A Design of Electronic Medical Record System based on Permissioned Blockchain

by

Aiden Feng

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical & Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Blockchain, due to its decentralized, verifiable, and security features, is increasingly used in more and more scenarios, but the original public chain cannot meet the needs of some enterprise or group financial scenarios or medical scenarios, and these more segmented special application scenarios have added many requirements for security, identity and role definition to the blockchain system, so people have launched a lot of research on permissioned blockchain. However, the Practical Byzantine Fault-Tolerant (PBFT) based permissioned chain is no longer able to meet the increasing demand of nodes in today's healthcare system due to the shortcomings such as high communication volume and poor scalability. In this thesis, we will explore the application of permissioned blockchain in medical scenario, and try to optimize the communication volume of PBFT by cascading network, optimize its communication complexity from $O(n^2)$ to $O(n)$, and add credit mechanism to reduce the probability of key nodes becoming Byzantine nodes. We also add a series of scalability mechanisms and new node verification addition and removal mechanisms to compensate for the disadvantage of non-dynamic number of nodes in the original PBFT.

Acknowledgements

I would like to thank Prof. Gong for her supervision of my thesis. Over the past two years, I have been constantly inspired by her to study and research, and she has always been able to patiently lead me to overcome them whenever I encountered difficulties. What I have felt from her is not only the guidance and expertise as a professor, but also the courageous attitude to explore the unknown.

Dedication

This is dedicated to my parents and my friends for their endless love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the development of modern science and communication technology, people have gradually and progressively replaced the original paper-based text file system with a digital data system [71]. As the coronavirus COVID-19 has affected people's lives for nearly two years, the need for digital health systems and smart healthcare has become more and more urgent. The healthcare industry and medical research are increasingly demanding interoperability, compliance, verifiability, privacy and traceability of patient records and medical information [25].

In addition, in 2021, there were more than 1,500 drug quality and counterfeit incidents in North America alone [52]. In fact, since a long time ago we have experienced the iterative change of data-based healthcare systems, and healthcare data is now mostly converted from paper to digital healthcare records (EMR) [37], and healthcare data is slowly becoming compliant, systematic, secure and efficient with electronic integration.

Besides, many digital-based health applications have emerged, such as the Apple Health personal medical information monitoring system, or the Keep's personal health management software that targets users who are keen to share their exercise status, as if everyone's medical and health information can be easily shared and analyzed [38, 69]. But also because of this, sensitive personal health information is constantly at risk of information leakage, and the difficulty of government regulation has greatly increased due to the proliferation of health management software [67].

In the case of hospitals, most of the current medical information systems are centralized, and each medical institution has its own relatively mature set of electronic medical record system [62]. Although the learning time cost of internal training is avoided to some extent, the information between the various systems is not very efficient, interoperable and

compatible, which creates a potential problem of isolated data island. For the management of sensitive medical data, the ownership of the data should belong to the individual, but because of the wide range of medical and health applications, users do not know which software or organizations their medical data is shared with, nor can they control the access rights of third parties to the data [74]. Moreover, with the increasing convenience of transportation, patients are likely to visit different medical institutions in different systems far away from each other, and medical records may be fragmented and incomplete [15].

For these reasons, we need a new medical information system to make up for these shortcomings. In this thesis, we will talk about how to use blockchain to store, share and verify medical information, and we will focus on the core of blockchain, its consensus algorithm. We will optimize the original consensus algorithm in terms of expanding functionalities and performance for the large number of nodes and dynamic nature of the medical scenario.

Chapter 2

Background and Related Works

We introduce some basics about blockchain in Section 2.1, including some introduction to its basic structure, basic kinds and application scenarios. Then we provide a detailed introduction to distributed systems in Section 2.2, introducing the concepts of Consistency problem and consensus protocol, and introducing and comparing some common consensus algorithms and their application scopes. Finally, we introduce some previous medical blockchain projects and BFT-based consensus models in Section 2.3, respectively, and further discuss and compare their design differences and advantages and disadvantages.

2.1 Blockchain

In a simple sense, blockchain can be described as a database shared among multiple distributed nodes, and the data stored on it is highly unforgeable, verifiable, access traceable, and transparent, without the need for a public trusted authority [56]. The implementation of blockchain technology is based on peer-to-peer transmission, consensus mechanisms, and cryptographic algorithms, and is widely used in various scenarios to solve the collaboration problems among multi-parties through its shared ledger due to its transparency characteristics [1]. The basic components of blockchain can be broken down into three parts: distributed ledger, consensus algorithm, and smart contracts [34].

The earliest conception of blockchain applications can be traced back to a well-known paper "Bitcoin: A Peer-to-Peer Electronic Cash System" [54], in which blockchain is a distributed ledger-style data structure for digital currency scenarios. It describes blockchain as a distributed bookkeeping technology that uses a chained data structure to provide a

tamper-proof record of data using digital digests. But in fact, the use of chained structures and the comparison of hash values and timestamps to ensure data security has been proposed as early as 1990 in an article called "How to Time-Stamp a Digital Document" [31] by Stuart Haber and W. Scott Stornetta.

2.1.1 Blockchain Structure

First of all, there are three basic components in the blockchain structure:

- Transaction: refers to all operations on the ledger.
- Block: It is used to record all the transactions and states within a specified time interval, and can be considered as a consensus result for the current state of the ledger.
- Chain: A set of blocks which are listed in chronological order, and can be considered as a historical log of all the time nodes of the whole ledger.

The simple structure of a blockchain is shown in the Figure 2.1 below, where the later blocks are linked into a chain by recording the hash value of the previous blocks. The blockchain is designed to pack and add a new block every once in a while, and this block is generated through the consensus protocol that we will mention later.

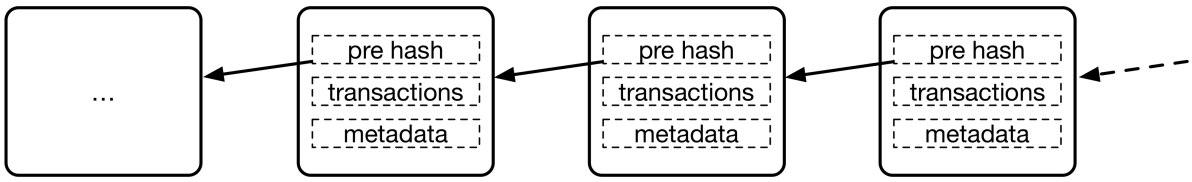

Figure 2.1: Example of the Basic Blockchain Structure

In the article "Bitcoin: A Peer-to-Peer Electronic Cash System" [54], the user first initiates a transaction request through the client, which is propagated to the network and awaits confirmation. The node in the network will pack multiple transaction requests and add hash values and other information to form a block structure. Then the node will try to find a nonce random string through computation so that its hash result meets a specific requirement, this whole process is often referred to as "mining". When a node finds a

Table 2.1: Comparison of Different Types of Blockchain

Type	Ledger-keeping rights	Degree of decentralization	Performance
Public	all	high	low
Consortium	partial	medium	medium
Private	private	low	high

nonce that satisfies the requirement, it submits a block that is broadcast to the network, and when other nodes receive this block, they verify it and add it to the local ledger.

It is important to note that once a block is submitted to the local chain, it cannot be deleted, which means that the length of the chain only increases and does not decrease, which is why the blockchain can provide the feature of no tampering, but it also limits the scalability of the whole system. This consensus mechanism is called Proof of Work (PoW), which can prevent malicious nodes from affecting the system to a certain extent, but it also wastes a lot of energy. We will include this concept in the later introduction of various consensus mechanisms.

2.1.2 Types of Blockchain

As shown in Table 2.1, the types of blockchain are divided into Public chain, Private chain and Consortium chain [2]. The core difference between them is the read and write access and the degree of decentralization. Public chains are open to everyone, private chains are open to individuals who meet certain conditions, and consortium chains are open to authorized organizations or institutions. In essence, consortium chains also belong to private chains, but the degree of privatization is different. Generally speaking, the higher the degree of decentralization, the higher the trustworthiness, and the lower the transaction efficiency.

Public Blockchain

The Public blockchain refers to a blockchain in which anyone in the world can access the system at any time to read data, send confirmable transactions, and compete for billing. A public chain is often considered "completely decentralized" because no single person or organization can control or tamper with the reading or writing of the data. The public chain generally encourages participants to compete for billing through a token mechanism to ensure data security. Bitcoin and Ethereum are typical public chains.

Private Blockchain

Private blockchain refers to a Blockchain in which write access is controlled by an organization or institution, and the eligibility of participating nodes is severely limited. Since the participating nodes are limited and controllable, the private chain can often have extremely fast transaction speed, better privacy protection, lower transaction costs, less vulnerable to malicious attacks, and can meet the requirements necessary in the financial industry such as identity authentication. In contrast to a centralized database, a private chain prevents a single node in an organization from deliberately concealing or tampering with data, and allows the source to be discovered quickly even if errors occur. Therefore, many large financial institutions are more inclined to use private chain technology at present.

Permissioned Blockchain

Permissioned Blockchain refers to the fact that every node participating in a Blockchain system is licensed. Unlicensed nodes cannot be connected to the system, so both private and consortium chains are licensed. Some license chains have no token mechanism because there is no need for tokens to encourage nodes to compete for billing.

Consortium Blockchain

Consortium Blockchain refers to a group of organizations that operate on one or more nodes, each of which contains data that is only allowed to be read, written, sent and recorded by different organizations within the system. A consortium chain is actually a concept between a public and private chain, which to some extent allows various organizational structures to cooperate and maintain the blockchain together. Consortium chains are usually permissioned blockchain, for example, hyperfabric introduced additional permission management mechanism, which is a typical example of consortium blockchain structure.

In blockchain medical applications, it is clear that hospitals and medical institutions can be entities in the system, while these institutions have to collaborate with insurance companies and scientific institutions, in addition it may be regulated by government departments, and this system needs very strict access control and hierarchical system. In addition, medical data, as sensitive data, must be strictly controlled in terms of access permissions. Therefore, we believe that the blockchain application in the medical scenario is well suited to adopt the above mentioned Consortium chain design [39].

2.1.3 Healthcare with Hyperledger Fabric Blockchain

In keeping with the times, blockchain could help healthcare organizations ease the financial pressures they face during the coronavirus COVID-19. J.P. Morgan cites a survey by the Healthcare Advisory Council that there is about 40 million to 44 million dollars in annual O&M costs for nonprofit healthcare systems, which, if saved, could significantly reduce the financial strain on healthcare organizations during an pandemic and provide long-term stable revenue [53]. Also, JPMorgan estimates that blockchain could save as much as 80% of the cost and time currently spent in authenticating medical information [53]. Often the authentication of medical information is complex and error-free, and some, such as the authentication of physician credentials, can be time-consuming because it involves collaboration between multiple organizations or systems. However, if we can use blockchain technology, hospitals can verify the authenticity of a proof in a very short period of time.

Hyperledger Fabric [5] is an open source permissioned blockchain framework. Hyperledger Fabric implements a PBFT consensus, which has decent performance compared to ethereum's blockchain, but since it is not a public blockchain, all participants must be authenticated [55]. The Hyperledger fabric blockchain network consists of different channels, and specific members can build subnets to ensure the security and privacy of their information communication. For example, if we set up a medical platform on the hyperledger fabric, it would enable collaboration between hospitals, patients, research institutions, insurance companies, and the healthcare logistics chain, e.g., patients could provide data to research institutions by filling out surveys or actively sharing medical information. Insurance companies can also submit requests to hospitals to verify the authenticity of insurance declarations. In the recent example of the coronavirus COVID-19, if we find a problem with a batch of vaccine, we can use the blockchain to trace the logistics information to determine which batch of goods is faulty and notify the patients who received the corresponding batch of vaccine. Hyperledger fabric provides an additional access control layer, where every transaction on the network is executed on a channel, and each party must be authenticated and authorized to conduct transactions on this channel. For example, if medical institutions and insurance agencies want to access patient information, they will notify the patient and apply for authorization. This ensures that patients have a complete control over access to their information, thus changing the hospital-centric model to a patient-centric healthcare delivery system is approachable in this case.

2.2 Core Technologies in Distributed Systems

In the book "Distributed Systems Concepts and Design" [17], a distributed system is defined as a system in which hardware or software components are distributed across different network nodes, they communicate and coordinate with each other only through message passing. With the increasing complexity of network systems, most system designs have shifted from monolithic to distributed architectures, and non-traditional distributed systems led by blockchain have gained even more popularity in the past few years. Technologies such as blockchain, which are based on distributed technologies, are highly dependent on data consistency and consensus mechanisms, and in this section we will introduce each of these concepts.

2.2.1 Consistency Problem

The consistency problem is the top priority of distributed systems and the first problem to be solved in blockchain. In the blockchain world, with an increasingly large number of nodes, the complexity of the system is increasing with the volume of the nodes, but in order to ensure the consistency of each node's pace, we have to meet the requirements of the whole system in terms of scalability and fault-tolerance [63].

Definition of Consistency

For multiple nodes in a distributed system, given a set of instructions, we need to make the entire system reach a certain degree of agreement on the outcome of the instruction processing through fulfilling the consensus protocol [70].

It may seem that all that is needed to get consistent results from distributed nodes is to ensure the serialized execution of instructions, but in reality it is difficult, for example:

- Network communication may be blocked and there may be time delays, which can cause disorder, delay or even loss of messages.
- The computing power and processing latency of each node and the results are not guaranteed, thus there may be a failed node at any time
- As the number of nodes increases, the complexity of communication between nodes may increase exponentially, making the system suffer from the trade-off between throughput and scalability.

Consistency Requirements

In Leslie Lamport's 1978 paper "Time, Clocks and the Ordering of Events in a Distributed System" [43], it is pointed out that for an asynchronous system to determine a consistent result, it is required that systems with different clocks to execute commands in the same order. However, since each node in an asynchronous system has its own clock and there are time delays in communication, this event order has to be reasonably ordered in order to achieve a correct common outcome.

According to [3], a distributed system must achieve the following 3 requirements.

- Termination: The system must be able to get the result in a limited time frame.
- Agreement: Each node must agree on the final outcome of the instruction.
- Validity: The decision must be generated in a compliant manner, even though there may be processing and communication delays among the nodes in the distributed system, if all nodes have the same initial value, then they must get the same result after executing the instruction.

In fact, there is a trade-off between the overall performance and consistency and scalability of an asynchronous system. So depending on the performance requirements of the system, one classifies the consistency requirements into different categories [9, 23].

Sequential Consistency and Linearizability Consistency [7] are the more common Strong Consistency systems. The former is mentioned in Leslie Lamport's [41] in 1979, where the problem of consistency between local order and global order is addressed. That is, if there is a sequential order of execution between two processes in the vision of a process, then this order needs to be guaranteed in the global order as well. The latter concept of Linearizability Consistency [32] is always mentioned in operating system-related concepts, and it allows a certain degree of flexibility in the global ordering between processes, but it is very difficult to implement because it relies heavily on global clocks or mutual exclusion locks.

In reality, systems with strong consistency are often not as easy to implement. Most systems that do not require a high level of consistency but have a very high volume of transactions, such as in some designs of online shopping systems, tend to require only eventual consistency [10], which means that the user is always allowed to add items to the shopping cart, but will be eventually prompted with "inventory not available" when he is ready to pay. If we take the blockchain as an example, we can easily find that most of the

time the ledger of each node may be inconsistent at different time periods, but every once in a while the consensus protocol will be used to achieve the ultimate consistency of the whole chain.

2.2.2 Consensus Protocol

Consensus algorithm in a distributed system refers to the process adopted by individual nodes to pursue certain levels of consistent results. In other words, consensus algorithm is: a process by which a distributed system exchanges information among nodes to make most of them agree on a proposal made by a leading node [64, 73, 75, 13].

For distributed individual nodes can be thought of as State-Machine Replication, where each node starts from the same initial state, and by operating the same instructions, they are all guaranteed to reach the same result state according to the previously mentioned validity principle of distributed systems. Therefore, the most critical thing for multiple nodes in the system is to agree on the order of multiple events.

As mentioned before, due to the presence of network state and physical delays, and the risk of having adversary in the system, we have to consider fault-tolerance in the consensus algorithm. In general, we will classify those faulty unresponsive nodes as Non-Byzantine Fault; and those nodes that deliberately spread false consensus information in order to interfere with consensus reaching In general, we will classify those faulty non-responsive nodes as Non-Byzantine Fault; and those nodes that deliberately spread false consensus information in order to interfere with the consensus, we will call them Byzantine Node.

2.2.3 FLP Impossibility Principle

FLP(Fischer, Lynch, and Paterson) impossibility principle is considered to be one of the most important principles in distributed systems. Fischer, Lynch and Patterson concluded in [24] that in asynchronous model systems with reliable networks but with the possibility of node failure, there is no deterministic consensus algorithm to solve the consistency problem.

2.2.4 CAP

Distributed computing systems cannot guarantee the following three characteristics at the same time: strong consistency, high availability, and partitioning. Therefore, in the

design of distributed systems, it is often necessary to weaken the guarantee of one of the characteristics [28].

- Strong consistency: all nodes in the distributed system can see the same data content at any time.
- High availability: Any node in the distributed system that is not non-faulty can respond to the request.
- Partitioned tolerance: The network may be partitioned, meaning that communication between partitioned nodes is not guaranteed.

2.2.5 Consensus Algorithms

It is often said that consensus algorithm is the core of blockchain, the reason is that the whole blockchain depends on it to ensure the consistency of distributed nodes and the trustworthiness of data on the blockchain.

Perhaps in a cloud-based information system, due to the existence of centralized nodes, we only need to upload and synchronize data and download updates for the central node. In a blockchain system, each node needs to ensure that its ledger is synchronized with the ledger information of other nodes, so specifying a process rule for each node to reach this consensus result becomes the top priority of the blockchain algorithm, and consensus algorithm refers to such an algorithm that enables distributed nodes to negotiate a consensus decision together beforehand. We will then briefly introduce the various consensus algorithms and their features.

Proof of Work (PoW)

PoW is the most common and original consensus algorithm used in blockchain applications. Its concept was first proposed by Cynthia Dwork and Moni Naor in 1993 [20], and was made more complete and coined in a 1999 article by Markus Jakobsson and Ari Juels [36]. The main applications that use the PoW consensus algorithm are Bitcoin [54], which we know best, Ethereum [11], and Litecoin [26], which is popular in the cryptocurrency trading market, as well as Dogecoin [78], which is getting a lot of attention these days because of Elon Musk's recommendation. In PoW, a blockchain participant (also known as a Miner) must solve a complex computational problem if it wants to add a piece of transaction. To keep the block generation time constant, the complexity of this computation changes

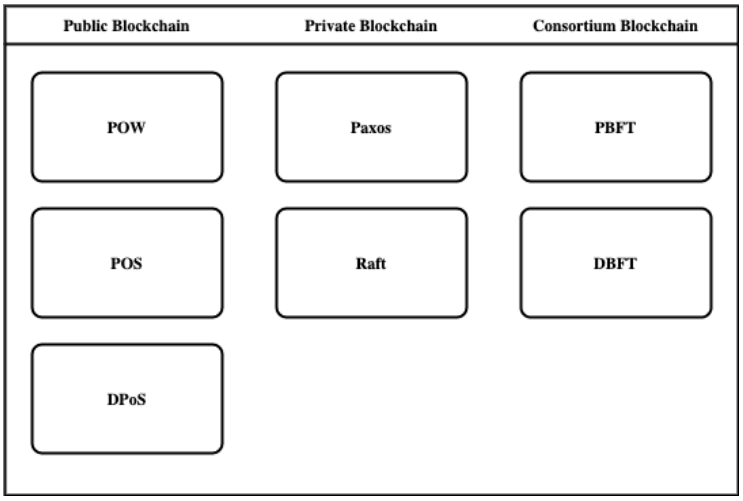

Figure 2.2: Summary of Consensus Algorithms and its Applied Scenarios

accordingly, e.g., in Bitcoin, the generation time of new blocks is set to every 10 minutes. When there are multiple miners working on a chain at the same time, then the one with the fastest growth and the longest chain length will be recognized as the winner. Under this premise, the chain is secure as long as more than half of the transactions submitted by miners are trusted, this has the advantage of avoiding the 51% attack, meaning that if the adversary want to take management control of the blockchain by generating the faulty blocks and making the majority (51%) accept it, such an attack would turn out to be unprofitable and costly. Although PoW-based blockchain systems have countless applications in the industry, this approach requires too much computing power thus wastes a lot of electricity resources, so many scholars are aiming to replace it.

Proof of Stake (PoS)

PoS is a good remedy for the shortcomings of the PoW algorithm and has been applied in the latest generation of Ethereum [12]. Compared to POW, which relies mainly on the costly power of arithmetic to prevent witch attacks, PoS relies on the "monetary policy" in the blockchain. PoS allows each Miner to mine or verify new blocks depending on the amount of cryptocurrency they have (Stake) and not on their computing power as in PoW. This means that in PoS, the more Stakes you have, the more chances you have to mine the next block, and the system also encourages verifiers to participate in the network through incentives and thus promotes consensus. This approach avoids wasting computing

power and electricity, but the node with more Stakes in this setup may have a long-term advantage.

Delegated Proof-of-Stake (DPoS)

DPoS, also known as "Proof of Share Authority Mechanism", was proposed by Dan Larime of Bitshares in 2014 [46]. It is a highly scalable blockchain consensus protocol based on a further optimization of the PoS method. It requires all nodes to vote every once in a while and elect a specified number of super nodes as block producers, where the weight of the vote is based on the number of stakes held, and the more stakes you have the more important the vote is. These super nodes have exactly equal weight and they vote and create blocks in a round robin order. When a block is voted on by more than 2/3 of the block producers, the block will be determined. Otherwise, the longest chain rule is followed. If during this process the verifier finds the presence of any malicious node, it is excluded in the next round of super node voting. Since only a small number of super nodes are voted in as block producers to participate in consensus, the throughput of DPoS is considerably higher than that of PoW. However, because of this, DPoS is also considered a solution that sacrifices decentralization for throughput, typical application are BitShares [65] and Ark [6].

Table 2.2: Consensus Algorithm Comparison

	PBFT	PoW	PoS	DPoS
Nodes Managements	Permissoned	Non-permissioned	Non-permissioned	Permissoned
Energy Consumption	Very Low	High	Low	Very Low
Bookkeeping Nodes	Dynamic	All	Highest stake	Highest stake
Respond Time	In seconds	10 mins	1 min	In seconds
Transaction/Second	∼1500	∼30	∼170	∼2500
Example	Hyperledger	Bitcoin	Ethereum	Bitshares

Paxos

The algorithm of Paxos was first proposed by Leslie Lamport in 1990 in [42] and further refined in [44], in which his design also takes into account fault tolerance, but only the impact of non-byzantine nodes, that is, there is no intentional evil nodes in the default system, so it is often used in distributed systems like zookeeper or private blockchain application scenarios. This algorithm uses a two-staged submission method (prepare stage

and commit stage), where a proposer is first selected from multiple proposal candidates, and then this proposal message is sent to other nodes, and the proposal is passed when the majority of nodes agree to it. The concept of majority of nodes is defined as "quorum" by Leslie Lamport in the paper. The fault tolerance of the Paxos algorithm is 1/2, which means that the system can reach a correct consensus when more than half of the nodes are working properly.

Raft

Raft's algorithm evolved from Paxos, which was proposed by Diego Ongaro and John Ousterhout in 2014 in the paper "In Search of an Understandable Consensus Algorithm" [60]. Raft is more structured and easier to understand than Paxos, and provides better security. It ensures that any node in the cluster is consistent in some kind of state transition.

Practical Byzantine Fault Tolerance (PBFT)

Castro and Liskov first proposed PBFT in 1999 in "Practical Byzantine Fault Tolerance and Proactive Recovery" [14]. It is an excellent solution to the problem of the Byzantine faulty nodes in distributed systems, which leads to unsynchronized and inconsistent information. It is essentially based on the optimization of the Paxos algorithm, which for the first time reduces the complexity of the Byzantine fault-tolerant algorithm from the exponential level to the polynomial level. Additionally, when the total number of nodes in the system is n and the number of faulty nodes is f, PBFT can guarantee the normal information communication of the whole system under the condition of $n \geq 3f + 1$. More detailed illustration of PBFT will be discussed in Chapter 4.

2.3 Review of Related Works

In addition to some basic introductions to blockchain and distributed system concepts mentioned above, we will further explore in this section the evolutionary history of some previous blockchain healthcare systems, and introduce and compare a series of BFT algorithms for Byzantine fault-tolerant designs which might be suitable to be implemented in the healthcare scenarios.

2.3.1 Medical Blockchain Projects

The initial idea was to use cloud services to optimize the storage and sharing of electronic medical records, which is now more widely accepted. However, this approach has many disadvantages. For example, in terms of cost, it needs to consider the cost when setting up the infrastructure, annual maintenance Cost, migration cost, and Productivity Loss. Moreover, such centralized storage mode has great risks, such as large-scale power outage, natural disasters, artificial server attacks, and necessary data center maintenance downtime, which will greatly affect the stability and efficiency of the system, and such Availability problems are unacceptable in the medical field where every second is needed. In terms of security, there are many points of vulnerability in the solution of cloud service. Although the research on this aspect has been very mature, we may need to consider the security architecture in the quantum computing era in advance. In addition, in order to meet the auditability and traceability of the healthcare system, the cloud service architecture may sacrifice part of the operational efficiency. But even so, the cloud service can only meet the auditability and capability of the healthcare system on a regular basis rather than in real-time. Therefore, most current solutions still rely on in-house information system to achieve auditability. Based on these reasons, people began to replace the hospital-centered medical system, and to seek a patient-centered solution [68].

As early as 2015, the researchers from MIT proposed the idea of using blockchain to achieve privacy-sensitive data access control in the era of big data [79]. They believe that data should not be centrally stored on any third-party platform, because such a single point of failure gives opportunity to the potential adversaries, and such a storage method is not conducive to the transmission or control of large amounts of data, nor can it prove the accuracy of data. Therefore, they proposed a model of storing user data with blockchain, which is not only beneficial to users' control of data sharing and access rights, but also can provide verifiability for data to be accessed, modified or stored, because it is computationally robust to tampering.

The earliest comprehensive attempt at blockchain-based healthcare came in 2016, when MIT's Media Lab published a project at the Beth Israel Deaconess Medical Centre on using blockchain to manage medical data and provide access control and notification for the patients [8]. Based on the inherent privacy, interactivity, verifiability and accountability of blockchain, they also provided a functional prototype trying to utilize the decentralization fact of Blockchain and allow patients to be freely manage the read and modification permissions of their sensitive electronic medical records. At the same time, it also points out a future development trend, using the Miner and Rewards mechanism of blockchain to build a healthy ecological system of voluntary medical data sharing and medical data

analysis among medical stakeholders and researchers, which can promote the development of medical technology in the long term.

However, the original version of MedRec still has many problems. For example, it is vulnerable to the Eclipse Attack, which is an attack method that can gradually extend from an adversary on the chain by using the interaction manipulation and finally obtain the information of the whole chain. Moreover, its scalability has also been criticized. In the subsequent version 2.0, they updated a new method of using Anonymizing Metadata to store information on the chain, which further reduced the storage burden of information on the chain and enhancing its extensibility, as well as making users' accounts more concealed [51].

Also in 2018, another promising medical blockchain system was launched, and it is called MedicalChain [50]. It is also a decentralized healthcare platform that aims to provide a transparent, secure and efficient healthcare information architecture. It has a unique dual chain architecture and utilizes Hyperledger Fabric for EMR access control. As an extension of the functionality of the platform, it implements a transaction system centered on MedToken, where patients can voluntarily share their health data or actively participate in the trial of new drugs in exchange for a specified number of MedTokens. At the same time, patients can also consume MedToken to obtain some price services such as online diagnosis or medicine purchase. In this virtuous circle, there will be greater transparency and greater participation in medical research and development.

In addition to the above projects focusing on medical data control and medical information sharing, with the popularity of the Internet of Things research in recent years, people are also gradually focusing on the further medical supply chain. Medical products are known to have stricter import and export regulations, shipping environmental standards, and the need for timeliness and traceability in emergency situations. Given this background, IOTEX [35] has launched its blockchain platform that associated secure hardware, decentralized identity, and real-world data oracles, its scalability can be used in a variety of applications including medical scenarios, such as tracking the body data of patients with chronic diseases, online identity verification based on sensors such as fingerprints, temperature and environmental monitoring of medical transportation systems, and so on [21]. While its performance in terms of latency or sensor data stability remains to be seen, its emergence certainly points to a bright direction for healthcare blockchain.

Based on the in-depth exploration of previous blockchain medical projects, it is easy to find that they are still mainly at the stage of verifying feasibility, which is probably due to the limitation of scalability to land the project into practical application. Due to the bottleneck in throughput of the underlying ethereum and hyperfabric networks, MedRec's

measured efficiency is around 15-30tps, which is difficult to support the application in real scenarios. Therefore, if we want to use blockchain in real medical scenarios, then we have to optimize its performance.

2.3.2 Other BFT-Based Consensus Models

In [72], the advantages as well as disadvantages of several kinds of most commonly used PoW-based and PBFT state-machine replication-based approaches are compared, comparing their limitations in terms of fault tolerance, power consumption, throughput and scalability. The article concludes by discussing and presenting the current status and possible future directions regarding the development of consensus protocols.

[29] proposed Scalable Byzantine Fault Tolerance (SBFT) algorithm. It treats a node as a fixed, error-free block producer. SBFT has a theoretical performance advantage of twice the throughput and 1.5 times the latency optimization compared to traditional PBFT. It introduces the concept of collectors in its design and uses threshold signatures to reduce communication to linear, and also borrows the fast path approach from Zyzzyva to reduce a significant amount of communication cost. Although this scheme greatly improves the performance of the PBFT algorithm, there are some drawbacks in its design, i.e., it does not consider the case where the primary node is a Byzantine node, and the performance improvement of SBFT is to some extent at the cost of reduced Byzantine fault tolerance.

[40] presented Zyzzyva, which pioneered the mechanism of speculation to reduce the communication cost of BFT-type consensus protocols. In Zyzzyva, the original three-phase commit protocol is replaced by the speculation model. This means that all replicas optimistically adopt the order proposed by a primary node. In case of an error in any of the replica, the client node finds the inconsistencies at the end and helps them synchronize their responses to the existing state. In terms of performance, it can reach a theoretical throughput of 10,000, which is much higher than the traditional PBFT protocol, but since it is not used in systems prone to Byzantine nodes, the application scenario is greatly limited.

[49] proposed the Stellar Consensus Protocol (SCP). Unlike other Byzantine protocol models, it presupposes a list of members and opens the access to the list conditionally in order to promote the benign growth of the network. It reduces the complexity of consensus messages by replacing the public key signature with a vector of message authentication codes. And it can dynamically adjust the number of checkpoints, and the response time and space usage efficiency of the system is improved, but a part of the system security is sacrificed while the operation efficiency is improved.

The HotStuff [77] algorithm demonstrates the impact of structural optimization on the transmission complexity of PBFT. It innovatively changes the original network topology to a star network topology, so that each communication will only depend on the central node. Instead of reaching consensus through multiple rounds of multicast, it sends messages to the central node, which finally matches and verifies the messages and broadcast the consensus result to all the other nodes. in addition to providing $O(n)$ linear communication complexity, the Hotstuff algorithm also pioneered the optimization of view switching protocol complexity, which is not covered in any other optimization designs for BFT.

OBFT [66] was proposed in 2012 and it is a client node based BFT protocol. The O in its name stands for obfuscated, because replicas are not aware of each other's existence. In other words, compared to message communication between replicas in PBFT, message communication in OBFT exists between client node and replicas, which greatly reduces the original message multicast. This prevents non-semantic attacks to some extent, because the attacked replica does not know about other replicas, so it does not disclose information about other replicas. In the consensus process, the primary node in the algorithm uses the speculative mode to send a request to the $2f + 1$ selected active set, when the request is matched by all the receiving nodes, the request will be submitted by the client node; otherwise the recovery mode will be enabled and the $2f + 1$ nodes will be reselected and started again. OBFT achieves good performance, compared with PBFT's limited scalability and rapid decline in throughput as the number of clients increases, OBFT can achieve hundreds of nodes in a wide area network to participate in consensus at the same time. However, OBFT also has the limitation that it defaults the nodes participating in consensus are non-malicious also known as non-Byzantine nodes, in other words, the system allows node crashes but they cannot cheat.

FastBFT [48], proposed in "Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing" in 2019, leverages novel message aggregation and hardware-based trusted execution environment to reduce the consensus transaction complexity from $O(n^2)$ to $O(n)$. Unlike other methods also based on message aggregation, its approach is based on lightweight secret sharing, so it does not require public-key operations like multisignatures but requires additional hardware support such as Intel SGX, thus saving a lot of compuatation overhead. It optimized the fault-tolerance performance, compared to the original PBFT of $3f + 1$, fastBFT only needs $2f + 1$ replicas to tolerate f faulty nodes. FastBFT uses the basic binary tree topology to optimize the communication efficiency to a certain extent, which can improve the system operation efficiency when the children of the tree are not Byzantine nodes.

hBFT [19] is a hybrid PBFT protocol with optimal resilience, which was proposed by Sisi Duan, Sean Peisert and Karl Levitt in 2015. In the normal case hBFT uses speculation

and assume the default primary is correct, i.e., replica directly adopts the order from the primary and sends a reply to the client. And when there is inconsistency in the replica, it will be up to the user node to detect the inconsistency and help replicas to recover to the proper consensus result. It shifts the critical work to the client and has much less cryptographic operations than the original algorithm. However, its algorithm is based on assumptions of fault-free and normal cases, so the applicability scenario is greatly limited.

Chapter 3

Blockchain Healthcare Model

In this chapter, we will briefly describe the overall model of blockchain-based healthcare system, because our focus is on the optimization of the application bottleneck, i.e., the optimization of the consensus protocol, so we will only briefly describe the whole system architecture and not describe its technical details in detail.

3.1 System Model

3.1.1 System Architecture Design

As mentioned earlier, blockchain systems are mainly divided into public chains, private chains and federated chains. For example, MedRec [51] is a public chain electronic medical record management system based on Ethereum, but due to the performance bottleneck of the public chain structure, its actual operational throughput is only 30 tx/s, which is far from meeting the performance requirements of the actual medical scenario. On the other hand, for the private chain, although its operation performance is excellent, it does not meet the usage scenario of private chain due to the existence of Byzantine nodes in the medical scenario.

Therefore, as shown in Figure 3.1, for performance and security reasons, this thesis will use a consortium chain to design a medical blockchain application, using hyperfabric as the underlying architecture and optimizing its PBFT consensus algorithm, trying to solve its throughput and scalability limitations. The following diagram shows the application

scenario of blockchain medical system, in order to make it more concrete, we can consider the following five roles: patients, hospitals, research institutions, government medical institutions and insurance companies. Data Operation Module can be an Nginx proxy server used to deploy and coordinate the code between front-end calls and platform services, blockchain services are called through chain code services, and Decentralized Storage System is another chain composed of a group of nodes for storing medical files with large data volumes, for detailed concepts refer to Filecoin System [22]

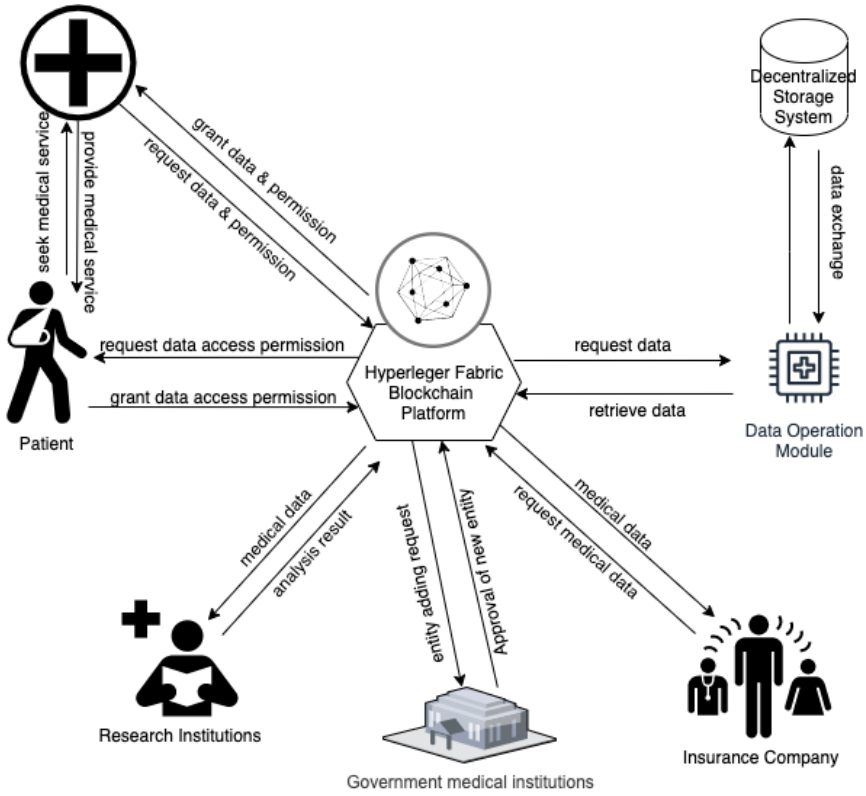

Figure 3.1: Blockchain Healthcare Model

3.1.2 Data Structure Design

First, as shown in Table 3.1 and 3.2, we designed the data structure of patient and medical data respectively, which simply satisfies the basic application of medical data management, but the data types and data relationships in the actual situation may be much more complicated.

Table 3.1: Patient Data Structure

serial	data field	data type	interpretation
1	p_id	int	patient's Id
2	p_name	string	patient's name
3	p_ref_num	string	Patient's unique medical reference number
4	p_nation	string	patient's nationality
5	p_bd	string	patient's birthday
6	p_addr	string	patient's current address
7	p_contact	int	the Id of patient's emergency contact
8	p_records	array	an array of all the medical records' Id

Table 3.2: Medical Record Data Structure

serial	data field	data type	interpretation
1	r_id	int	record's Id
2	p_id	int	the p_id associated to
3	r_timestamp	string	timestamp of creation
4	r_disease_type	string	type of disease
5	r_symptom	string	detailed symptom
6	r_file_link	string	side chain storage links for medical data
7	r_file_digest	string	data's digest
8	r_query_list	array	the Ids with access to the medical record
9	r_modify_list	array	the Ids with permission to modify the medical record
10	r_history	array	access and revision history

It is important to mention that p_id and r_id are the unique identifiers for patient and medical record respectively, which are used to distinguish different patients and different medical records.

Moreover, the relationship between the two data structures can be clearly seen in Figure 3.2: for a patient, he can correspond to a series of medical records; but for a medical record, there is only one related patient. Finally, for operational efficiency reasons, the huge amount of medical data will be stored in another distributed structure, which will be interconnected with the medical record by the file address and file digest.

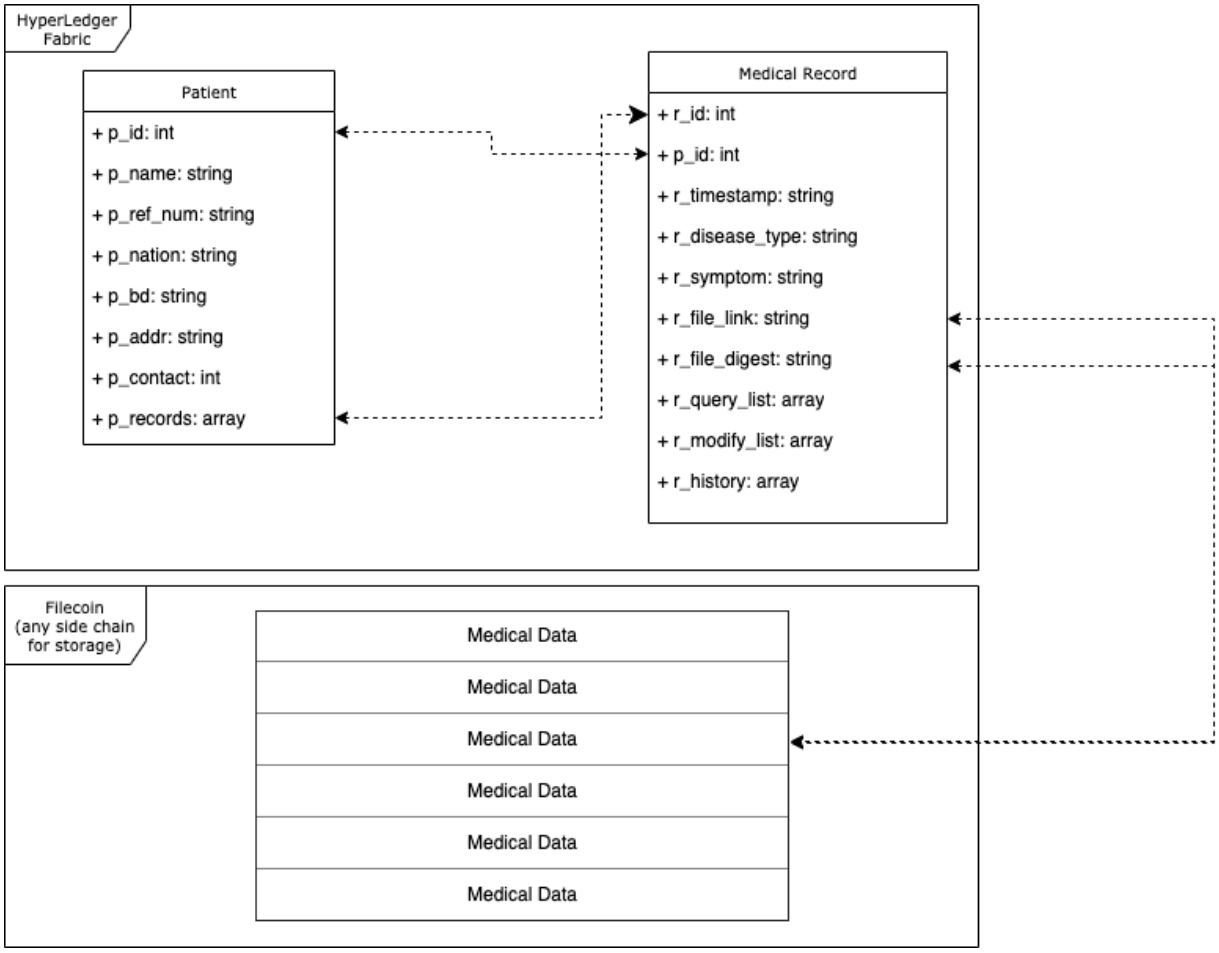

Figure 3.2: Structure Diagram for Medical Data Storage

3.1.3 System Flow Design

Figure 3.3 details the process of storing medical data to the chain. As mentioned above, for performance reasons, we first encrypt the data with the public key of the data owner, then upload it to the sidechain and return the corresponding address and digest.

After confirming that the information is not repeatedly submitted, we call the putState method of the hyperledger fabric to update the medical data records on the chain and return the corresponding end state code.

Figure 3.4 details the flow of retrieving medical data to the chain. When the system

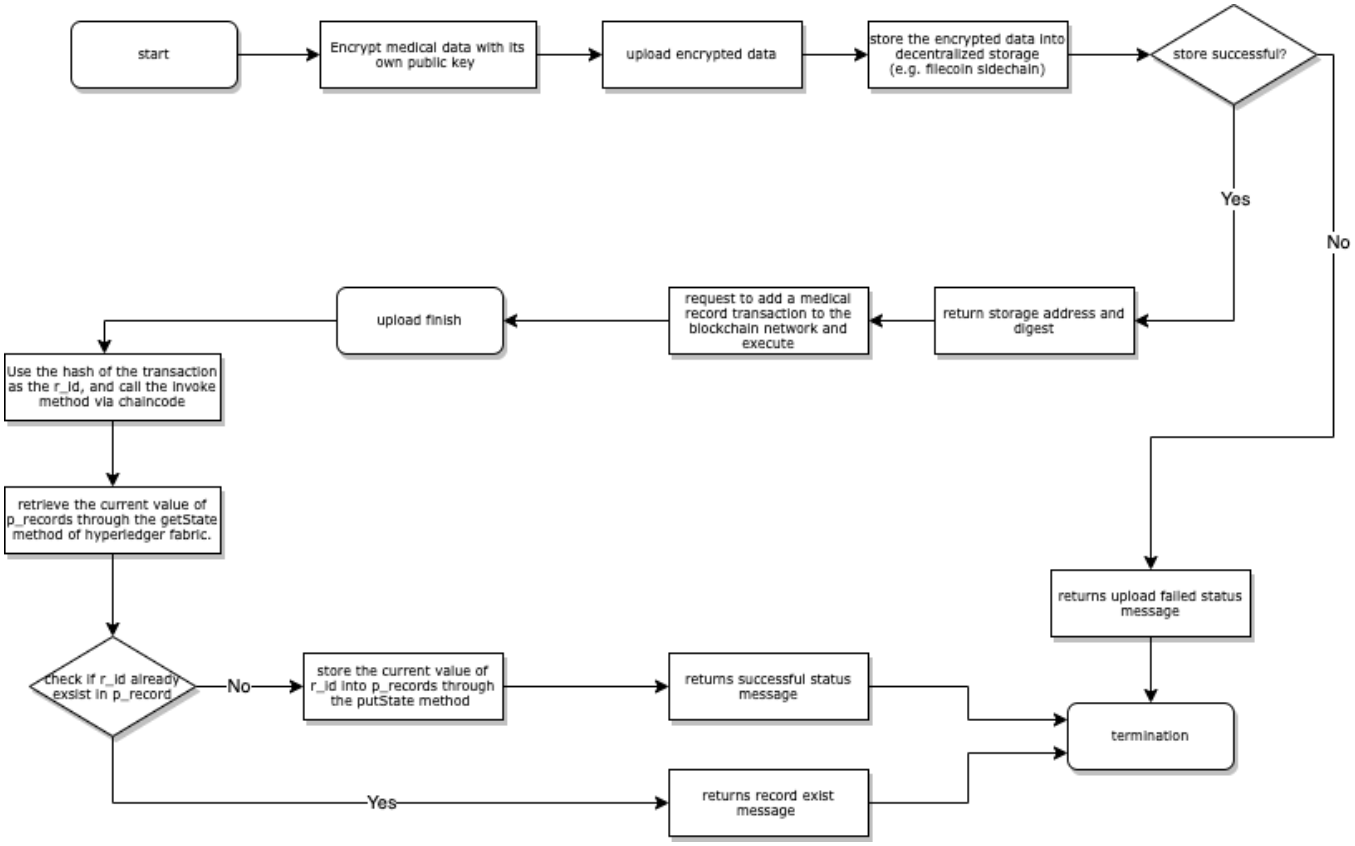

Figure 3.3: Flowchart of Storing Medical Data to the Chain

receives a data access request, it first needs to determine the identity of the data requestor and obtain the appropriate permissions. After the permission is granted, the system will use the getState method of the hyperledger fabric to get the address and digest of the data stored on the sidechain, and then it will retrieve the data and compare the digest to determine whether the data has been tampered with or not. In addition, it will call putState to update the data access record in time and return the corresponding termination state value at the end.

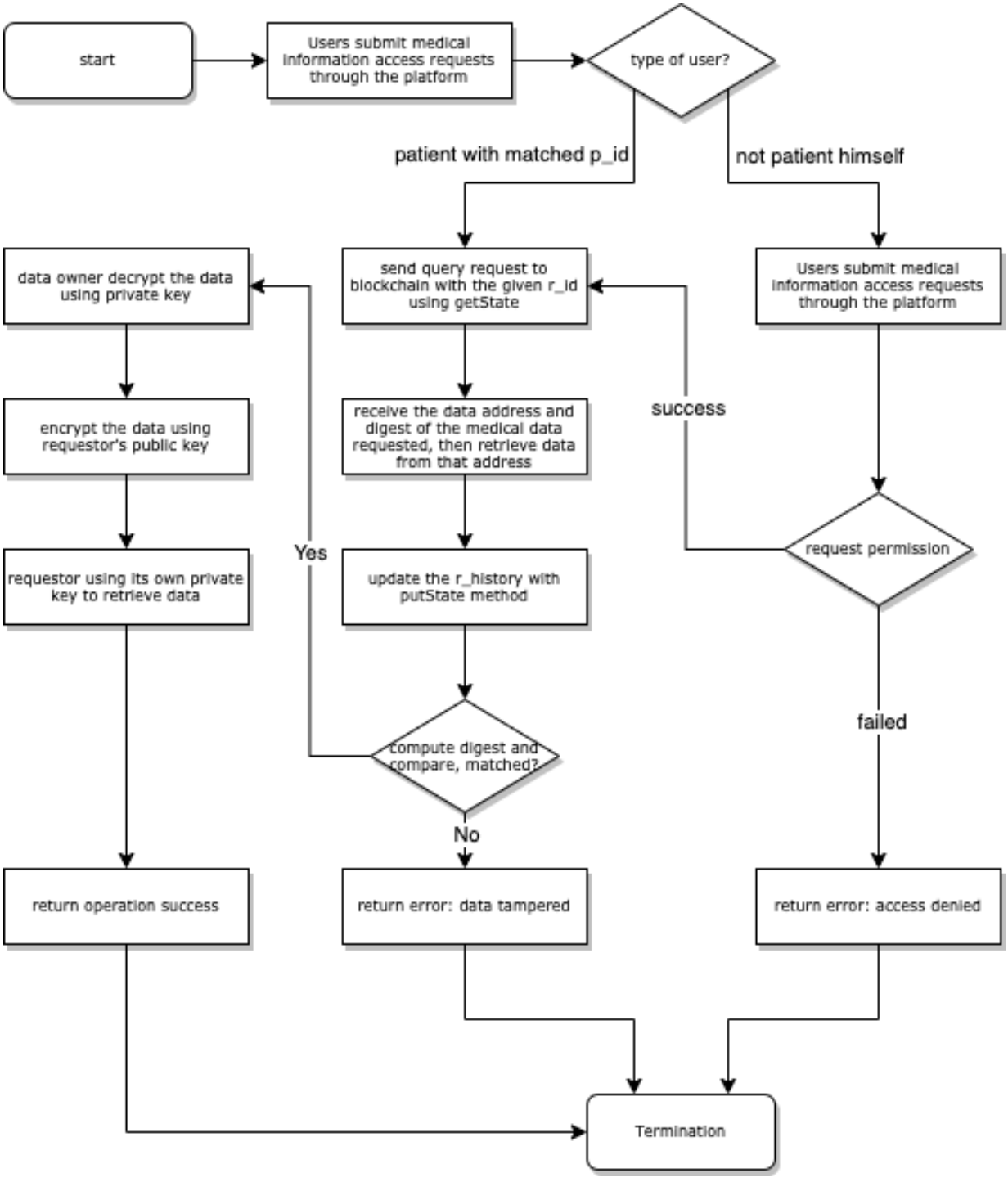

Figure 3.4: Flowchart of Retrieving Medical Data from the Chain

Chapter 4

Analysis for the Original PBFT

Due to the special nature of the design of medical blockchain information systems, we have to take Byzantine fault tolerance into account in the design, so we choose to explore the PBFT algorithm in depth. In this section, we will provide an in-depth background knowledge of the PBFT algorithm and the details of the algorithm, and analyze its shortcomings in practical application scenarios.

4.1 Background Knowledge

4.1.1 Practical Byzantine Fault Tolerance

PBFT [14] was first proposed by Castro and Liskov in 1999, and it is a good solution to the problem of the Byzantine faulty nodes in distributed systems, which leads to unsynchronized and inconsistent information. In other words, when the total number of nodes in the system is n and the number of malicious nodes is f, PBFT can guarantee the normal information communication of the whole system under the condition of $n > 3f + 1$.

4.1.2 The Two Generals Problem

Before we launch into an exposition of the discussion of the Byzantine problem, we first start with a more classical problem, the two generals problem, which was introduced back in 1975 in [4]. The two generals problem refers to the problem of two generals in an army whose units are separated by two locations, and they have to communicate with each other

through their communications to reach an unanimous decision to either attack or withdraw. However, there is no universal solution to this problem according to the FLP impossibility principle, since the communicators may receive the harassment from the enemy troops in the middle of the process leading to the loss of secret messages.

4.1.3 The Byzantine Problem

As an extension of the dual generals problem, the Byzantine problem is more in line with the model characteristics of asynchronous distributed systems. The Byzantine problem was first formally introduced by Leslie Lamport in 1982 in the paper "The Byzantine Generals Problem" [45], while Byzantine is the capital of the ancient Eastern Roman Empire. The generals are referring to the nodes in the model of the distributed system. Since there may be traitors among the generals who may refuse to convey messages or even try to convey wrong orders to interfere with the consensus, the Byzantine problem is to solve the problem that the participants of consensus protocols in asynchronous distributed systems may appear to be faulty.

4.1.4 Preliminary Ideas

In the paper "The Byzantine Generals Problem" [45], it is mentioned that if we assume that there are n nodes and f faulty nodes under the asynchronous system in the Byzantine problem, then the whole system can reach the correct consensus result only when $n > 3f+1$. Let's take a more practical example, there are 3 nodes and 1 faulty node in the system, when the proposer sends a consensus proposal, the faulty node may give the opposite result to interfere with the consensus, then the last remaining node may receive one positive and one negative message, in this case it is difficult to discern who is right and who is wrong, so we cannot get the consensus result. Let's generalize this example, similarly, when a node sends consensus proposal A and it is not an evil node, for a non-faulty node he will see $n-f$ consensus messages A, and at most f consensus messages that is not A, denoted as $\overline{A}$. Then obviously only when $n-f > f$, that is, when $n > 2f$ we can get the correct consensus result.

What if the consensus proposal is made by the evil-doer? From the standpoint of the evildoer, in order to interfere with the consensus result, it will send contradictory results to $n-f$ participants, then there will be $(n-f)/2$ participants receiving the message A and another $(n-f)/2$ participants receiving the message $\overline{A}$. These honest participants will think that the system currently has half of the A consensus message and half of

Table 4.1: Relationship between Transaction Volume and Rounds

# round	# message visited	tolerate ability	message spread	# messages
1	1	f	$n-1$	$n-1$
2	2	$f-1$	$n-2$	$(n-1)\cdot(n-2)$
...	...	...	...	...
x	x	$(f+1)-x$	$n-x$	$(n-1)(n-2)\ldots(n-x)$
$x+1$	$x+1$	$(f+1)-x-1$	$n-x-1$	$(n-1)(n-2)\ldots(n-x-1)$
$f+1$	$f+1$	0	$n-f-1$	$(n-1)(n-2)\ldots(n-f-1)$

the $\overline{A}$ consensus message regardless of the result they receive. Moreover, in addition to the proposers themselves being evildoers, there are $f-1$ evildoers remaining in the system, whose behavior is uncertain; they may send consensus message A, or they may send consensus message $\overline{A}$, or they may even refuse to participate in the consensus. Then, if honest participants want to get a consistent consensus result, they have to communicate with each other to confirm and count the consensus messages they receive, and select the majority result as the consensus result.

In a subsequent paper by Leslie Lamport entitled "Reaching agreement in the presence of faults" [61], it is shown that we can negotiate a consensus through at most $f+1$ rounds of interaction when the perpetrators do not exceed 1/3 of the total number of nodes in the system just as shown in Table 4.1. Otherwise, the consensus result is not guaranteed.

Imagine that for a consensus A, there are f evil nodes and t honest nodes in a system with n total nodes, where f evil nodes can give the opposite result or refuse to participate in the consensus. When all the evil nodes give the opposite result $\overline{A}$ and f of the t honest nodes refuse to give a consensus response, the remaining $t-f$ honest nodes must have $t-f>f$, and because $t=n-f$ so $n-f-f>f$, so finally our Byzantine fault tolerance is $n>3f$.

4.1.5 Types of Byzantine Failures

There are several types of Byzantine Failures, and we can simply divide them into nodes that are actively evil and nodes that are passively evil [27].

Active byzantine nodes may be adversary nodes used to try to steal user information or deliberately send incorrect consensus votes, and they may use a large number of byzantine nodes to frequently trigger attempts to switch protocols and thus affect the overall operational efficiency of the system.

There can be many kinds of passive nodes, the most typical ones being those that fail to return a result in time due to natural disasters or lack of maintenance resulting in long or temporary downtime.

4.2 PBFT Protocols

We can also use a simple language to explain the design concept of PBFT: Suppose I, as a member of the consensus nodes, receive a message from the leader, I will first subjectively judge it right or wrong and reject those messages that I think are wrong. But even if I think the message is right, I will still ask and refer to other nodes' judgment. Only when I count the existence of more than $2f + 1$ people who believe that the leader's message is correct, will I approve it and execute it. When the majority of nodes believe that the leader is wrong, we need to re-elect a new leader.

Basically, PBFT is divided into three main parts: Consistency Protocol, View Switching Protocol, Checkpoint Protocol

4.2.1 Consistency Protocol

Consistency protocol is the core component of the entire PBFT algorithm, which ensures the consistency of information across all nodes in the blockchain nodes. It divides the communicating nodes on the chain into 3 categories, client nodes, primary nodes and replica nodes. where the client node is the initial sender of the request and it sends a request to the primary node $< REQUEST, o, t, c >$, where o is the specific operation, t is the timestamp. Once the primary node receives the request from client code, it need 3 interactive stages before it replies the client code, they are pre-prepare stage, prepare stage and commit stage.

- In the pre-prepare stage, when the primary node receives a request, it first analyzes the request and discards the incorrect requests, then it sorts the remaining correct requests and assigns the number n in order, and then broadcasts $<<$ PRE-PERPARE$, v, n, d >, m >$ messages to the other replica nodes in the system. Where d represents the summary of the REQUEST message, n is the number of the message m, and v represents the view number.

- In the prepare stage, node i, which receives the pre-prepare message, first determines whether it agrees to enter the prepare stage and verifies the accuracy of the message, and broadcasts $<$ PREPARE$, v, n, d, i >$ messages to other nodes including the primary node.

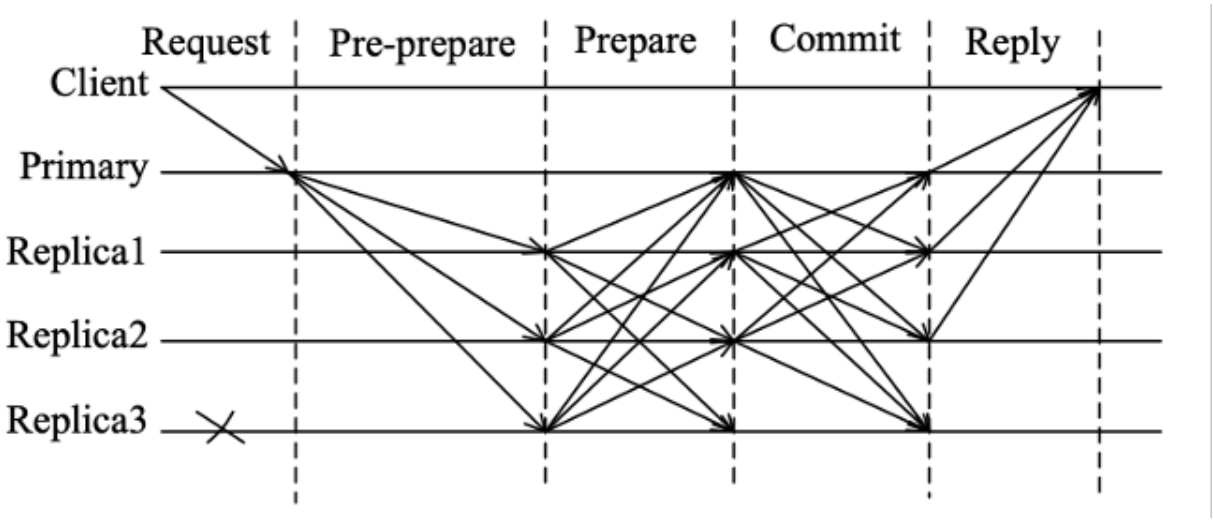

Figure 4.1: Practical Byzantine Fault Tolerance Communication Graph

- In the COMMIT stage, when node i receives $2f+1$ validated messages, it goes from the PREPARE stage to the COMMIT stage and sends $<$ COMMIT$, v, n, D(m), i >$ messages to other nodes including the PRIMARY node.

- In the reply stage, when node i receives $2f+1$ commit messages to meet the consistency requirement, it then sends a consensus message in the format of $<$ REPLY$, c, I, v, t, r >$ to the client node. r represents the result of client request execution by node i. Finally, this consistency interaction is completed when the client node receives at least $f + 1$ REPLY messages. After that the PRIMARY node needs to write the consensus data into the blockchain. If this data block is not generated, then this primary will be classified as a Byzantine node and the system will select a new primary node according to the view-change mechanism.

4.2.2 View-Change Protocol

The concept of view ensures that each node in the blockchain works under the same configuration information. When one of the following conditions is met, the primary node will be considered as a Byzantine node and the system will select a new primary node according to the view switching protocol:

- No pre-prepare broadcast message is received from the primary node at time t_1

- No new module is generated within the time limit $t_2 > t_1$

After triggering the protocol for switching views, we first generate a number for the new view, $v' = v + 1$, then select the new primary node based on $p = v' \bmod n$, and broadcast the view change message to all replica nodes.

Then when each replica node receives $2f + 1$ view change messages including its own, they will send a view-change-ack confirmation message to the new master node of the new view, and then the new master node will enter the new-view phase.

Finally, the new master node selects the checkpoint as the starting state of the new-view request and then executes the consistency protocol according to the local data block.

4.2.3 Checkpoint Protocol

In the process of consensus generation, a large amount of stored data is generated. The periodic operation of the checkpoint protocol largely reduces the data storage size of nodes and avoids the waste of resources. We will not go into details here.

4.3 Advantages of PBFT

- **A certain degree of waste of resources is avoided.** PBFT does not require as much computing power as other distributed consensus protocols, such as those based on Proof of Work. Therefore, PBFT is a relatively energy-saving approach.
- **PBFT provides transaction finality.** As the name suggests, PBFT-based distributed systems eliminate the need for multiple validations after a request is submitted, unlike traditional Bitcoin, which requires each node to validate all transactions when adding a block to the local area; once a request is made, we can be sure that the request will generate a response within a limited time threshold.

4.4 Bottleneck of PBFT Performance

The whole process requires two multi-casts for all nodes, which severely consumes the total number of communications and slows down the throughput, causing a large degree of resource consumption. In PBFT, the communication cost of each step is as follows:

1. the communication cost of request phase is 1,
2. the communication cost of pre-prepare phase is $n - 1$,
3. the communication cost of prepare phase is $(n - 1)(n - 1)$,
4. the communication count of commit phase is $n(n - 1)$,
5. the communication count of reply phase is n.

Therefore, the overall complexity is $2n^2 - n + 1 => O(n^2)$. For a network model with a small number of nodes, e.g., n less than or equal to 10, the overall data communication volume will not be very large, but considering that the number of nodes in today's real-world application scenarios is often up to 1000, the squared time complexity is certainly not enough.

Chapter 5

Potential Optimizations for PBFT

Based on the needs of the business scenario of consortium blockchain electronic medical record system, high performance is one of the goals it must achieve. Therefore, the performance optimization of the consensus mechanism needs to be a top priority.

We optimize the blockchain consensus model based on PBFT for the hierarchical characteristics of the healthcare system. First, we can think of a simplified distributed healthcare information sharing scenario where the identities can be divided into hospitals, clinics, insurance companies and patients. Each of these identities has the risk of becoming a Byzantine node, for example:

1. the hospital node may be the main target of an attack since it is where a lot of patient information and medical resources come together
2. The information recording process of the clinic node may be inaccurate and non-compliant, resulting in incorrect information
3. the insurance company may deny the facts and reject an insurance claim for its own benefit
4. adversary may use the fake patient nodes for sybil attack [18]

5.1 Optimization by Role Grouping

Initially, our first idea is to divide all nodes into i roles. For example, in the above medical example, we divide all nodes into 4 groups, hospital nodes, clinic nodes, insurance company

nodes and patient nodes. Thus, we only need to select a total of i sub-primary nodes in the node subset of each group, and then pass the consensus information from these i sub-primary nodes to the client node of the message requestor.

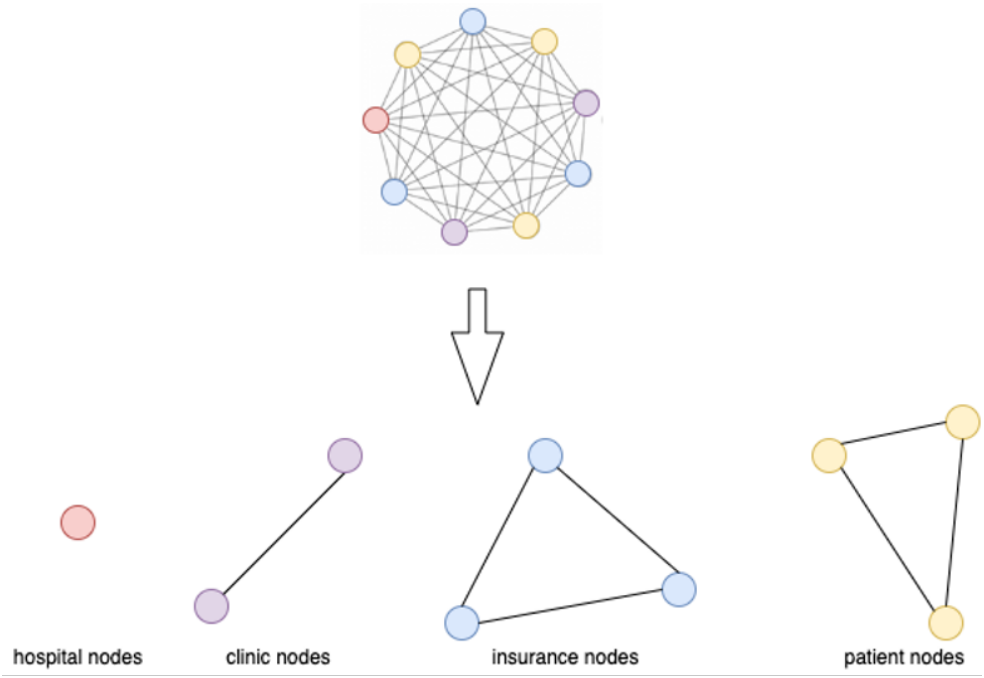

Figure 5.1: Grouped Network for Medical Example

Assume i groups are evenly distributed. In this first attempt, the communication cost of each step is as follows:

1. the communication cost of request phase is i,
2. the communication cost of pre-prepare phase is $(\frac{n}{i} - 1) * i$,
3. the communication cost of prepare phase is $(\frac{n}{i} - 1)(\frac{n}{i} - 1) * i$,
4. the communication count of commit phase is $\frac{n}{i} * (\frac{n}{i} - 1) * i$,
5. the communication count of reply phase is $\frac{n}{i} * i + i$.

Therefore, the overall complexity is $\frac{2}{i} * n^2 + 2i - n => O(n^2)$, since i is a fixed number

The result is clearly within our expectation, even though it is $O(n^2)$ as before, but its total complexity is i times smaller. When the number of nodes is large, it has to be said that the grouping approach is an effective improvement for the operational efficiency of the system.

In addition, we subsequently come up with the thought that in a medical scenario, since most nodes are "bona fide" by default, fault tolerance does not necessarily need to be as high as 1/3, a decent tolerance rate with 1/6 or even 1/10 is still sufficient for the real-world scenario. Therefore, we next conducted a study on the feasibility of sacrificing fault tolerance for throughput in the specific context of medical applications.

5.2 m-ary Tree Structure Optimization

It is known that the efficiency of dividing nodes into groups can be optimized by multiplying the number of divisions, so we try to make the use of grouping more generalized. We first think of using some data structures to optimize it, such as m-ary tree, because the results of optimization with trees are often in log level which is better than $O(n^2)$. m-ary tree in traditional graph theory refers to a tree storage structure with the number of children less than or equal to m. We often refer to a binary tree as an m-ary tree when $m = 2$ [30].

Here we consider a complete m-ary tree as the specified data structure, which fills up the space of each level as much as possible before building the next level. Unlike traditional PBFT, we put the original node into the data structure of complete m-ary tree as shown in Figure 5.2, compared to the original PBFT that only has one primary node, we need to select a sub-primary in each subsets to act as the original primary node but in the subsets, and pass the consensus result of the group to the upper layer, so that the consensus result is passed on the line iteratively until it is passed back to the uppermost client node.

To facilitate our quantitative analysis, we then assume:

1. the total number of nodes is n;
2. for the convenience of evaluation, we assume the nodes can be perfectly fit into a m-ary tree, which means each node in a tree has m branches;
3. we assume on-chain configuration settings;
4. for the convenience of calculation, we temporarily set the sub-group size to be also m.

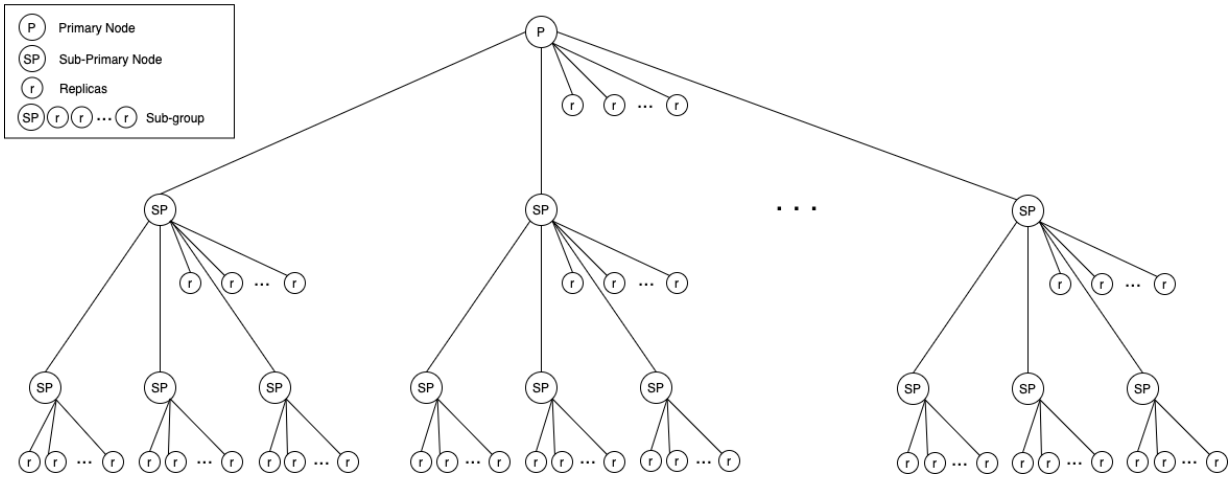

Figure 5.2: A m-ary tree structured PBFT

Then for each level of the tree, we have:

Tree level 1:
number of groups in this level $= m$
number of nodes in this level $= m \cdot m = m^2$
Tree level 2:
number of groups in this level $= m^2$
number of nodes in this level $= m^2 \cdot m = m^3$
...
Tree level t:
number of groups in this level $= m^t$
number of nodes in this level $= m^{t+1}$

$$\text{Total number of nodes } = m^2 + m^3 + m^4 + \ldots + m^t + m^{t+1} = \sum_{k=2}^{t+1} m^k$$

Suppose we have n nodes in total and use an m-ary tree, we also set the size of sub-group to be m for now. We then set the number of communications within the subsets to a multi-cast which is m^2 for ease of calculation.

In this case, if we divide all nodes into 4 layers with the current data structure as we just did, we get:

$$\text{total number of nodes} = (m^5 + m^4 + m^3 + m^2) = n \tag{5.1}$$

$$\text{total communication volume} = \text{number of groups} * \text{communications in the subsets} \tag{5.2}$$

$$= (m^4 + m^3 + m^2 + m) * m^2. \tag{5.3}$$

When $m = 4$, total communication volume $= 340 * 4^2 = 5440$, this can contain number of nodes $= 1360$. For a traditional PBFT, the overall transaction is $2n^2 - n + 1 = 3697841$, which is 680 times bigger! Even for a 4-group medical PBFT as we proposed in the former section, the overall transaction will be $(2/m)n^2 + 2m - n = 923448$, where m is the number of groups that we have defined in last section.

Based on the nature of m-ary trees in graph theory, we wish to further quantify the overall operational efficiency. We know that for a complete m-ary tree with a total of n nodes, its height is $\log_m n$. Therefore, we have:

$$\text{number of groups} = \text{number of PBFTs} = \text{number of sub-primary nodes} \tag{5.4}$$

$$\begin{aligned} &= m + m^2 + m^3 + m^4 + ... + m^t \\ &= m(1 - m^t)/(1 - m) \qquad \text{where } t = \log_m n \\ &= m(1 - n)/(1 - m) \\ &= m(n - 1)/(m - 1) \\ &= O(n). \end{aligned} \tag{5.5}$$

Recall that number of transactions inside each group $= O(m^2)$ So, in overall, the communication complexity $= O(m^2 * n)$.

5.2.1 Consensus Protocol under m-ary Tree Structure

In the previous section, we briefly introduced the general structure of our m-ary PBFT and estimated its operational efficiency. In this part, we will explain the specific steps of the consensus algorithm in more detail.

Step 1: the Client node N_{client} sends a message $<$ REQUEST, $0, t, N_{\text{client}} > \sigma_{\text{client}}$ to the Primary node who is the root of the m-ary tree, where o is the specific operation, t is the timestamp.

Step 2: After receiving the request, the Primary node firstly analyze the correctness of the request, then sorts the remaining correct requests and broadcast a message $<<$ PRE-PERPARE, v, $n, d > \sigma_{primary}, N_{primary}, m >$ to all of its child nodes, where v is the current series number of the view, d is the signature of the message and m is the request message itself. If the child node in this propagation is also the sub-primary node of the next subnet, it continues to broadcast the prep message to the lower layers in the same message format but with the corresponding view number and message digest. This process is repeated until they reach the bottom layer and all its sub-primaries.

Step 3: After receiving the pre-prepare message, the replica nodes in the bottom layer verify the message m and digest d and compare the hash values, and verify whether the view number v matches the sequence number n and has been processed, whether the source of the message is its own parent node and so on. If the above verification passes, the corresponding replica nodes enter the PREPARE phase and broadcast $<$ PREPARE $, v, n, d, i > \sigma_i$ to the other replicas in the group and adds both messages to its log, where i is the series number of the node inside its group. Otherwise, if verification failed, it would do nothing.

Step 4: After the nodes in the bottom-most layer (including the master node) collect $2f + 1$ matching PREPARE messages, they go from PREPARE stage to the COMMIT stage and broadcast a confirmation message $<$ COMMIT, $v, n, d, i > \sigma_i$ to other nodes in the group including the sub-primary node in order to confirm that the view number and sequence number has not changed.

Step 5: If the child nodes in the bottommost sub-group have collected $2f+1$ commit messages to meet the consistency requirement, they send a reply message $<$ REPLY, $v, t, c, i, r > \sigma_i$ to the sub-primary node of the group, where r represents the result of client request execution by node i.

Step 6: Each sub-group in the system counts the number of REPLYs they receive in real time, and if there are $f + 1$ consistent results, they confirm the intra-group consensus result as r and continue to pass it to the upper subnets. If any sub-primary node is found to be evil or not responding in timeout or view number change during the process, a consensus failure message $<$ FAILED, $n, d, r, i > \sigma_i$ is broadcasted down

to the bottom subnet and the consensus fails. If the above contingency does not occur, the result set r of sub-groups is sent to the upper subnet, and the process is repeated up to the highest subnet. The primary node in the highest-level subgroup counts whether it receives $f+1$ matching consensus results and provides feedback to the client node c.

5.3 Node Rating Mechanism

However, it is easy to see that since we need to elect a sub-primary node in each subset, while the traditional PBFT only needs to elect one primary node, our fault tolerance will be largely affected by the probability that the sub-primary nodes are Byzantine nodes, so we need a more sensible sub-primary node selection mechanism and node rejection mechanism.

Since the selection of primary node and voting node in the original PBFT model is random, we want to optimize the selection of node roles based on the different trustworthiness of each level of the healthcare system in order to reduce the waste of transaction amount.

We can establish a reputation hierarchy to set the promotion and elimination conditions for different nodes, and dynamically change and evaluate the credit of nodes based on their past performance, e.g., increase by 1 each time the node information is verified to be correct, and clear the points once the node response timeout or malicious node is encountered.

We can establish the credit threshold L1 and L2 as high credit threshold and low credit threshold respectively, when the credit score is higher than L1, it means the node has an excellent past record and can be selected as the primary node; when the credit score is lower than L2, it means the node has a bad record and will be eliminated from the consensus node and will not participate in the consensus but need to accept the consensus result until its score is higher than L2, which can largely reduce the communication overhead of switching nodes and improve the operational efficiency.

Credit Tier	Range	Character
Excellent	$[L1, 1]$	can be selected as primary or sub-primary nodes
Good	$[L2, L1)$	can participate in consensus protocol voting
Bad	$[0, L2)$	cannot participate in voting, but only accept results

In our design, in order that the concept of credit value can be quantified and dynamically accumulated or deducted, we use the sigmoid function to trap the value within the real number interval from 0 to 1 . Due to the nature of the sigmoid function, the growth rate of its credit value is very slow in the low credit and high credit intervals, which well increases

the difficulty of malicious nodes to intentionally swipe high credit values, and the difficulty of normal nodes to enter the high credit interval to become sub-primary key nodes. This approach can prevent sybil attacks to some extent and also ensures the credit accumulation efficiency of normal operating nodes in the intermediate interval.

Set credits will be dynamically accumulated with the "view change" event, corresponding to the current view v and the new view $v+1$, whose credits can be expressed as C_v and C_{v+1} respectively.

1. If this master node successfully generates a valid block, or the message content of the nodes participating in consensus is verified to be consistent with the consensus result, then we will increase its credit accordingly.

$$C_{v+1} = \text{Sigmoid}(C_v + a)$$

where a is the preset parameter for the credit growth rate, c_1 and c_2 are adjustable parameter for the cumulative difficulty of the low and high credit zones, as shown in Figure 5.3.

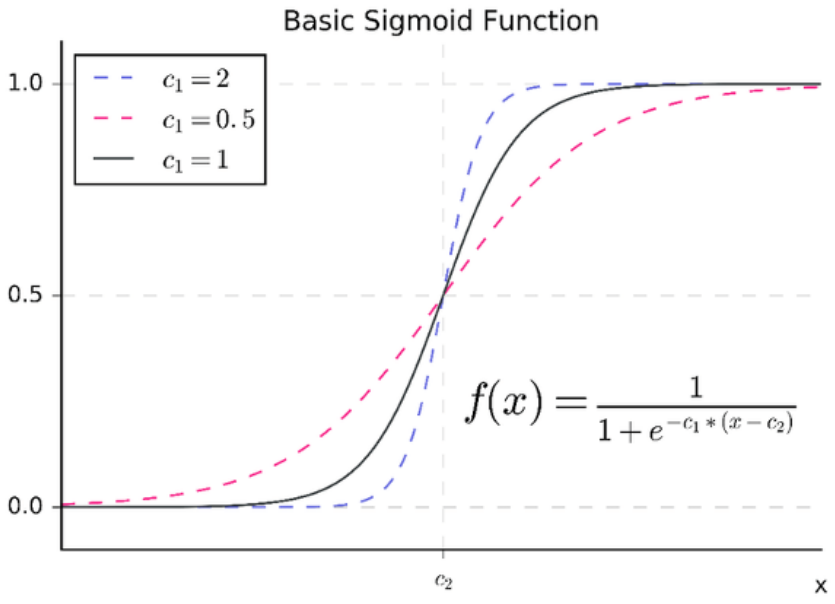

Figure 5.3: Sigmoid Function [47]

2. If the response wait of the sub-primary node exceeds the current exponential fallback delay value, or if the new block generation fails, or if the messages of the nodes participating in the consensus are inconsistent with the final consensus result or even do not respond successfully, then its credit value needs to be adjusted downward.

$$C_{v+1} = \text{Sigmoid}(b * C_v)$$

where $0 < b < 1$ is the tolerance factor of credit to the faulty nodes.

5.3.1 View Switching Protocol

The attempted switchover protocol exists to ensure that when the primary node goes down, the system can recover from the failure in time to ensure liveness.

The view switch is triggered by the timer of the replicas under the primary node, so that replica can know the unresponsiveness of its primary node and initiate a request to replace the primary node. After each view change is triggered, the corresponding delay T is doubled in order to accommodate different network conditions.

In a hierarchical tree PBFT, it is clear that the presence of a faulty sub-primary node is fatal, so the goal of our view switching protocol is not only to detect the faulty node and replace its sub-primary role and select a successor based on its creditworthiness, but also to try to push it to a lower level sub-node in order to reduce its impact on the tree structure.

Step 1: If a replica within a group finds that its sub-primary node has a response timeout or is identified as a Byzantine node, we update the credit value of the node using the credit mechanism described above. Then all its subordinate child nodes will be evaluated for intragroup credit, and if the current sub-primary node of the group is not the node with the highest credit within the group, a view replacement message $<$ VIEW-CHANGE$, v+1, n, i > \sigma_i$ is broadcasted within the group.

Step 2: After the slave node with the highest credit value in the subnet collects $2f + 1$ matching view replacement messages (including itself), it will broadcast a new view message $<$ NEW-VIEW$, v+1, s, i > \sigma_i$ within the subnet, where S is the set of $2f+1$ view replacement messages. At the same time, it will broadcast a new view message $<$ NEW-VIEW$, v' + 1, S, sp, C_{sp}, i > \sigma_j$ to the next level of the subnet it leads to prove that the new master node is valid after this view replacement, where v' is the current view of the lower level subnet, sp is the replaced subprimary, and C_{sp} is the reputation value of the replaced master node.

Step 3: After receiving the new view message, other nodes in this subnet verify the correctness of the signature and the validity of the view replacement message, and then add the new view message to the message log and enter the view $v+1$. After receiving the new view message, the slave nodes in the next subnet verify the validity of the message, and if the message is valid, check whether there is a slave node in the subnet whose reputation value is greater than the reputation value of the dispatched master node. in the subnet. If not, the new view message is stored in the log and the view $v'+1$ is entered. Otherwise, the view replacement protocol is triggered again in this subnet range.

Step 4: Repeat steps 1 to 3 until the failed node becomes the master node of a lower-level subnet or becomes a slave node of the lowest-level subnet.

This credit-based view switching protocol can largely avoid Byzantine nodes being selected as sub-primary, so when the whole system has been running for a period of time and maintains a stable structure, the system can slowly reduce the chance of view switching protocol triggering and maximize the operational efficiency of the system, and under this stable condition, the fault tolerance will be infinitely close to $f = (n-1)/3$ in the original PBFT.

In order to ensure the efficient operation of the whole m-ary tree, it is not difficult to find that it is very inefficient to gradually adjust the structure of the tree just by view switching, which makes the whole system gradually enter the stable state very slowly. However, if we want to reach a stable state as quick as possible, a good initialization state and regular full node checking are essential. In this case, we can ensure that the high credit nodes are assigned to the relatively high layer and ensure the stability of operation efficiency. As we know that low credit nodes have more possibility to become Byzantine nodes, if we place them in lower layers, their impact on the consensus result and view-change latency will be minimized if they have problems.

For the above reasons, we try to use an efficient m-ary heapify algorithm [16] as shown in Figure 5.4 and 5.5 to dynamically sort the nodes according to their creditworthiness with a complexity $O(m \log_m n)$. Whenever the system needs to initialize the m-ary tree, or when it is in idle state for a long time, or when the view-change is triggered more frequently than a preset threshold, or any node has been added or deleted from system (will be discussed in the next section), we can trigger the m-ary heapify to assign nodes according to their credit from highest to lowest.

For simplicity of illustration, suppose nodes are stored in a datatype A just like an array and node can be access by its serial number i where the value of the node is its credit, here is the specific algorithms for sorting this m-ary heap:

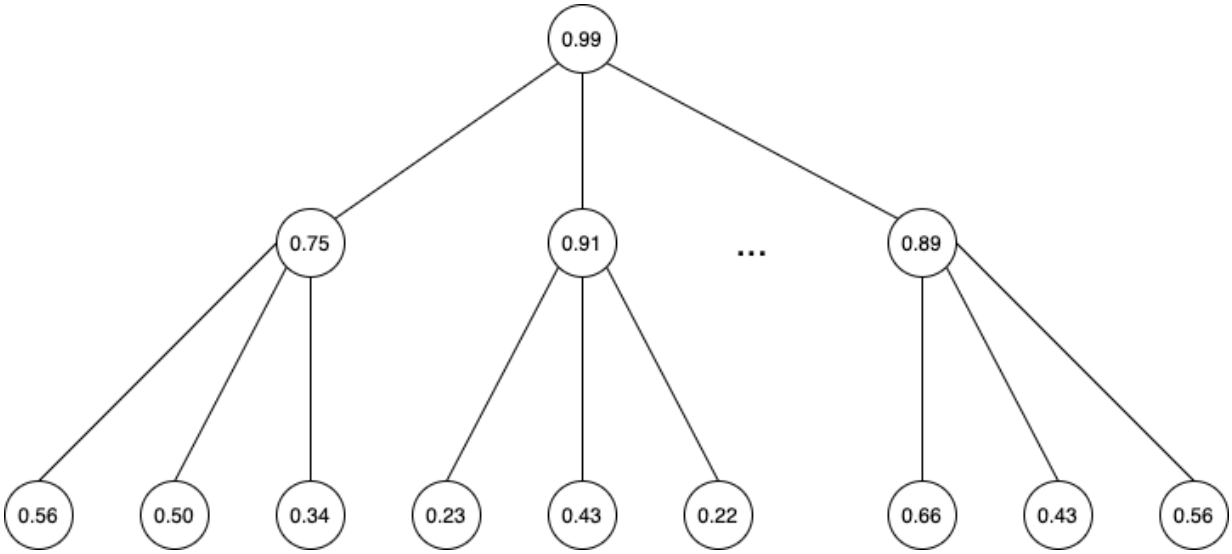

Figure 5.4: m-ary PBFT before Heapify

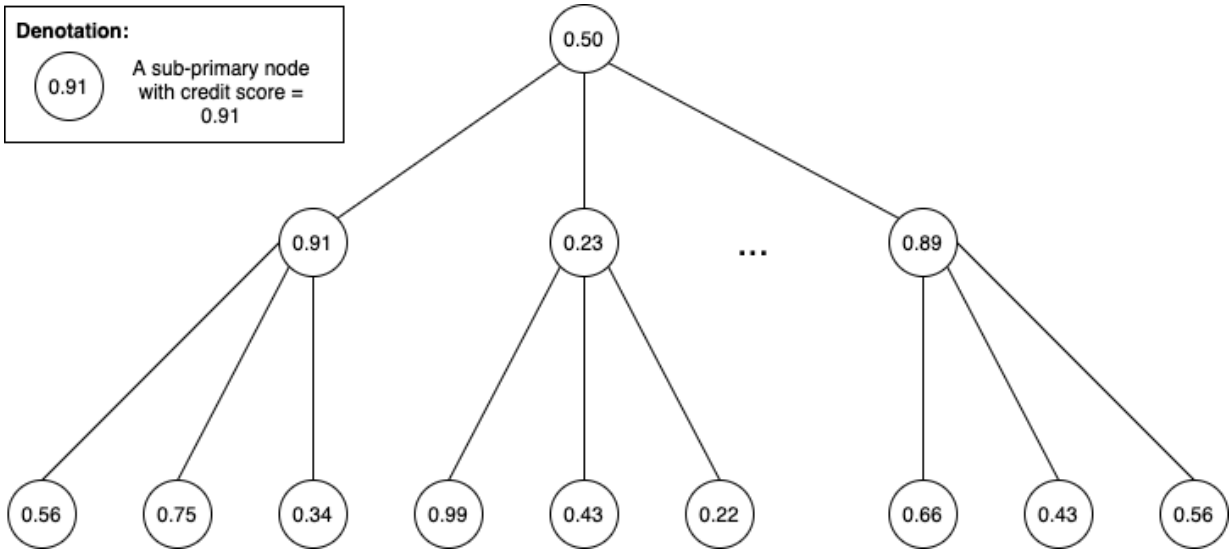

Figure 5.5: m-ary PBFT after Heapify

Algorithm 1: Heapify(A, i, m)

```
max ← i;
for j = 1 to m do
    if j ≤ heap − size[A] and A[Child(i, j) > A[i] then
        max ← Child(i, j)
    end
    if max ≠ i then
        exchange A[i] with A[max];
        Heapify(A, max, m);
    end
end
```

Algorithm 2: Build_Heap(A, m)

```
heap-size[A] = length[A];
for i = ⌊((length[A]−2)/m + 1)⌋ downto 1 do
    Heapify(A, i, m);
end
```

Algorithm 3: Sort_mary_Heap(A, m)

```
Build_Heap(A, m);
for i = length[A] downto 2 do
    exchange A[1] with A[i];
    heap-size[A] = heap-size[A] - 1;
    Heapify(A, 1, m);
end
```

5.4 Improving the Scalability and Dynamic Registration of Nodes

The three optimizations approaches mentioned above are not contradictory and can be mixed together.

However, since more layers also mean more waiting time, in other words the upper layer has to wait for the results of the lower layer, in time for parallel computation, some

of the interactions of different layers can be run simultaneously but overall we still increase the latency for less transactions. This means that our model is cheap and scalable, but also relatively slow, which is suitable for the needs of current medical data management systems. Of course, we can use concurrency to implement simultaneous computations between different layers, but there is still a degree of dependency between layers, which leads to latency growing with the size of the tree. Therefore, in order to maximize the parallel efficiency of the entire system and minimize the impact of hierarchy on latency, we must control the size of tree and optimize its parallel computing efficiency for cases with large number of nodes.

5.4.1 Random Forest for m-ary PBFT

We are inspired by the algorithm of Random Forest [33] in artificial intelligence algorithm for training multiple decision trees to generate consensus results, we can generate multiple m-ary trees by randomly selecting a subset of reputable nodes from the node pool based on different categories, each with a different node selection, and generate consensus results based on majority. In this way, due to the unique data structure and node audit mechanism, together with the fact that the probability of Byzantine tree appearance itself is already very small, this random forest-like approach introduces random sampling of reputable nodes and is able to dynamically exclude inferior nodes from the consensus nodes, so that they do not participate in the consensus, but only accept it afterwards, thus the overall algorithm receives minimal influence from Byzantine nodes. In addition, the multiple consensus trees model of random forest can largely address the concept of concurrency since multiple consensus trees can be running concurrently hence improves the performance and scalability.

5.4.2 Dynamic Join Protocol

Another fatal flaw of the original PBFT is its inability to dynamically add new nodes. To remedy this drawback to accommodate the increasing number of nodes in the healthcare environment, we consider a special trusted node to manage the registration information of all nodes. The concept of this special node can be pooled, in other words it can be a common behavior of multiple extremely trusted nodes. This approach may be inappropriate in a business scenario because there is no guarantee that large companies or companies with good credit will not modify information or compete maliciously for increasingly, but in a medical scenario, for example, the government medical departments of each province can be

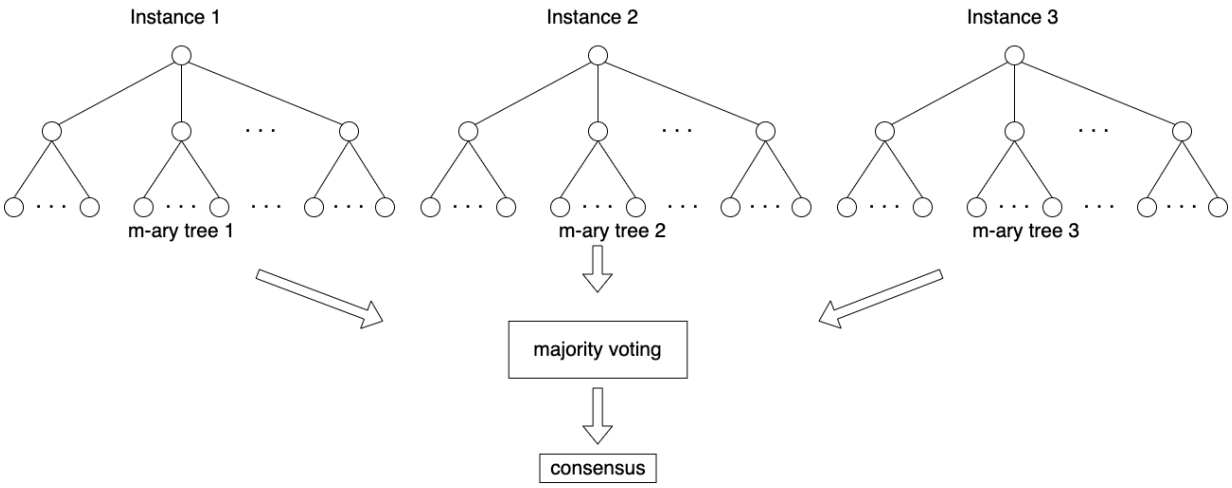

Figure 5.6: Random Forest for PBFT

identified as fully trusted nodes. However, these government nodes may also suffer DDOS attacks or catastrophic downtime, so the fully trusted node here is a pooling concept, which can be a consensus group of medical nodes in multiple provinces.

- First of all, if a Node i wants to apply to join this PBFT consensus, it must first send an application to the Absolutely Trusted Node N_{trust}. After the Absolute Trusted Node N_{trust} reviews its background information and grants its application, it will broadcast a $<$ADD_REQUEST$, P, pk, v, C_i, i > \sigma_{trust}$ to all nodes, where P is the sub-primary node being assigned to, pk_i is its public key, v is the current view number, C_i is the initial credit score assigned to this node, i is the serial number of this node.

- Once a node j receives this new node request, they need to verify the signature and multicast a $<$ADD_READY$, v+1, C_i, S, i, j > \sigma_j$ message to all other nodes, where S is the set of $2f+1$ new node ADD_REQUEST messages.

- When the corresponding parent node p assigned to Node i received $2f+1$ valid ADD_NODEs, it will broadcast a $<$ADD_COMMIT$, v+1, S, P, C_i, i > \sigma_p$ to all other nodes including node i that just added, where S is the set of $2f+1$ new node ADD_READY messages, P is the parent node p assigned to the added node i.

- Upon receiving the ADD_COMMIT from node P, a given node k will first check the validity of the signature and then send a $<$ADD_COMPLETE$, v+1, S, P, C_i, k > \sigma_k$ to Node i and N_{trust}, where S is the set of $2f+1$ new node ADD_COMMIT messages.

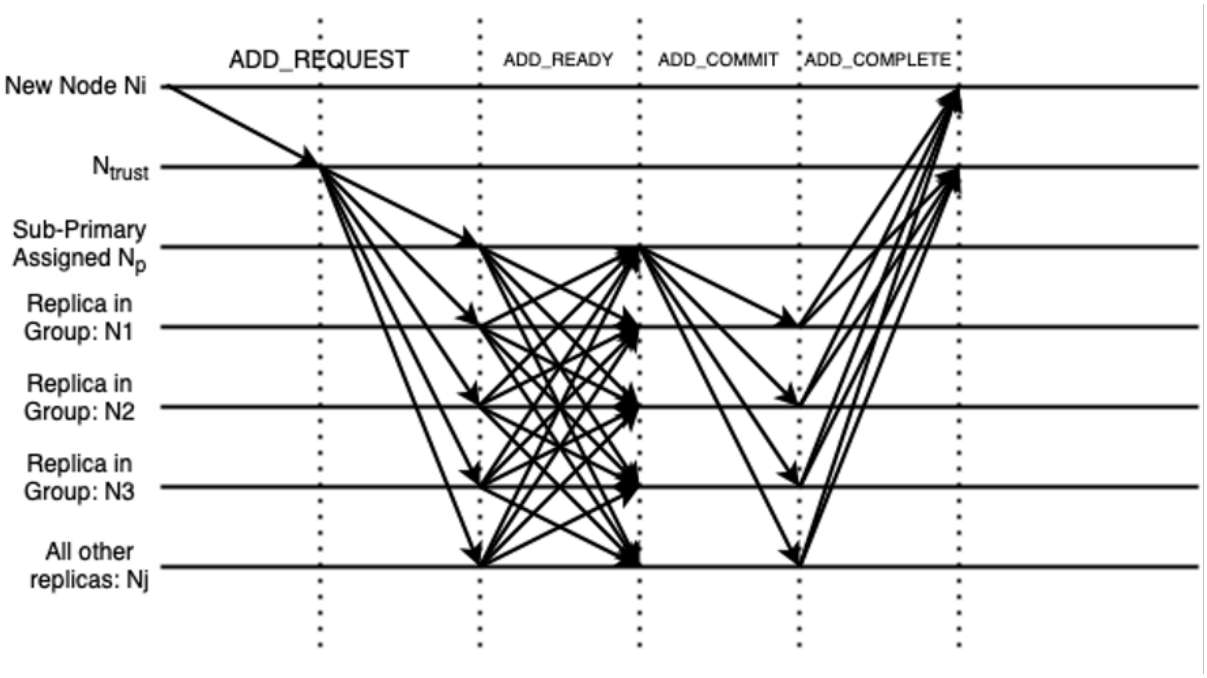

Figure 5.7: New Node Adding Protocol

- If Node i has received $f+1$ valid and consistent ADD_COMPLETE, it could now assume its NEWNODE_REQUEST has been carried out and it can start to participate in the consensus protocol with an initial credit score C_i.

For deleting a current node in the network, a DEL_REQUEST can be done in a similar manner as the previous adding case, which won't be repeatedly discussed in detail here.

Chapter 6

Performance and Discussion

In this section, we first prove the safety and liveness property of our optimized consensus algorithm. Then, we evaluate our PBFT based on m-ary hierarchical structure and compare its performance with other algorithms in terms of transaction volume, throughput and fault tolerance. Finally, to demonstrate our design of blockchain medical data system, we build a medical blockchain platform based on hyperledger fabric and react. Hence, we demonstrate the front-end and back-end logic of our system using vaccination clearance as an application scenario, which shows the unparalleled verifiability and tracability of blockchain healthcare.

6.1 Proof of Safety Property

Here, we define safety as the ability of the system to always process the same sequence of tasks and guarantee their consistency and validity.

Theorem 1. If the system has the property of safety, then the validity of the results in the blocks generated by consensus can be guaranteed, and that all non-faulty nodes should be consistent in their local blocks at a given height.
Proof. To prove that the locally stored blocks of two honest nodes are consistent in the same consensus round, we can assume that there are two such block A and block B, and if they are both recognized, then block A and block B must be the same.

Imagine that if block A is recognized, then at least $2f + 1$ nodes will agree on block A and save block A locally. If at the same time $2f+1$ nodes also agree to block B, then $2f+1$

nodes also need to agree to it at this point. Then, we need $(2f+1)+(2f+1)-(3f+1) = f+1$ nodes to accept both block A and block B, it is known that there are at most f Byzantine nodes in the whole system, so there cannot be a situation where $f + 1$ nodes approve two different blocks at the same time, because there is at least one honest node at this point.

In addition, we also need to consider the consistency problem in multi-tasking scenarios. In order to fully utilize the processing efficiency of each node and maximize the throughput, the consensus protocol allows a certain degree of concurrency, which is based on the view code and task sequence number, and each node will also prioritize the task corresponding to the earliest sequence number and discard two tasks with the same sequence number under the same view to avoid duplicate operations.

Therefore, the safety property of the consensus protocol has been satisfied.

6.2 Proof of Liveness Property

In the context of this thesis, liveness refers to the ability of the system to process requests made by a client node within a finite time threshold.

Theorem 2. A request made by a client node at any time node must receive a final result at the end, regardless of whether a Byzantine node is encountered or multiple view switches are triggered.
Proof. In order to have the liveness property, the node starts a timer during the pre-prepare phase and triggers the view switching protocol when the response time exceeds a threshold value. In addition, this response time threshold refers to the concept of exponential backoff in computer network systems, and is dynamically extended after each switchover in order to adapt to different network states. Even if a Byzantine node appears in a critical position such as a sub-primary node, due to the design of such a response timer, the system triggers the view switching protocol within a limited waiting time delay and works with the heapify method mentioned in the Node credit mechanism section to move the Byzantine node to a lower layer's non-primary node to avoid affecting the efficiency of subsequent operations. So, even in the worst case, the system must give the result in a limited time, so the liveness property of the consensus protocol has been satisfied.

6.3 Fault Tolerance

One of the very big reasons why PBFT is widely accepted is because it can handle a certain degree of Byzantine node problem. The original PBFT's fault tolerance capability is $f = (n-1)/3$, where n is the total number of the nodes, which means that the whole system is able to withstand approximately one-third of the total number of Byzantine nodes at most and generate the correct consensus result.

In our hierarchical system, if we assume that $N = m^2 = 3f_1 + 1, m = 3t + 1$ where f_1 is the maximal number of faulty node in the network system, then we have the probability for randomly select a node in the pool is $\frac{1}{3}$, denoted as

$$P_1 = Pr\{\text{a node is faulty}\} = \frac{1}{3}. \tag{6.1}$$

Now we arrange those m^2 nodes into an m-ary tree structure of height 2 as shown in Figure 6.1:

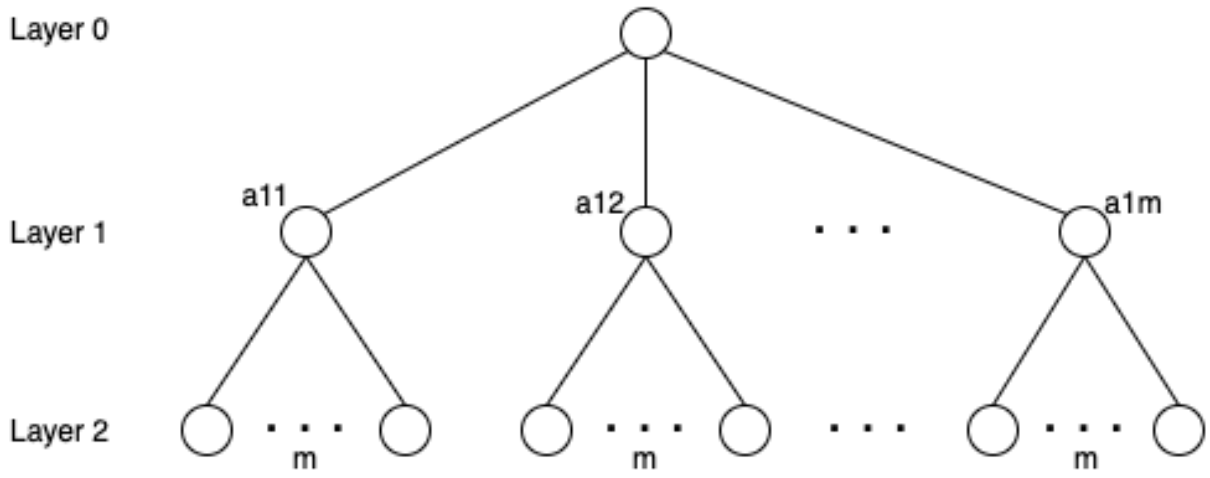

Figure 6.1: m-ary Tree Structure Example

where $a_{ij} = \begin{cases} \text{fault} & \text{if the group with } m \text{ member nodes, has more than } t \text{ fault nodes} \\ \text{non-fault} & \text{otherwise} \end{cases}$

In order to have the m-ary tree PBFT works, we need to have

$$| \{1 \leqslant j \leqslant m \mid a_{ij} \text{ fault}\} | \leqslant t \tag{6.2}$$

Let $T = \{P_1, P_2 \ldots, P_N\}$ be the set consist of all the N nodes.

$$Pr = P_j\{\text{fault}\} = \frac{1}{3}. \tag{6.3}$$

We define $P_j^* = Pr\{P_j \text{ fault} \mid \text{given that } T \text{ is partitioned into } m \text{ } m\text{-sets and PBFT will be running on each } m\text{-set }\}$

Thus $P_j^* = Pr\{P_j \text{ fault} \mid \text{given that } P_j \text{ is in a randomly selected } m\text{-set }\}$

Now, how many faults can a configuration of an m-ary tree with height r tolerate?
Theorem 3. For a m-ary tree based PBFT with $N = m^r, m = 3t + 1$, the system can tolerate $(t + 1)^r - 1$ faults.
Proof. Suppose the case $N = m^2, m = 3t + 1$. In order layer one can make the decision by running PBFT in a m-set, there are at most t groups in layer 2 which yield fault results.

In those t groups, each has $t + 1$ faults (the minimum number of the faults in the group which will produce a faulty result), the rest of the groups should be:

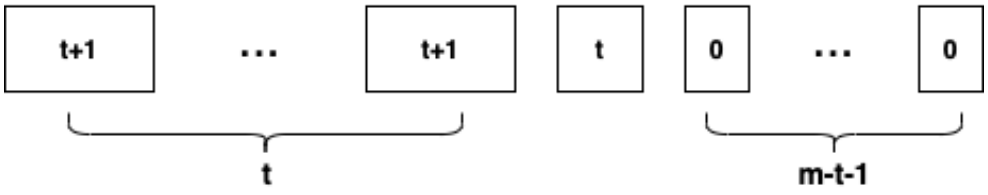

Figure 6.2: Minimum Faults Case in Groups to Provide a Faulty Result

So the system can tolerate $f_2 = t(t + 1) + t$ faults.

$$f_2 = t^2 + 2t = t^2 + 2t + 1 - 1 = (t + 1)^2 - 1$$

$$E.g. m = 4, t = 1, N = 16 = 4^2$$

$$f_2 = (1+1)^2 - 1 = 3 \text{ (It can tolerate 3 faults instead of 5)}$$

Suppose the case $N = m^3, m = 3t + 1$**.**

Let f_3 he the number of faults that can be tolerated. Then in order to have layer 2 works, it requires f_2 groups in fault decisions, maximally. Thus

$$f_3 = f_2(t+1) + t = (t+1)^3 - 1$$

By mathematical induction, for general $N = m^r$**,**

$$f_r = (t+1)^r - 1$$

From Theorem 3, the probability of a randomly selected node being a fault node is given by:

$$\rho = \frac{f_r}{N} = \frac{(t+1)^r - 1}{(3t+1)^r} \approx \frac{1}{3^r} \text{ for large t}$$

Thus, when the height of an m-ary tree increases by 1, the number of faults can be tolerated decrease by $\frac{1}{3}$. For example, for m-ary tree of height 2, its rate is $\frac{1}{9}$; for m-ary tree of height 3, its rate is $\frac{1}{27}$.

6.4 Implementations and Performance

Our platform was implemented in earlier stages using Ethereum, Ganache and Metamask for proof of concept, the blockchain environment was simulated and a vaccination Clearance management platform was built. In order to test our PBFT, we gradually migrated our focus to the hyperledger fabric, which is also based on PBFT, and tested our m-ary PBFT by replacing it with our design. In order to better demonstrate the system interaction and the logic of the back-end of the system, we built an online platform using React, which also involved Solidity, truffle, web3.js and other technologies.

To instantiate the use of blockchain in a medical scenario, we built a platform with React to manage the vaccine information of incoming students. In the case of the University of

Table 6.1: Comparsion of Blockchain-based Medical System Designs

System	consensus	computing power	Consensus leader selection	Scalability
MedRec [51]	PoW	high demand	no	low
MedicalChain [50]	PBFT	low demand	round robin	low
our work	optimized PBFT	low demand	credit based	high

Waterloo's Quest system [59], for example, vaccine verification is currently done by sending an email to students asking them to upload their vaccine certificates and then the medical team has to manually verify this information. In the future, we could add a new module to Quest's website called Health Care, which would prevent students from taking offline classes in the next semester if they have not completed their medical clearance.

6.4.1 Comparison of Blockchain-based Medical Platforms

Table 6.1 shows a comparison of Blockchain-based Medical Platforms for MedRec, MedicalChain and our design. Our platform uses an optimized PBFT consensus, which requires less computing power than the PoW-based MedRec platform and avoids wasting energy; On the other hand, when comparing with MedicalChain, which is also based on PBFT consensus, our optimization of PBFT gives us a better advantage in system performance and scalability.

6.4.2 Transaction for a Single Consensus

As we can see in Figure 6.3, compared to the traditional PBFT, our optimized PBFT has a linear increase in transaction volume as the number of nodes increases, which means that our PBFT has better performance and scalability.

6.4.3 Performance Comparison

Because there is obstacle in finding the full version of the official implementation of other BFTs, we can only simply compare the implementation of the original paper PBFT and Hotstuff, and we found that they simply cannot reach the throughput test with more than 100 nodes. Then, for further comparison and reference, we refer to some results in [76]. Based on our actual tests, we find that the original PBFT has a much lower

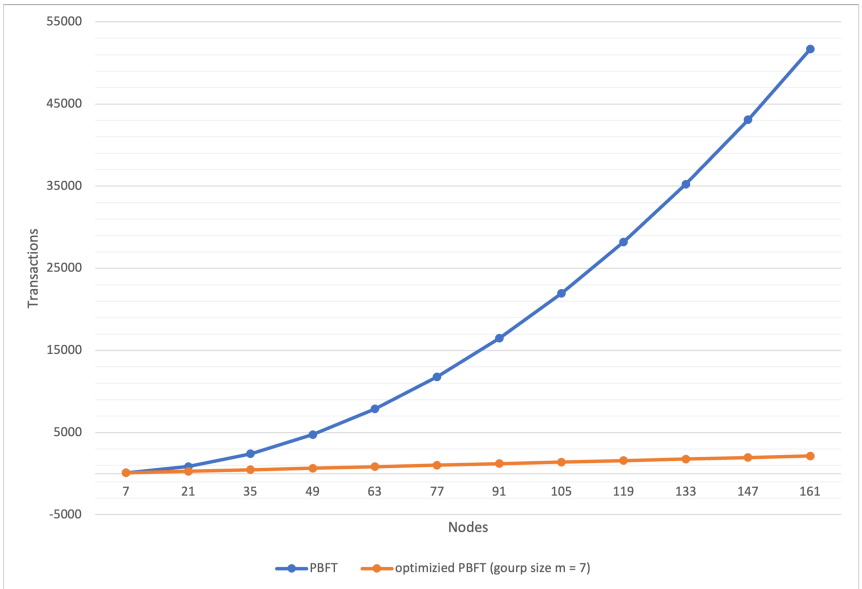

Figure 6.3: Transaction Volume for a Single Consensus

throughput than some other designs with tens of thousands of tx/s. The original PBFT can reach 1576tx/s in peak throughput in our test environment. Due to the lack of full concurrency optimization in our implementation, the cascading information transfer will cause some performance degradation, our experimental data is not as good as expected but still has a stable performance. Since we convert a significant portion of communication from broadcast between all nodes to intra-group communication, we introduced appropriate concurrency optimization in our implementation so that intra-group consensus processes between different layers can be carried out simultaneously so that upper layer does not need to wait until it receives the lower layer consensus.

6.4.4 Design of Blockchain-Based Vaccination Clearance Platform

In order to demonstrate the advantages of a blockchain healthcare system, we explored some real-life application scenarios and we found medical clearance to be a very notable example in the context of the current coronavirus pandemic. Unlike Europe and Canada where there are no very strict medical examinations and vaccine requirements for new students, the US university system mandates that every new student must pass a medical

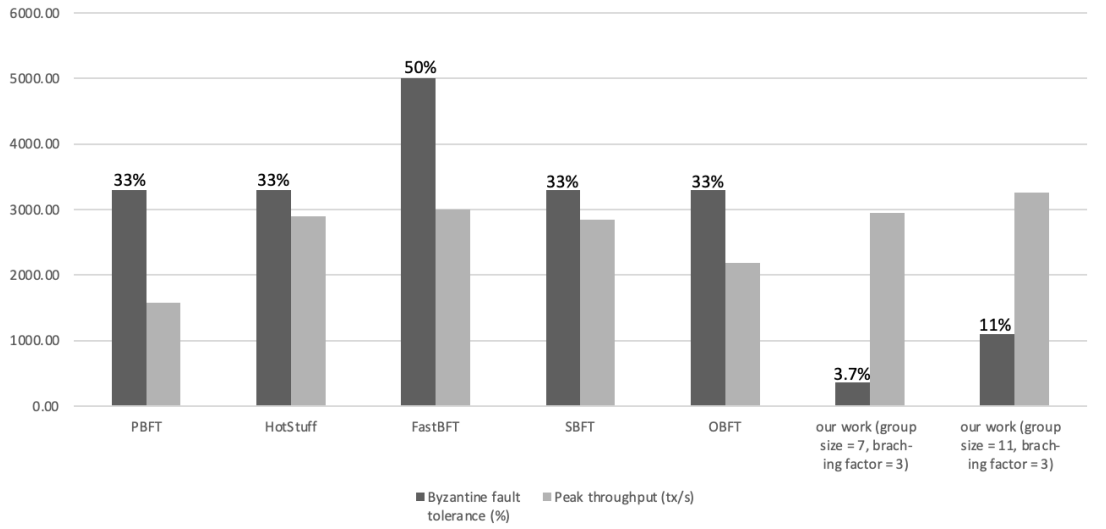

Figure 6.4: Comparison of the Consensus Protocol Performance

clearance to be able to enroll, which includes vaccines for some of the more common infectious diseases, such as coronavirus COVID-19, Infuenza, Mumps, etc., as well as some routine tests for tuberculosis.

Typically, students are required to submit proofs of a dozen vaccinations and hospital examinations one by one prior to enrollment. For some international students, the process is even more cumbersome, as they may need to return to their home country to have certificates issued by various vaccination centers and hospitals, and then go to an official agency to have them translated into English and uploaded into the system for manual verification. During the verification process, the verification process is slow and sloppy, and does not effective and accurate as expected because of the variety of sources that issue the proofs and the lack of compliance. Therefore, medical clearance is time-consuming, labor-intensive and have potential public safety hazard.

For example, for University of South California (USC), they have a list of immunization requirements for all students[58]. They have mandatory requirements for incoming students for Measles, Varicella, COVID-19 Vaccines and etc. According to the school's official website for Student Health, there will be a verification team manually checking the medical records. According to [57], in the 2021-2022 academic year, USC has a total of 49,500 enrolled students, of which 8,884 are recent new students and 23.8% are international students, where the vaccine proofs provided by the non-US students may be more difficult to manually check. If we assume that it takes 3 minutes to verify a local student's information

and 5 minutes to verify an international student's information, if every student needs to be verified for COVID-19 Vaccine proof during the semester, the total labor cost would be nearly 2771 hours, just for COVID-19 but not for all vaccines required. If we apply the blockchain-based medical clearance system, we will save these labor costs and greatly help the efficiency and accuracy of the verification.

As shown in Figure 6.5, we have designed a blockchain-based vaccination clearance platform for incoming students with reference to the University of California's admission requirements. From our design, the students need to upload the QR code received from the vaccination station to the platform in order to provide the vaccination proof, where this proof will automatically be verified with the proof stored on blockchain. This design will make the operation of vaccination and verification process more efficient and effective, and has strong security and non-falsifiability.

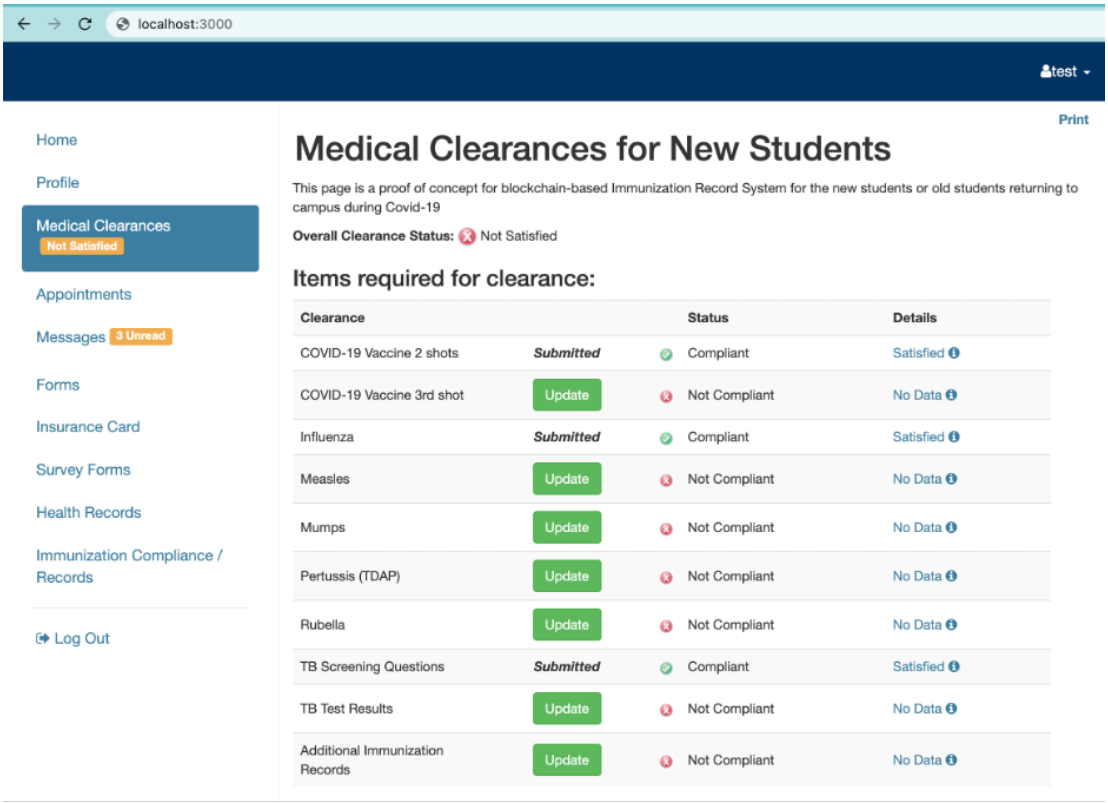

Figure 6.5: Design of the Vaccination Clearance Platform

As shown in Figure 6.6, when a patient receives a vaccination at a vaccination station,

the nurse will enter the patient's personal information and the vaccination information, where the vaccination information is connected to the blockchain of the supply chain and can contain the product name, type, batch, production date, expiration date, etc., so that if there is a problem with any batch of vaccine in the future, we can investigate and track the patients who received this batch of vaccine. Finally, it generates a Proof of Vaccination and automatically uploads the vaccination information to the blockchain.

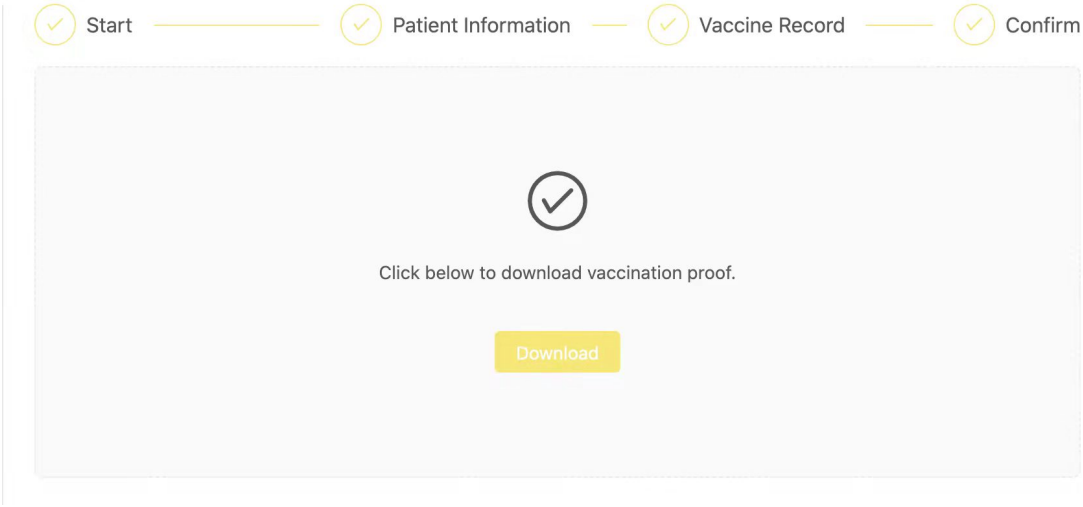

Figure 6.6: Generation of Patient Vaccination Information

In order to demonstrate the logic of the backend operation, we designed a backend module with the administrator's view, and the next interface is actually invisible to the users (medical institutions and patients). As shown in Figure 6.7, we can search all Medical Records generated by the specified vaccination station by the address of the station. In the actual design, we can design it so that only the administrator authority of the vaccination station can see the medical records of their own vaccination station, and only the higher authority administrator authority of the government or health care department can search all medical records of the station by entering the address of different vaccination stations.

We can see that all vaccination records are generated with a hash value to compare and verify the validity of the vaccination proof with the QR code uploaded by the user on the platform.

Any institutional access or verification of the vaccine record generates a record that is permanently retained on the blockchain, a process that cannot be denied or deleted. When

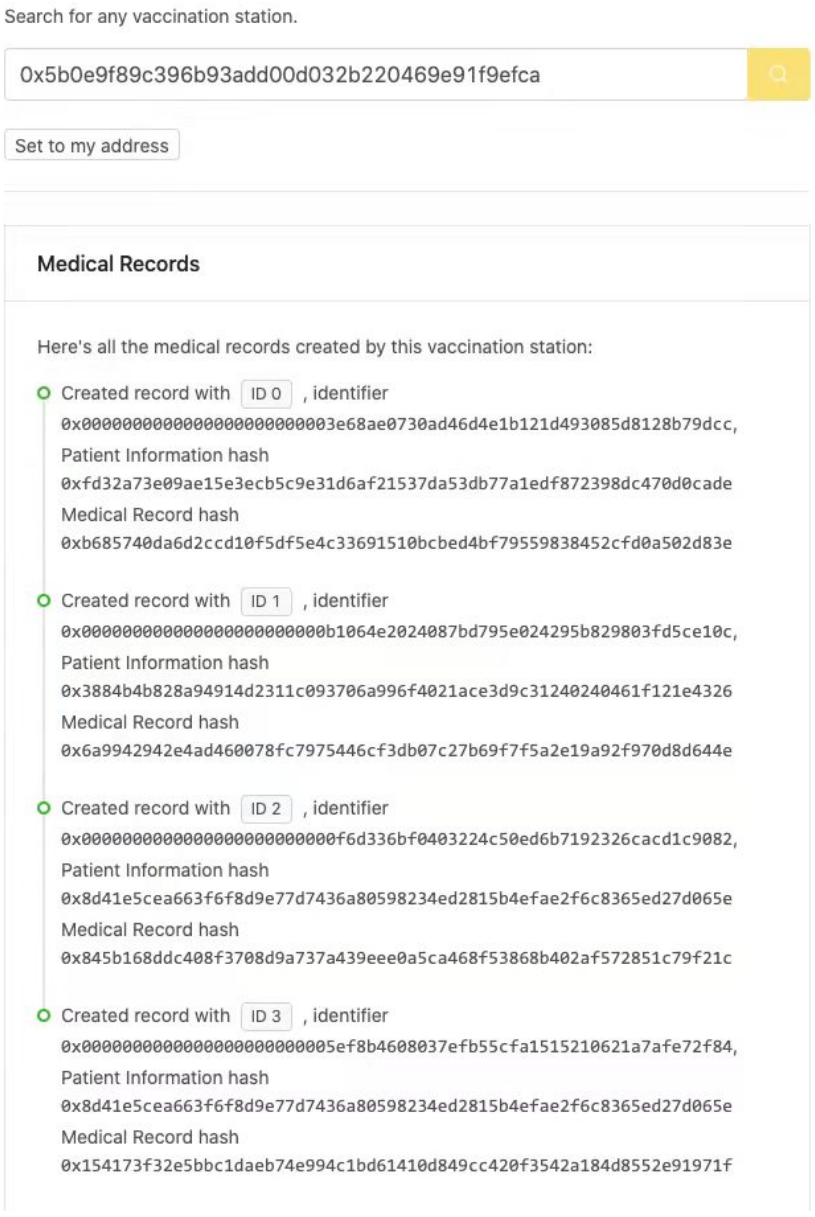

Figure 6.7: Back-end Vaccination Information Record (View of Vaccination Station Administrator)

an organization accesses a vaccine record, it can be set up so that it must first request access from the producer of the vaccine record (which we have simulated using Metamask). This ensures the security of the user's information and enables the traceability of each access record for medical information. As shown in Figure 6.8, In the backend of the system, for a particular vaccine record, we can clearly see the hash of the patient information, the hash of the vaccine information, the generator of the vaccine certificate and all reader and verifiers of this record. When a user uploads a QR code for a medical proof on medical clearance's website, it will be parsed for specific content and matched with information on the blockchain platform for verification. An example of the QR code and its parsed context is shown in Figure 6.9 and 6.10. In this example, this patient has 2 records whose validation can be done simultaneously, rather than one by one. In other words, our system is completely paitent-centric. In fact, when the user is uploading proof on the medical clearance website, only one QR Code uploading entry is needed to verify all the medical examination and vaccination information.

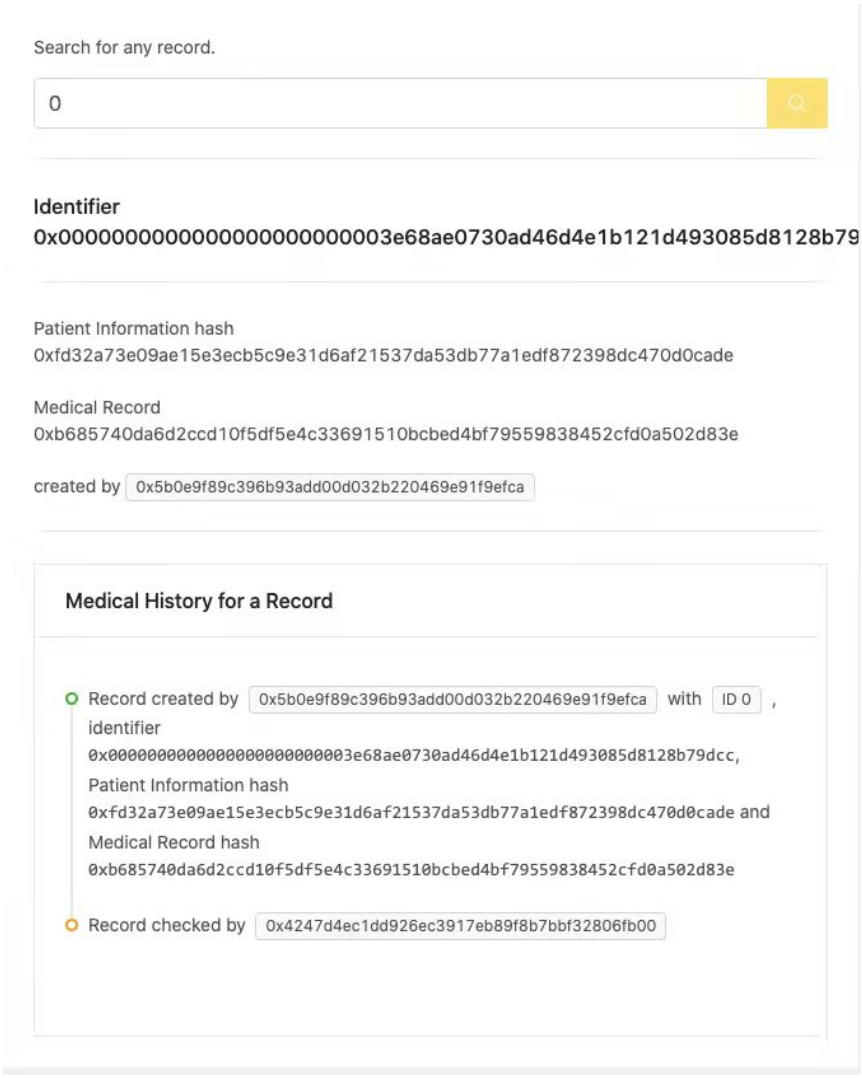

Figure 6.8: Vaccine Record System Backend

Figure 6.9: QR Code Uploaded by the User of Medical Clearance System

Patient Information hash:
0x8d41e5cea663f6f8d9e77d7436a80598234ed2815b4efae2f6c8365ed27d065e
identifier:
0x000000000000000000000000f6d336bf0403224c50ed6b7192326cacd1c9082,
Medical Record hash
0x845b168ddc408f3708d9a737a439eee0a5ca468f53868b402af572851c79f21c
identifier:
0x000000000000000000000005ef8b4608037efb55cfa1515210621a7afe72f84,
Medical Record hash
0x154173f32e5bbc1daeb74e994c1bd61410d849cc420f3542a184d8552e91971f

Figure 6.10: Parsed QR Code Context for Medical Record

Chapter 7

Conclusion and Future

In this chapter, we provide a conclusion for this thesis and list some directions for improvement of future work.

7.1 Conclusions

In this thesis, we design a medical information management system based on permissioned blockchain and build a platform using vaccination clearance during the coronavirus COVID-19 pandemic as an example to show the logic of running the front and back end of the system respectively. In terms of performance, we improved the PBFT protocol of the original hyperledger fabric, and the improved consensus protocol utilizes the hierarchical structure of the m-ary tree and incorporates a credit mechanism. In addition, we added a node authorization mechanism to the protocol for the medical scenario where nodes may change dynamically, further improving its application in the medical scenario.

Our experiments show that, after the optimization, the system has better scalability and more stable throughput, and the communication complexity is reduced from $O(n^2)$ to $O(n)$. Although we sacrifice some fault tolerance, the introduction of credit mechanism greatly reduces the probability of Byzantine nodes appearing on critical roles in real situations, further reducing their impact on the operational efficiency of the system.

7.2 Future Work

Finally, we believe that we can further improve this consensus algorithm in the future, for example, for the authorization protocol of nodes, we simply achieved the proof of concept, but did not optimize its performance. In fact, if we use the trusted node as a collector instead of treat it in parallel with other replicas, this will reduce the decentralization of the whole protocol, but also greatly reduce the complexity of the node authorization protocol. In addition, we also hope to optimize the parallel efficiency of implementation in the future, because the delay of the hierarchical structure is interlocked, and if we can maximize the parallel efficiency of different layers, there must be room for further improvement of its performance. Finally, we can continue to improve our blockchain Based Medical Clearance Platform to provide better user experience and security.

References

[1] Irani Acharjamayum, Ripon Patgiri, and Dhruwajita Devi. Blockchain: a tale of peer to peer security. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 609–617. IEEE, 2018.

[2] Mohiuddin Ahmed and Al-Sakib Khan Pathan. Blockchain: Can it be trusted? *Computer*, 53(4):31–35, 2020.

[3] Mukesh Singhal Ajay Kshemkalyani. Consensus and agreement. *Cambridge University Press*, Distributed Computing: Principles, Algorithms, and Systems.

[4] Eralp A Akkoyunlu, Kattamuri Ekanadham, and Richard V Huber. Some constraints and tradeoffs in the design of network communications. In *Proceedings of the fifth ACM symposium on Operating systems principles*, pages 67–74, 1975.

[5] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[6] Ark.io. Ark - the simplest way to blockchain. URL: https://ark.io/.

[7] Hagit Attiya and Jennifer L Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems (TOCS)*, 12(2):91–122, 1994.

[8] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *2016 2nd international conference on open and big data (OBD)*, pages 25–30. IEEE, 2016.

[9] Kenneth P Birman. Maintaining consistency in distributed systems. In *Proceedings of the 5th workshop on ACM SIGOPS European workshop: Models and paradigms for distributed systems structuring*, pages 1–6, 1992.

[10] Sebastian Burckhardt. Principles of eventual consistency. 2014.

[11] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1:22–23, 2013.

[12] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[13] Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[14] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[15] Randall D Cebul, James B Rebitzer, Lowell J Taylor, and Mark E Votruba. Organizational fragmentation and care quality in the us healthcare system. *Journal of Economic Perspectives*, 22(4):93–113, 2008.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2 edition, 2001.

[17] George Coulouris, Jean Dollimore, and Tim Kindberg. Distributed systems: Concepts and design edition 3. *System*, 2(11):15.

[18] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[19] Sisi Duan, Sean Peisert, and Karl N. Levitt. hbft: Speculative byzantine fault tolerance with minimum cost. *IEEE Transactions on Dependable and Secure Computing*, 12(1):58–70, 2015. doi:10.1109/TDSC.2014.2312331.

[20] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual international cryptology conference*, pages 139–147. Springer, 1992.

[21] Xinxin Fan, Qi Chai, Zhefeng Li, and Tian Pan. Decentralized iot data authorization with pebble tracker. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–2. IEEE, 2020.

[22] Filecoin. A decentralized storage network for humanity's most important information. URL: https://filecoin.io/.

[23] Michael J Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *International conference on fundamentals of computation theory*, pages 127–140. Springer, 1983.

[24] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[25] Erika Fry Fred Schulte. Death by 1,000 clicks: Where electronic health records went wrong, Jun 2019. URL: https://khn.org/news/death-by-a-thousand-clicks/.

[26] Adrian Gallagher. Litecoin core v0.18.1 release, Jun 2020. URL: https://blog.litecoin.org/litecoin-core-v0-18-1-release-233cabc26440.

[27] Felix C Gärtner. Byzantine failures and security: Arbitrary is not (always) random. *INFORMATIK 2003-Mit Sicherheit Informatik, Schwerpunkt" Sicherheit-Schutz und Zuverlässigkeit"*, 2003.

[28] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2):51–59, 2002.

[29] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 568–580, 2019. doi:10.1109/DSN.2019.00063.

[30] Donald E.; Patashnik Oren (1994) Graham, Ronald L.; Knuth. Properties of m-ary trees. In *Concrete Mathematics: A Foundation for Computer Science (2nd Edition)*., 1994.

[31] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990.

[32] Maurice P Herlihy and Jeannette M Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990.

[33] Tin Kam Ho. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC*, page 14–16, August 1995.

[34] IBM. What is blockchain technology? URL: https://www.ibm.com/hk-en/topics/what-is-blockchain.

[35] IoTeX. Iotex - building the connected world. URL: https://iotex.io/.

[36] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure information networks*, pages 258–272. Springer, 1999.

[37] Karim Keshavjee, John Bosomworth, John Copen, James Lai, Beste Küçükyazici, Rizwana Lilani, and Anne Marie Holbrook. Best practices in emr implementation: a systematic review. 2006.

[38] Yoojung Kim, Bongshin Lee, and Eun Kyoung Choe. Investigating data accessibility of personal health apps. *Journal of the American Medical Informatics Association*, 26(5):412–419, 2019.

[39] Cleverence Kombe, Mussa Dida, and Anael Sam. A review on healthcare information systems and consensus protocols in blockchain technology. 2018.

[40] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. 27(4), jan 2010. doi:10.1145/1658357.1658358.

[41] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. In *Concurrency: the Works of Leslie Lamport*, pages 197–201. 2019.

[42] Leslie Lamport. The part-time parliament. In *Concurrency: the Works of Leslie Lamport*, pages 277–317. 2019.

[43] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. 2019.

[44] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[45] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.

[46] Daniel Larimer. Delegated proof-of-stake (dpos). *Bitshare whitepaper*, 81:85, 2014.

[47] Tali Leibovich-Raveh, Daniel Jacob Lewis, Saja Al-Rubaiey Kadhim, and Daniel Ansari. A new method for calculating individual subitizing ranges. *Journal of Numerical Cognition*, 4(2):429–447, 2018.

[48] Jian Liu, Wenting Li, Ghassan O. Karame, and N. Asokan. Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Transactions on Computers*, 68(1):139–151, 2019. doi:10.1109/TC.2018.2860009.

[49] David. Maziereso. Stellar consensus protocol. *Open Problems in Network Security*, 2016. URL: https://www.stellar.org/papers/stellar-consensus-protocol. [Accessed:22-Oct-2021].

[50] Medicalchain. Medicalchain homepage. URL: https://medicalchain.com/en/.

[51] MedRec. Medrec homepage. URL: https://medrec.media.mit.edu/technical/.

[52] Matej Mikulic. Pharmaceutical counterfeiting incidents by world region 2020, Sep 2021. URL: https://www.statista.com/statistics/253152/share-of-worldwide-counterfeiting-incidents-in-by-region/.

[53] JP Morgan. Jp morgan report on healthcare financial challenges. URL: https://www.jpmorgan.com/commercial-banking/insights/healthcare-financial-challenges-2019.

[54] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[55] Qassim Nasir, Ilham A Qasse, Manar Abu Talib, and Ali Bou Nassif. Performance analysis of hyperledger fabric platforms. *Security and Communication Networks*, 2018, 2018.

[56] M Niranjanamurthy, BN Nithya, and S Jagannatha. Analysis of blockchain technology: pros, cons and swot. *Cluster Computing*, 22(6):14743–14757, 2019.

[57] University of South California. Facts and figures about usc. URL: https://about.usc.edu/facts/.

[58] University of South California. Usc student health - immunization requirements. URL: https://studenthealth.usc.edu/immunizations/.

[59] Unviersity of Waterloo. Quest - student information system, May 2021. URL: https://uwaterloo.ca/quest/.

[60] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[61] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[62] Rachel Pryor, Connie Atkinson, Kaila Cooper, Michelle Doll, Emily Godbout, Michael P Stevens, and Gonzalo Bearman. The electronic medical record and covid-19: Is it up to the challenge? *American journal of infection control*, 48(8):966, 2020.

[63] Mayank Raikwar, Danilo Gligoroski, and Goran Velinov. Trends in development of databases and blockchain. In *2020 Seventh International Conference on Software Defined Systems (SDS)*, pages 177–182. IEEE, 2020.

[64] Lakshmi Siva Sankar, M Sindhu, and M Sethumadhavan. Survey of consensus protocols on blockchain applications. In *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1–5. IEEE, 2017.

[65] Fabian Schuh and Daniel Larimer. Bitshares 2.0: general overview. *accessed June-2017.[Online]. Available: http://docs. bitshares. org/downloads/bitshares-general. pdf*, 2017.

[66] Ali Shoker, Jean-Paul Bahsoun, and Maysam Yabandeh. Improving independence of failures in bft. pages 227–234, 2013. doi:10.1109/NCA.2013.22.

[67] Marina Sokolova, Khaled El Emam, Sean Rose, Sadrul Chowdhury, Emilio Neri, Elizabeth Jonker, and Liam Peyton. Personal health information leak prevention in heterogeneous texts. pages 58–69, 2009.

[68] AK Soman. *Cloud-based solutions for healthcare IT*. CRC Press, 2011.

[69] Paul C Tang, Joan S Ash, David W Bates, J Marc Overhage, and Daniel Z Sands. Personal health records: definitions, benefits, and strategies for overcoming barriers to adoption. *Journal of the American Medical Informatics Association*, 13(2):121–126, 2006.

[70] Maarten Van Steen and A Tanenbaum. Distributed systems principles and paradigms. *Network*, 2:28, 2002.

[71] Vishesh Ved, Vivek Tyagi, Ankur Agarwal, and Abhijit S Pandya. Personal health record system and integration techniques with various electronic medical record systems. pages 91–94. IEEE, 2011.

[72] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. pages 112–125, Cham, 2016. Springer International Publishing.

[73] Abdul Wahab and Waqas Mehmood. Survey of consensus protocols. *arXiv preprint arXiv:1810.03357*, 2018.

[74] Lauren Wilcox, Dan Morris, Desney Tan, and Justin Gatewood. Designing patient-centric information displays for hospitals. In *proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2123–2132, 2010.

[75] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020.

[76] Chen Yang, Mingzhe Liu, Kun Wang, Feixiang Zhao, and Xin Jiang. Review on variant consensus algorithms based on pbft. pages 37–45, Singapore, 2020. Springer Singapore.

[77] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain, 2019. arXiv:1803.05069.

[78] Ian Young. Dogecoin: A brief overview & survey. *Available at SSRN 3306060*, 2018.

[79] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184, 2015. doi:10.1109/SPW.2015.27.