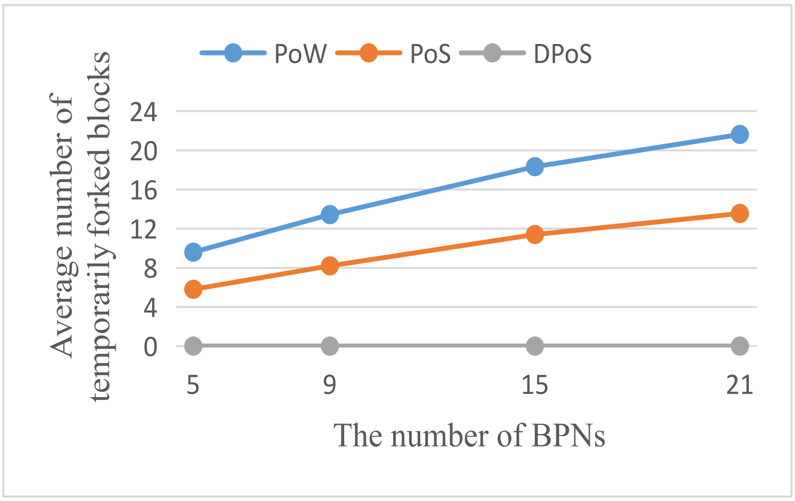

Figure 18. Average number of temporarily forked blocks in an epoch.

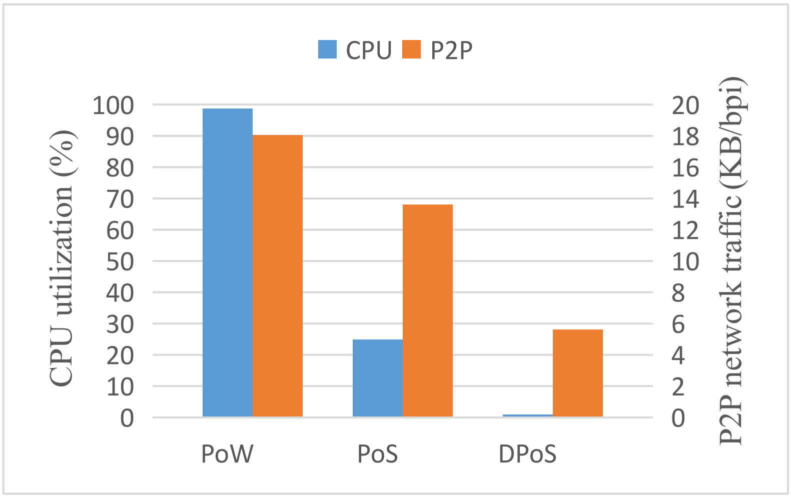

Figure 19. Average CPU utilization and P2P network traffic under different consensus algorithms.

4.5. Security Analysis

Identity or code theft. Some people attempt to use others' community account links to pass OSCMS authentication. Unless they hack a community account and modify the profile, identity theft cannot pass oracle verification. The submission of OI with others' code warehouse links will also fail because OSCMS verifies whether the community account to which a link belongs is bound to the submitter. Some people may copy others' codes to their code warehouse and submit them to the OSCMS for reputation or reward. These behaviors cannot be avoided at present, but the reputation will be offset by the objective ratings of others, and strict timestamps will prove who submitted the code to the warehouse first.

Destroy the meaning of reputation. If reputation in the OSCMS is vulnerable to overgrowth or defamation, it will lose the meaning of incentive and reference. Some possible destruction means have been analyzed in Section 3.1.2. In addition, whitewashing means registering a new OSCMS account to re-enter after having a bad rating that cannot be tampered with. As the initial reputation is zero, the user would not gain much from re-entering. However, a bad rating is worse than no rating, so there could be an incentive for doing so. Community account links that have been bound can no longer be used for authentication, even if they are replaced. Users can still reuse them by changing domain names, but it costs money. These restrictions urge users with a certain reputation not to submit bad OI, which cannot be whitewashed.

Bad behaviors of delegates within the rules. Delegates may exploit loopholes in the rules described in Section 3.5 for bad behavior. The first behavior is to intentionally ignore someone's transactions when producing blocks, such as reward solutions, but they will be processed by other honest delegates in a round and final results will not be affected. The second is to intentionally conduct different production frequencies in different blockchains, which will block the faster blockchain during epoch switching and affect system availability. Such behavior brings no benefits and greatly reduces this delegate's block rewards. Delegates who are too slow in a blockchain may be voted out by others. The third is double production, which refers to a delegate producing blocks in every temporarily forked chain. Honest delegates will eventually gather on the longest chain. Double production ensures that this delegate's block will be added to it, but this harms the system's ability to quickly recover stability. Honest delegates should monitor perpetrators according to cryptographic evidence and vote them out.

Double spending and 51% attack. Double spend occurs when an attacker benefits from a normal transaction that has been processed by the blockchain, creates a conflicting transaction and pushes it into a new forked fraudulent chain in an attempt to revert the previous transaction [57]. In DPoS, as delegates can only produce blocks within their own time windows, a minority of dishonest delegates cannot create a longer forked chain to replace the previous one. However, an attacker who has more than 50% of the power in the blockchain can control blockchain forks and which transactions are processed. In an OSCMS, this power comes from tokens for voting and accounts that can vote. There is a limit to the number of voting tokens for each account, and account authentication also has costs, so it is hard to achieve a 51% attack.

Sybil attack. In this attack, an attacker attempts to create/control as many fraudulent identities as required to exert influence on a P2P network [56]. In an OSCMS, the identity is bound to OS community accounts, which always require some type of trust, such as email/SMS. Meanwhile, a fraudulent identity without enough tokens cannot influence consensus and non-developers cannot be candidates.

4.6. Results and Discussion

Through a series of experiments on the system prototype, parameters that can be used for reference in future project deployment were obtained, including the time window size, the block size and the number of block production nodes (delegates). Under reasonable parameter settings, the system has a feasible submission response time, query delay and throughput. Under different numbers of nodes and delegate offline rates within the Byzantine fault tolerance range, the system still shows good performance, scalability and reliability. By comparison with different architecture schemes, the proposed scheme in this paper, including the triple-blockchain architecture and improved DPoS consensus, is superior. Furthermore, through performance and reliability analyses as well as testing of CPU utilization and P2P network traffic, it shows that DPoS is more suitable for the proposed system than any other mainstream blockchain consensus. Finally, some analysis is conducted on the possible negative behaviors that may occur in the system, indicating that the system has a certain level of security.

Based on blockchain-related technologies, the OSCMS not only explores a comprehensive range of solutions for the decentralized transformation of an OS ecosystem but also focuses on code control, incentive, interaction and cooperation, automation, transparency and fairness of rights and responsibilities. Compared to traditional OS governance methods mentioned in Section 2.2.1, the OSCMS has the following advantages:

- The OSCMS supports the rapid collection of global information, which is immutable and traceable. OSCMS can record the relationship between projects through smart contracts, and it can also provide easily accessible datasets for technologies such as machine learning.
- The token and decentralized reputation supported by blockchain are more conducive to incentives.
- The OSCMS has reshaped the trust model in the organization, trustlessness, enabling interaction and collaboration anywhere, anytime. A new identity management mechanism compatible with multiple communities has also been implemented in the OSCMS.
- Automation relies on smart contracts, which have higher transparency and credibility.
- The inherent decentralization of blockchain drives transparency and fairness in the governance process. Even delegates elected through the decentralized method are subject to comprehensive supervision. OS users can fully participate in every aspect.

Section 2.2.2 analyzes the application of blockchain in some related scenarios, which did not comprehensively discuss OS governance. However, in one or several aspects, they provide a reference for the study of an OSCMS. The OSCMS also improves their shortcomings:

- The requirements for the skills and costs of the participants have been reduced. Automation has been fully utilized, and some theft issues have been resolved through process coordination.
- The evaluation process has been redesigned to avoid losing credibility of the token or reputation.
- The identity authentication and permission management of the blockchain system have been studied, and solutions that are compatible with major communities have been proposed.
- These studies did not discuss the fairness and trustworthiness of consensus, and there was no in-depth discussion on the selected miners or supervision board. In the OSCMS, smart contracts cannot be freely published by someone or an organization. Delegates are transparently elected with limited power and subject to supervision by everyone.
- A novel triple-blockchain architecture is proposed, with higher efficiency and more coordinated relationships.

5. Conclusions and Further Work

In this paper, we designed an open-source coordination management system using a novel triple-blockchain architecture. We discussed the decentralized implementation of various open-source scenarios and gave consideration to security, sustainability, coordination, automation, transparency and fairness. Furthermore, a triple-blockchain DPoS consensus mechanism was added to drive the system. Based on the system prototype, the experimental results demonstrated that the OSCMS has good feasibility, scalability, reliability and performance. Finally, we also provided a security analysis and results discussion for reference. The OSCMS provides services that constitute a complete decentralized open-source ecosystem, and provides a novel ecosystem that users can fully participate in. The design of the architecture, modules and consensus algorithm proposed in this article, as well as the related experiments and discussions, have significant research value and practical implications for the decentralized transformation of open-source ecosystems.

In the future, we will focus on a more sophisticated reputation evaluation model that evaluates various open-source and consensus activities. Additionally, a more efficient consensus mechanism combining reputation will be further studied.

Author Contributions: Conceptualization, investigation, software, writing—original draft preparation, J.C.; methodology, J.C. and Y.Z.; validation, project administration, Y.Z.; supervision, Y.Z. and X.C.; writing—review and editing, X.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Natural Science Foundation of China under grant no. 62072342.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Butler, S.; Gamalielsson, J.; Lundell, B.; Brax, C.; Mattsson, A.; Gustavsson, T.; Feist, J.; Kvarnström, B.; Lönroth, E. Considerations and challenges for the adoption of open source components in software-intensive businesses. *J. Syst. Softw.* **2022**, *186*, 111152. [CrossRef]
2. Maldeniya, D.; Budak, C.; Robert, L.P., Jr.; Romero, D.M. Herding a Deluge of Good Samaritans: How GitHub Projects Respond to Increased Attention. In Proceedings of the Web Conference 2020, WWW'20, Taipei, Taiwan, 20–24 April 2020; pp. 2055–2065. [CrossRef]
3. Ohm, M.; Plate, H.; Sykosch, A.; Meier, M. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment, Lisbon, Portugal, 24–26 June 2020; Maurice, C., Bilge, L., Stringhini, G., Neves, N., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 23–43. [CrossRef]
4. Rousseau, G.; Di Cosmo, R.; Zacchiroli, S. Software provenance tracking at the scale of public source code. *Empir. Softw. Eng.* **2020**, *25*, 2930–2959. [CrossRef]
5. Kaur, R.; Kaur Chahal, K.; Saini, M. Understanding community participation and engagement in open source software Projects: A systematic mapping study. *J. King Saud Univ.-Comput. Inf. Sci.* **2020**, *34*, 4607–4625. [CrossRef]
6. Constantino, K.; Souza, M.; Zhou, S.; Figueiredo, E.; Kästner, C. Perceptions of open-source software developers on collaborations: An interview and survey study. *J. Softw. Evol. Process* **2021**, *35* , e2393. [CrossRef]
7. Charleux, A.; Viseur, R. Exploring impacts of managerial decisions and community composition on the open source projects' health. In Proceedings of the 2nd International Workshop on Software Health, Montreal, QC, Canada, 28 May 2019; pp. 1–8. [CrossRef]
8. Valentim, N.; Lopes, A.; César, E.; Conte, T.; Maldonado, J.C. An Acceptance Empirical Assessment of Open Source Test Tools. In Proceedings of the 19th International Conference on Enterprise Information Systems, Porto, Portugal, 26–29 April 2017; pp. 379–386. [CrossRef]
9. Lu, S.; Li, H.; Jiang, Z. Comparative Study of Open Source Software Reliability Assessment Tools. In Proceedings of the 2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS), Dalian, China, 20–22 March 2020; pp. 49–55. [CrossRef]
10. Tamburri, D.A.; Palomba, F.; Serebrenik, A.; Zaidman, A. Discovering community patterns in open-source: A systematic approach and its evaluation. *Empir. Softw. Eng.* **2019**, *24*, 1369–1417. [CrossRef]
11. German, D.M.; Robles, G.; Poo-Caamaño, G.; Yang, X.; Iida, H.; Inoue, K. "Was My Contribution Fairly Reviewed?" A Framework to Study the Perception of Fairness in Modern Code Reviews. In Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE) , Gothenburg, Sweden, 27 May–3 June 2018; pp. 523–534. [CrossRef]
12. Sharma, P.N.; Savarimuthu, B.T.R.; Stanger, N. Extracting Rationale for Open Source Software Development Decisions—A Study of Python Email Archives. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 25–28 May 2021; pp. 1008–1019. [CrossRef]
13. Sharma, P.; Savarimuthu, T.; Stanger, N. Influence of Roles in Decision-Making during OSS Development—A Study of Python. In *EASE 2021, Proceedings of the Evaluation and Assessment in Software Engineering, Trondheim, Norway, 21–23 June 2021*; Association for Computing Machinery: New York, NY, USA, 2021; pp. 50–59. [CrossRef]
14. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 29 March 2023).
15. Wood, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Available online: https://files.gitter.im/ethereum/yellowpaper/VIyt/Paper.pdf (accessed on 29 March 2023).
16. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15. [CrossRef]

17. Brown, R.G. Corda: An Introduction. Available online: https://corda.net/content/corda-platform-whitepaper.pdf (accessed on 29 March 2023).
18. Morgan, J. Quorum Whitepaper. Available online: https://github.com/ConsenSys/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf (accessed on 29 March 2023).
19. Verma, N.; Jain, S.; Doriya, R. Review on Consensus Protocols for Blockchain. In Proceedings of the 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), Greater Noida, India, 19–20 February 2021; pp. 281–286. [CrossRef]
20. King, S.; Nadal, S. Ppcoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. Available online: https://bitcoin.peryaudo.org/vendor/peercoin-paper.pdf (accessed on 29 March 2023).
21. Larimer, D. Dpos Consensus Algorithm-The Missing White Paper. Available online: https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper (accessed on 29 March 2023).
22. Arasev, V. POA Network Whitepaper. Available online: https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper (accessed on 29 March 2023).
23. Castro, M.; Liskov, B. Practical Byzantine Fault Tolerance. In *OSDI '99, Proceedings of the Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, LA, USA, 22–25 February 1999*; USENIX Association: 1999; pp. 173–186. [CrossRef]
24. Szabo, N. Formalizing and Securing Relationships on Public Networks. *First Monday* **1997**, *2*. [CrossRef]
25. Caldarelli, G. Understanding the blockchain oracle problem: A call for action. *Information* **2020**, *11*, 509. [CrossRef]
26. Breidenbach, L.; Cachin, C.; Chan, B.; Coventry, A.; Ellis, S.; Juels, A.; Koushanfar, F.; Miller, A.; Magauran, B.; Moroz, D.; et al. Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks. 2021. Available online: https://naorib.ir/white-paper/chinlink-whitepaper.pdf (accessed on 29 March 2023).
27. Woo, S.; Park, S.; Kim, S.; Lee, H.; Oh, H. CENTRIS: A Precise and Scalable Approach for Identifying Modified Open-Source Software Reuse. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 25–28 May 2021; pp. 860–872. [CrossRef]
28. Rath, M.; Tomova, M.T.; Mäder, P. SpojitR: Intelligently Link Development Artifacts. In Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada, 18–21 February 2020; pp. 652–656. [CrossRef]
29. Smirnova, I.; Reitzig, M.; Alexy, O. What makes the right OSS contributor tick? Treatments to motivate high-skilled developers. *Res. Policy* **2022**, *51*, 104368. [CrossRef]
30. Barcomb, A.; Kaufmann, A.; Riehle, D.; Stol, K.J.; Fitzgerald, B. Uncovering the Periphery: A Qualitative Survey of Episodic Volunteering in Free/Libre and Open Source Software Communities. *IEEE Trans. Softw. Eng.* **2020**, *46*, 962–980. [CrossRef]
31. Gerosa, M.; Wiese, I.; Trinkenreich, B.; Link, G.; Robles, G.; Treude, C.; Steinmacher, I.; Sarma, A. The Shifting Sands of Motivation: Revisiting What Drives Contributors in Open Source. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 25–28 May 2021; pp. 1046–1058. [CrossRef]
32. Liao, Z.; He, D.; Chen, Z.; Fan, X.; Zhang, Y.; Liu, S. Exploring the Characteristics of Issue-Related Behaviors in GitHub Using Visualization Techniques. *IEEE Access* **2018**, *6*, 24003–24015. [CrossRef]
33. Bock, T.; Hunsen, C.; Joblin, M.; Apel, S. Synchronous development in open-source projects: A higher-level perspective. *Autom. Softw. Eng.* **2022**, *29*, 1–53. [CrossRef]
34. Zhang, P.; Liu, P.; Wang, N. Evolutionary analysis of developer collaboration network in Cloud Foundry OSS community. In Proceedings of the Knowledge and Systems Sciences: 20th International Symposium, KSS 2019, Da Nang, Vietnam, 29 November–1 December 2019; Proceedings 20; Springer: Berlin/Heidelberg, Germany, 2019; pp. 87–105. [CrossRef]
35. Mishra, A.; Otaiwi, Z. DevOps and software quality: A systematic mapping. *Comput. Sci. Rev.* **2020**, *38*, 100308. [CrossRef]
36. McAffer, J. Getting Started With Open Source Governance. *Computer* **2019**, *52*, 92–96. [CrossRef]
37. Eckert, R.; Stuermer, M.; Myrach, T. Alone or Together? Inter-organizational affiliations of open source communities. *J. Syst. Softw.* **2019**, *149*, 250–262. [CrossRef]
38. Drost-Fromm, I.; Tompkins, R. Open Source Community Governance the Apache Way. *Computer* **2021**, *54*, 70–75. [CrossRef]
39. Salman, T.; Zolanvari, M.; Erbad, A.; Jain, R.; Samaka, M. Security Services Using Blockchains: A State of the Art Survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 858–880. [CrossRef]
40. Bose, R.J.C.; Phokela, K.K.; Kaulgud, V.; Podder, S. BLINKER: A Blockchain-Enabled Framework for Software Provenance. In Proceedings of the 2019 26th Asia-Pacific Software Engineering Conference (APSEC), Putrajaya, Malaysia, 2–5 December 2019; pp. 1–8. [CrossRef]
41. Hu, Q.; Asghar, M.R.; Zeadally, S. Blockchain-based public ecosystem for auditing security of software applications. *Computing* **2021**, *103*, 2643–2665. [CrossRef]
42. Zheng, Y.; Boh, W.F. Value drivers of blockchain technology: A case study of blockchain-enabled online community. *Telemat. Inform.* **2021**, *58*, 101563. [CrossRef]
43. Zeng, Q.; Zhang, X.; Wang, T.; Shi, P.; Fu, X.; Feng, C. BBCPS: A Blockchain Based Open Source Contribution Protection System. In *Blockchain and Trustworthy Systems* ; Zheng, Z., Dai, H.N., Tang, M., Chen, X., Eds.; Springer: Singapore, 2020; pp. 662–675. [CrossRef]

44. Alimoğlu, A.; Özturan, C. Design of a Smart Contract Based Autonomous Organization for Sustainable Software. In Proceedings of the 2017 IEEE 13th International Conference on e-Science (e-Science), Rome, Italy, 9–11 October 2017; pp. 471–476. [CrossRef]
45. Canidio, A.; Costa, G.; Galletta, L. VeriOSS: Using the blockchain to foster bug bounty programs. *Open Access Ser. Inform.* **2021**, *82*, 1–14. [CrossRef]
46. Qayum, A.; Razzaq, A. A Self-Evolving Design of Blockchain-based Open Source Community. In Proceedings of the 2020 3rd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 29–30 January 2020; pp. 1–11. [CrossRef]
47. Singi, K.; S, P.D.; Kaulgud, V.; Podder, S. Compliance Adherence in Distributed Software Delivery: A Blockchain Approach. In *ICGSE '18, Proceedings of the 13th International Conference on Global Software Engineering, Gothenburg, Sweden, 27 May–3 June 2018*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 131–132. [CrossRef]
48. Król, M.; Reñé, S.; Ascigil, O.; Psaras, I. ChainSoft: Collaborative Software Development Using Smart Contracts. In *CryBlock'18, Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems, Munich, Germany, 15 June 2018*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1–6. [CrossRef]
49. R P, J.C.B.; Singi, K.; Kaulgud, V.; Phokela, K.K.; Podder, S. Framework for Trustworthy Software Development. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), San Diego, CA, USA, 11–15 November 2019; pp. 45–48. [CrossRef]
50. Li, C.; Qu, X.; Guo, Y. TFCrowd: A blockchain-based crowdsourcing framework with enhanced trustworthiness and fairness. *EURASIP J. Wirel. Commun. Netw.* **2021**, *2021*, 168. [CrossRef]
51. Ren, W.; Wan, X.; Gan, P. A double-blockchain solution for agricultural sampled data security in Internet of Things network. *Future Gener. Comput. Syst.* **2021**, *117*, 453–461. [CrossRef]
52. Wang, W.; Wang, L.; Zhang, P.; Xu, S.; Fu, K.; Song, L.; Hu, S. A privacy protection scheme for telemedicine diagnosis based on double blockchain. *J. Inf. Secur. Appl.* **2021**, *61*, 102845. [CrossRef]
53. Fu, Y.; Zhu, J. Trusted data infrastructure for smart cities: A blockchain perspective. *Build. Res. Inf.* **2021**, *49*, 21–37. [CrossRef]
54. Vassallo, C. Enabling Continuous Improvement of a Continuous Integration Process. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 11–15 November 2019; pp. 1246–1249. [CrossRef]
55. Koblitz, N. Elliptic curve cryptosystems. *Math. Comput.* **1987**, *48*, 203–209. [CrossRef]
56. Ferdous, M.S.; Chowdhury, M.J.M.; Hoque, M.A. A survey of consensus algorithms in public blockchain systems for crypto-currencies. *J. Netw. Comput. Appl.* **2021**, *182*, 103035. [CrossRef]
57. Bamakan, S.M.H.; Motavali, A.; Bondarti, A.B. A survey of blockchain consensus algorithms performance evaluation criteria. *Expert Syst. Appl.* **2020**, *154*, 113385. [CrossRef]