electronics

MDPI

Review

Byzantine Fault-Tolerant Consensus Algorithms: A Survey †

Weiyu Zhong [1,2], Ce Yang [1,2], Wei Liang [1,2], Jiahong Cai [1,2], Lin Chen [1,2], Jing Liao [1,2] and Naixue Xiong [1,2,*]

1 School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China
2 Hunan Key Laboratory for Service Computing and Novel Software Technology, Xiangtan 411201, China
* Correspondence: xiongnaixue@gmail.com; Tel.: +86-15001842414

† Some former contributions (30%) appeal in proceedings of the 9th IEEE International Conference on Edge Computing and Scalable Cloud (IEEE EdgeCom 2023), Xiangtan, China, 1–3 July 2023.

Abstract: The emergence of numerous consensus algorithms for distributed systems has resulted from the swift advancement of blockchain and its related technologies. Consensus algorithms play a key role in decentralized distributed systems, because all nodes in the system need to reach a consensus on requests or commands through consensus algorithms. In a distributed system where nodes work together to reach consensus, there may be Byzantine nodes present. The emergence of Byzantine nodes will affect the consensus of nodes in the distributed system. Therefore, tolerating Byzantine nodes in a distributed system and then reaching a consensus is an essential function of a consensus algorithm. So far, many Byzantine fault-tolerant (BFT) consensus algorithms have emerged, and there are correspondingly many methods to improve the performance of these algorithms. In order to allow researchers to have a clearer understanding of the existing methods, this paper systematically investigated and studied the research progress of the current Byzantine fault-tolerant consensus algorithm. The scope of the research ranged from the classic Byzantine consensus algorithm to some of the latest Byzantine consensus algorithms. The articles were classified according to the methods used to improve the Byzantine consensus algorithm. Through classification and centralized analysis and discussion, we achieved a clearer understanding of the development of Byzantine consensus algorithms and, at the same time, clarified the advantages and disadvantages of this type of method and the latest research progress using this method. At the end of this article, an in-depth discussion and analysis is also presented. By analyzing the impact of the use of these methods on the performance of the BFT consensus algorithm, it is proposed that future research can be improved.

Keywords: consensus algorithm; Byzantine fault tolerance; practical Byzantine fault tolerance; distributed system; blockchain

check for updates

Citation: Zhong, W.; Yang, C.; Liang, W.; Cai, J.; Chen, L.; Liao, J.; Xiong, N. Byzantine Fault-Tolerant Consensus Algorithms: A Survey. *Electronics* **2023**, *12*, 3801. https://doi.org/10.3390/electronics12183801

Academic Editor: Martin Reisslein

Received: 26 July 2023
Revised: 4 September 2023
Accepted: 6 September 2023
Published: 8 September 2023

1. Introduction

A blockchain is a linked list data structure that arranges blocks in chronological order to form a chain, which is a decentralized ledger for storing data information [1]. Blockchain, the technology that underlies Bitcoin [2], is widely used in many scenarios, such as smart contracts [3], identity verification and management [4,5], finance [6], the Internet of Things [7–9], medical information management [10], government information management [11], and supply chain management [12]. Blockchain adopts a combination of consensus algorithms, cryptography, and other technologies to achieve security, reliability, tamper identification, and decentralization. So far, blockchains are mainly divided into public chains, private chains, alliance chains, and hybrid chains. Public chains are the most widely known type of blockchain, such as Bitcoin and Ethereum. Public chains are completely open, and anyone can participate in them, conduct transactions, create smart contracts, and verify blocks. Since public chains are completely public, it is obviously inappropriate to apply them to a fixed group, so private chains and consortium chains that are suitable for smaller groups and present improved performance have emerged.

A private chain, also known as a permissioned chain, is a blockchain that restricts the rights of participants, and only certified individuals or organizations can participate in it. Due to the small number of participating nodes and high quality, a private chain has higher performance and security. An alliance chain involves a larger scale than a private chain. It is a blockchain network jointly managed by a group formed of multiple entities or organizations. Similar to private chains, participants need to be authorized to join, but consortium chains are usually composed of multiple independent entities that jointly manage and verify transactions. A hybrid chain is a kind of chain that has only appeared in recent years ad combines the characteristics of public chains and alliance chains. It has high transparency and security and can provide permission control and privacy protection for specific participants to meet the needs of different practical application scenarios [13,14].

In distributed systems, the consensus algorithm is a crucial component because it ensures consistency between different nodes, so that the entire system can run normally. The primary purpose of a consensus algorithm is to enable nodes within a distributed system to come to an agreement so that the system can handle concurrent operations and ensure data reliability, consistency, and security. In a distributed system, the state between nodes may change due to delays and network failures in the communication between nodes, leading to problems such as data inconsistency and loss errors. Not only that, but a single node can be malicious or fail. The consensus algorithm can make different nodes reach a consensus through negotiation, voting, and election, to ensure the system's stability and correctness when the above situation occurs [15–19].

To guarantee the safety of a distributed system, it is crucial for all nodes in the network to come to a consensus on the specific order in which instructions should be executed. Only when all nodes agree on this sequence can the system operate securely and without errors. Ensuring consistency is key to the system providing a secure service [20–22]. In a distributed system, it is possible to have nodes that are not functioning properly. These malfunctioning nodes can be classified into two types: Byzantine fault nodes and non-Byzantine fault nodes. A Byzantine error node refers to a node that publishes error messages to other nodes or intentionally delays the consensus process. A non-Byzantine error node means that the node crashes and cannot be used. A Byzantine fault-tolerant (BFT) [23] consensus algorithm can be used to resolve Byzantine faults. Non-Byzantine error nodes are also known as crash nodes, with corresponding crash fault-tolerant (CFT) [24] consensus algorithms to resolve crash errors.

The increasing interest in blockchain research has led to a growing enthusiasm for consensus algorithms, which are a crucial component of blockchain technology. Therefore, this paper investigates the BFT consensus algorithms proposed so far and summarizes the existing BFT consensus algorithms according to the different methods proposed to solve BFT (a consensus algorithm may belong to multiple classifications, this paper only divides them according to a certain aspect of the algorithm), so as to provide a reference for subsequent researchers engaged in research in this field. The primary accomplishments of this work are as follows:

1. We conducted an in-depth review of the origin and development of BFT consensus algorithms.
2. We divided them into several different types according to the method used by the BFT consensus algorithm, from the earliest BFT consensus algorithm to the latest BFT consensus algorithm. Then, some existing BFT consensus algorithms for each category were explained.
3. We systematically investigated and analyzed the existing BFT consensus algorithms. Through research and analysis, we discussed the limitations and deficiencies of existing methods and proposed directions for improvement in the future, providing some references for researchers who want to engage in related research.

The following sections of this work are organized as below. In Section 2, we present an introduction to the classical consensus algorithm. Secondly, in Section 3, this paper classifies and summarizes the existing BFT consensus algorithms according to different ideas that can be considered to enhance the BFT consensus algorithms' performance. Then, in Section 4, we discuss the methods for solving the BFT problem, analyze the strengths and weaknesses of the existing BFT consensus algorithms, and consider directions for further research in the future and the impact on existing technologies. Finally, the paper is concluded in Section 5.

2. Classic Consensus Algorithm

2.1. The Predecessor of the Byzantine Generals Problem

Pease, Shostak, and Lamport (PSL) proposed a method enabling independent processors or processes to achieve a mutual consensus in a distributed fault-tolerant system. The result of using this method is interactive consistency [23]. The algorithm outlined in this document, which exhibits an exponential correlation between the quantity of messages and nodes, is commonly known as the exponential information gathering (EIG) algorithm.

The premise of the EIG algorithm is that the maximum number of faulty nodes (which are nodes that cannot deliver messages or transmit error messages) in a set of n independent nodes is m or less; information is communicated (the message delivery is reliable, with no issues of channel failure or delay, and the recipient can consistently ascertain the identity of the sender); each non-faulty node has a private value, which is used to communicate with other nodes; and nodes that are not faulty always communicate in a truthful manner, while nodes that are faulty may provide false information. The algorithm was designed to achieve consensus among non-faulty nodes.

The algorithm considers two failure scenarios, single-node failure and multi-node failure. If there is only one node that fails, the protocol can achieve consensus with just two rounds of information exchange among the non-faulty nodes. In the initial stage, every node shares their individual valuation, and in the second round, they share the results obtained in the first round. However, when there are multiple node failures, the number of required rounds increases to $m + 1$. During the first round, nodes exchange their private values, and in the subsequent m rounds, they exchange the value they received in the previous round. The entire process follows an exponential pattern. It has been proven in the literature that consensus cannot be reached among non-faulty nodes when the aggregate number of nodes is fewer than $3m + 1$.

The EIG algorithm demonstrates the possibility of achieving consensus among non-faulty nodes in the presence of faulty nodes, but it is not feasible for practical implementation. Ref. [23] is the predecessor of the Byzantine problem paper, which basically introduces the content of the Byzantine problem, and the specific Byzantine generals problem will be introduced next.

2.2. The Byzantine Generals Question

In 1982, Leslie Lamport [25] introduced the Byzantine generals problem as a theoretical challenge in creating reliable computer systems. The problem arises from the need to design systems that can function effectively despite the presence of faulty components that may provide conflicting information to different parts of the system. The problem is named after the ancient Byzantine Empire, where a group of generals and their armies needed to coordinate their actions to attack an enemy city. Communication between the generals was possible only through messengers who could be compromised by traitors [26].

In the Byzantine generals problem, there may be one or more traitors among the generals who may provide false information to disrupt the battle plan. Additionally, the messengers who carry the information may also be compromised and deliver false messages. In such a scenario, it becomes critical to find a solution that ensures that the loyal generals can agree on a common battle plan despite the presence of traitors [27].

To overcome this challenge, Lamport proposed a solution that requires the loyal generals to exchange messages with each other and reach a consensus through a process called Byzantine agreement. The Byzantine agreement process involves multiple rounds of message exchange among the loyal generals to weed out false information and establish the correct decision.

The Byzantine generals problem has been a significant challenge in the field of computer science, particularly in the development of distributed systems. It has led to the development of numerous consensus protocols, including practical Byzantine fault tolerance (PBFT), which is widely used in blockchain technology to ensure that distributed nodes can agree on a common state despite the presence of faulty nodes.

In summary, the Byzantine generals problem highlights the challenge of designing reliable computer systems that can function effectively despite the presence of faulty components. The problem draws its name from the ancient Byzantine Empire, where a group of generals had to coordinate their actions to attack an enemy city. The solution to the problem involves establishing consensus among the loyal generals through a process of Byzantine agreement, which has significant applications in the development of distributed systems.

Figure 1 shows a simple Byzantine generals problem. At this time, there are only three Byzantine generals, named A, B, and C, and they want to discuss whether to attack or retreat tomorrow. For this reason, the Byzantine generals vote according to the principle of "the minority obeys the majority", that is to say, the decision is taken as long as two people agree on it. Through voting, the generals of the three parties know the ratio of attack and retreat, and then they can reach a consensus on the action. However, if a traitor appears among the three generals, the agreement among the generals will be destroyed. For example, in Figure 1, both A and B are willing to launch an attack, while C is a traitor. He will send different messages to the generals on both sides, thereby disrupting the achievement of the combat plan.

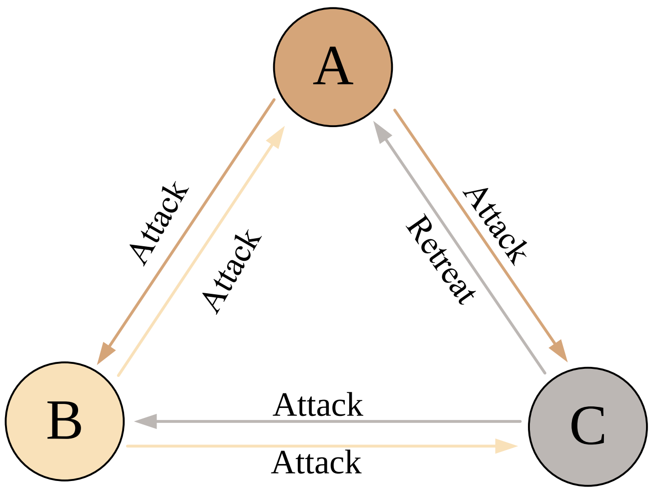

Figure 1. Simple Byzantine generals problem.

The algorithms used to solve the Byzantine generals problem are collectively known as Byzantine fault-tolerant algorithms. There were many early algorithms for solving the Byzantine general problem, such as the solutions based on oral information (OM) [25], the messages with signatures (SM) solution [25], SIFT [28], FTMP [29], and MMFCS [30]. Among them, both OM and SM were proposed by Lamport when he posed the Byzantine general problem, and the complexity of both messages is exponential.

2.3. PBFT Consensus Algorithm

Castro et al. [31] introduced the practical Byzantine fault tolerance (PBFT) consensus algorithm in 1999, which effectively addressed the BFT problem. It describes the first practical state machine replication [32] protocol. This protocol is designed to function effectively even when some nodes in a partially synchronous network exhibit Byzantine faults. Additionally, it represented an advancement over the traditional SM algorithm by achieving greater operational efficiency and reduced algorithmic complexity.

PBFT is a consensus protocol that is intended for networks with N nodes that are partially synchronous, where $N \geq 3f + 1$ and f is the number of faulty or Byzantine nodes. To prevent replay and spoofing attacks, the system uses cryptographic methods such as public key signatures, hash functions, and message authentication codes (MACs). The PBFT algorithm consists of three sub-algorithms, namely regular consensus, garbage collection, and view change. During normal operation, the system uses the regular consensus algorithm, and garbage collection is executed to remove unnecessary log messages. In the event of a failure of the primary node (also known as leader), the view change mechanism is utilized to designate a new primary node. Consensus is achieved through a series of views, each of which has a primary node and multiple replica nodes, with each replica assigned a number that is incremented in each round of consensus.

To reach a consensus on a request, PBFT typically employs a conventional mechanism consisting of three segments: pre-prepare, prepare, and commit. Figure 2 describes the procedure of reaching a consensus under normal circumstances, where there is a Byzantine node. In the pre-prepare phase, the client's request is assigned a sequence number n by the primary replica, and the resulting pre-prepared message is then distributed to all replicas. Simultaneously, the primary replica will add the message to its log. After receiving a pre-prepare message, replica i will begin the preparation phase by sending a prepare message to all other replicas via multicast. Other replicas will verify the correctness of the prepare message when they receive it and add it to their logs if it is correct. If there are at least $2f + 1$ valid prepare messages received from distinct replicas by replica i, then that particular replica will commence broadcasting the commit message. During the commit phase, the replica validates the accuracy of the commit messages received from other replicas within the system. The replica will send a request acknowledgement message to the client if and only if it receives a minimum of $2f + 1$ correct commit messages, including its own. The client can consider the system to have reached a consensus on the request as soon as it receives an $f + 1$ acknowledgement message.

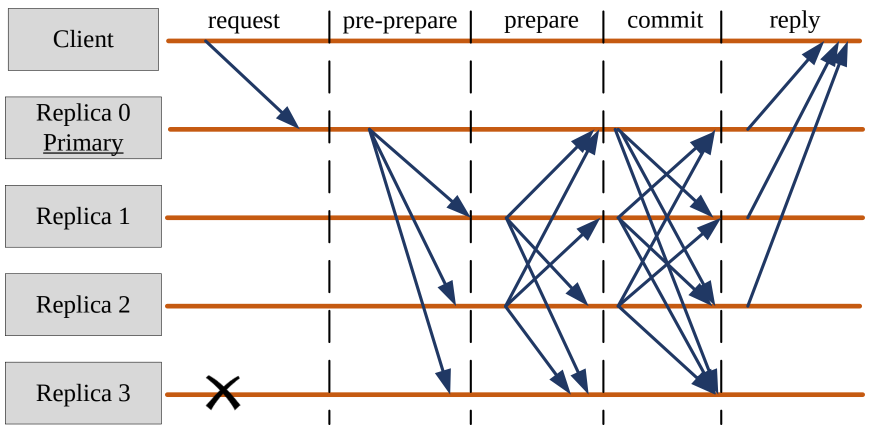

Figure 2. PBFT consensus process [31].

Stable checkpoints are used for the rubbish collection mechanism in PBFT. To ensure the security of PBFT, a replica in the system records all relevant messages in the log before a request is executed by no less than $f + 1$ correct nodes. If a request has been executed, the replica should also be able to prove this to other replicas when the view is changed. Furthermore, in cases where certain messages are absent from a replica and have been deleted by other functioning replicas, the state of the replica must be updated by retransmitting portions or all of the service state. Additionally, the replica must possess a mechanism to demonstrate the accuracy of its state. For the above factors affecting security, PBFT uses a mechanism called stability checkpoints to counteract them. Checkpoints with proofs are called stable checkpoints. After a checkpoint has been established as a stable checkpoint, the node will remove from its local message log all pre-prepare, prepare, and commit messages that pertain to requests with serial numbers that are less than or equal to n.

In addition, the node will delete any earlier checkpoints along with their corresponding checkpoint messages.

PBFT uses a view change mechanism to provide liveness, which will be triggered if the primary replica error causes the system to fail or wait indefinitely. Once the view change mechanism is triggered, the replicas in the system start voting with each other to elect a new master replica and enter a consensus process dominated by the next view [33].

Typically, in regular situations, the primary node would require a request to be sent by the client. During this process, there may be an error in the primary node. At this time, the client is required to send the request to every node, resulting in a communication overhead of N. During the pre-prepare stage, the primary node communicates with the replica node, also incurring a communication overhead of N. In both the prepare and commit stages, N nodes broadcast messages, and the communication overhead of the two phases is $2N^2$. Finally, the node needs to reply to the client, with a communication overhead of N. To sum up, the total communication overhead of PBFT should be $2N^2 + 3N$, and the communication complexity is $O(N^2)$.

3. Presentation and Analysis of Improved BFT Consensus Algorithm

With the growing popularity of blockchain research, consensus algorithms, a crucial technology for implementing blockchain, are gaining significant attention from researchers. The classical PBFT consensus algorithm requires every replica node to communicate with other nodes via peer-to-peer (P2P) communication to exchange information and achieve consensus. As the number of nodes increases, PBFT's performance significantly deteriorates, although it performs admirably with a limited number of nodes. Consequently, a significant amount of research has been devoted to enhancing the performance of the PBFT consensus algorithm. While PBFT has gained attention, many researchers have also paid attention to the methods for better solving the BFT problem. As a result, several ways to optimize the BFT consensus algorithm have emerged, including the inclusion of trusted hardware, the use of multiple leaders, quorum-based approaches, and the layering of consensus replicas.

3.1. Grouping or Hierarchical Consensus Algorithms

The PBFT consensus protocol consists of two types of replicas: primary and backup replicas, which work together to maintain the network's normal operation. However, the scalability of PBFT is constrained because increasing the number of replicas results in higher delays and reduced throughput. To address this limitation, an alternative approach could be to eliminate the primary replica and incorporate trusted components and layering techniques to enhance the system's scalability. This section focuses on improving the performance of BFT protocols by layering or grouping consensus replicas. Table 1 contains a list of the consensus algorithms referred to in this section.

Table 1. Comparison of algorithms using grouping or hierarchical consensus.

Protocol	Hierarchical or Grouping	Method	Network Model	Communication Complexity (Normal)
HiBFT [34]	Grouping	Local area network	Partial synchronization	$O(S^2)$
vBFT [35]	Grouping	Vote	Partial synchronization	$O(N^2)$
DGBFT [36]	Grouping	Confidence evaluation mechanism	Partial synchronization	$O(6N^2/S)$
QPBFT [37]	Grouping	Analytic hierarchy process	Partial synchronization	$O(N^2)$
GeoBFT [38]	Grouping	Topology-aware	Partial synchronization	$O(N^2)$

Table 1. *Cont.*

Protocol	Hierarchical or Grouping	Methods	Network Model	Communication Complexity (Normal)
FCBFT [39]	Grouping	Feature grouping and credit optimization	Partial synchronization	$O(S^2)$
PoPT [40]	Hierarchical	Specific hash function	Partial synchronization	$O(N^2)$
DHBFT [41]	Hierarchical	Credit mechanism	Partial synchronization	$O(N^2)$
DRBFT [42]	Hierarchical	Random selection algorithm	Partial synchronization	$O(N^2)$
SVBFT [43]	Hierarchical	Node status evaluation mechanism	Partial synchronization	$O(N)$
PeerCensus [44]	Hierarchical	Proof of Work	Partial synchronization	$O(N^2)$

In the table, S is the number of groups, N is the total number of nodes. The N listed in the table is the minimum value of N.

The core idea behind the design of grouping BFT consensus algorithms is to group the nodes participating in the consensus in a specific way (directly grouping or assigning different responsibilities to the nodes) and, after grouping, design the corresponding communication method again to allow all honest nodes to reach a consensus. Figure 3 illustrates the idea of a relatively simple grouping BFT consensus algorithm.

Steward [45] divides replicas into different groups, and each group is called a site. With $N > 3f + 1$ replicas in each site, f Byzantine replicas are allowed. A primary site will be elected from all sites, and the client will directly contact the primary site to send a request. When the primary site fails, there will be a corresponding processing mechanism to change the primary site. The main site's responsibility is to allocate a unique order number to the request, which is then disseminated to all other sites for further processing. Steward was designed such that site representatives communicate solely with other site representatives during communication, rather than all replicas communicating with one another. This approach improves server scalability. HiBFT [34] is a consensus protocol that is built on top of a permissioned blockchain. It employs a grouping structure to improve the scalability of the system, similar to the Steward approach. This protocol operates similarly to PBFT, but instead of physical replicas, it uses logical replicas to represent groups of replicas. HiBFT organizes several replicas in a local area network (LAN) into a group, and each group will also elect a master node. The process of voting for the master node is similar to PBFT. However, when it comes to the overall consensus process, an overall master node is elected. The consensus process starts with negotiation and communication within each group, and then the group master node is sent to communicate with all master nodes to reach a consensus. By grouping replicas together, the message complexity of the consensus process is reduced from $O(N^2)$ in PBFT to $O(S^2)$, where N represents the total number of replicas and S represents the number of replica groups involved in the consensus process. In HiBFT, the stability of a group relies on the presence of no more than f faculty members in a group of $3f + 1$ members. To ensure the tolerance of f_g unstable groups, at least $N \geq 3f_g + 1$ groups are required from a global perspective. This protocol enhances the scalability of the BFT protocol and guarantees security and liveness, assuming final synchronization. Although the method of grouping replicas can effectively reduce the computational load of replicas, it also brings challenges such as increasing communication steps and increasing latency. Therefore, many researchers have focused on how to comprehensively enhance the functionality of consensus algorithms in the context of using group-based methods [35–39,46–49].

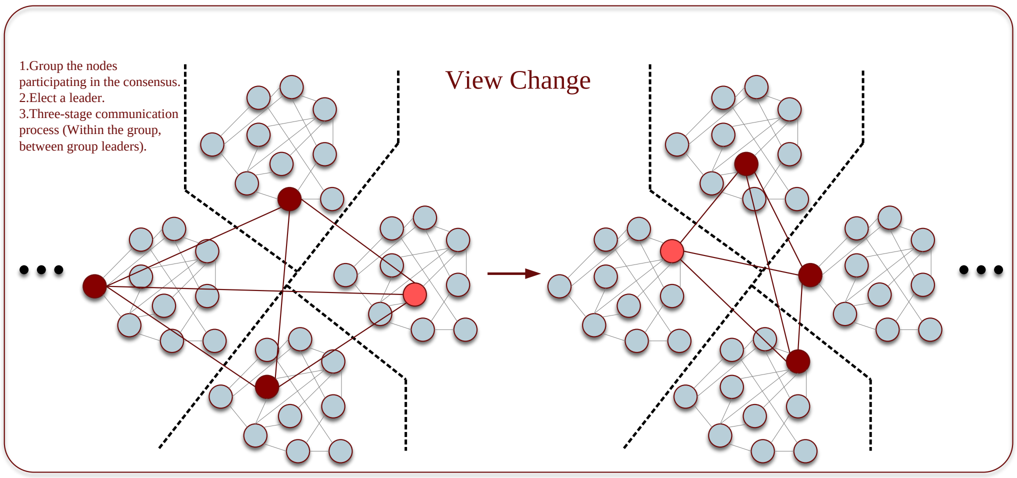
1.Group the nodes participating in the consensus.
2.Elect a leader.
3.Three-stage communication process (Within the group, between group leaders).
View Change

Figure 3. An example of a relatively simple grouping BFT consensus algorithm idea.

Grouping consensus nodes can enhance the performance of the consensus algorithm to a certain extent, such as reducing latency, increasing throughput, and enhancing scalability. However, the grouping method has extensive requirements for this type of BFT consensus algorithm, and an inappropriate grouping method will directly cause the consensus algorithm to be unusable. However, whether it is achieved by dividing nodes into different roles for management [37] or the reputation scoring mechanism proposed in [39], the effective group management of nodes can be realized. Just because grouping can bring benefits such as high throughput, low latency, improved security, and high scalability, the method of grouping consensus nodes can be effectively applied to systems that need to process a large number of transactions, large-scale alliance chains, and systems that require security. However, there are also some challenges in using groups to improve the performance of a BFT consensus algorithm, such as achieving more efficient and secure communication between different groups, further realizing decentralization, and reducing the complexity of group management. When designing a group BFT consensus algorithm, the above factors should be fully considered.

Comparable to the idea of grouping consensus copies, it is also possible to stratify the copies participating in the consensus, and assigning different tasks to different levels can also play an important role. Usually, a specific function is used or an efficient selection algorithm is designed to stratify the replicas participating in the consensus. Figure 4 presents a simple case of a hierarchical consensus algorithm. In PoPT [40], a counting node is chosen from a group of potential candidates using a particular hash function that is designed for this purpose. The role of the counting node is to maintain accurate records, and this approach also incorporates the concepts discussed earlier. The main application scenario of PoPT is JointCloud. Compared with other BFT consensus algorithms, the application scenario is relatively singular. DHBFT [41] proposes a credit-based dynamic layered Byzantine fault-tolerant consensus algorithm to solve the problems of classic PBFT nodes being unable to dynamically join and exit and reliable nodes being difficult to motivate. The protocol divides the nodes into three layers: consensus nodes, candidate nodes, and ordinary nodes. Each layer of nodes has different tasks. DHBFT also proposes specific reward and punishment schemes and dynamic upgrade and downgrade schemes to improve the system's level of security and stability. Through the improvement of PBFT, the DHBFT consensus algorithm can provide higher efficiency and reliability. Not only that, but in a more complex environment, it also has better performance, that is, it has stronger stability. However, DHBFT has higher requirements for computing power, due to the demands of the adaptive master node selection mechanism and reward and punishment mechanism designed by the algorithm. In the face of short-term disruptions and the impact of mobility disruptions, DHBFT takes a long time to recover, because the layering and other sub-algorithms need to be adjusted. The evaluation method for node credibility is enhanced in CRPBFT [50], which also introduces a comprehensive reputation model for quantifying the credibility of nodes in the network. The algorithm also devises a rotation mechanism for clearing malicious nodes in the network. Through mutual cooperation between the two, the robustness of the network can be improved and a higher degree of decentralization can be maintained. DRBFT [42] proposed an algorithm called RS, which utilizes random selection to choose a certain number of replicas from all available replicas to participate in the consensus process. This approach reduces the number of replicas involved in the consensus process, improves consensus speed, and ensures both security and liveness. Ref. [51] applied multiple signatures to a hierarchical Byzantine consensus protocol to provide stronger security and privacy for the BFT consensus protocol. This protocol innovatively combined the Schnorr ring signature [52,53] and MuSig [54] signature scheme, resulting in three levels of privacy protection during the consensus process. DRBFT was also an improved consensus algorithm based on PBFT, and related work includes [43,55–57].

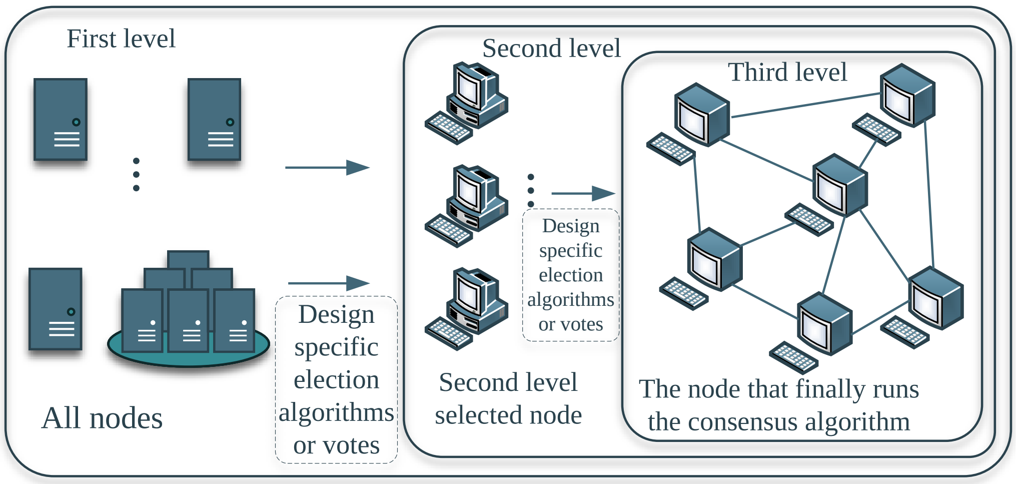
First level
All nodes
Design specific election algorithms or votes
Second level
Design specific election algorithms or votes
Second level selected node
Third level
The node that finally runs the consensus algorithm

Figure 4. An example of a relatively simple hierarchical BFT consensus algorithm idea. All nodes are selected as different levels, and different levels may have different tasks. After the levels are divided, different levels cooperate with each other to reach a final consensus.

The election committee refers to selecting a certain number of replicas from all replicas to form a committee through a specific method, and the committee utilizes the BFT consensus protocol to achieve consensus (PBFT is more commonly used). The consensus algorithm of the election committee is usually a hybrid consensus protocol that combines the BFT consensus algorithm with the proof-of-work algorithm (POW) or proof-of-stake algorithm (POS). In the realm of public blockchains, both proof-of-work (PoW) and proof-of-stake (PoS) consensus algorithms are widely acknowledged, but they have drawbacks in terms of operational efficiency due to their heavy computational requirements, and they can be susceptible to node monopolies from those with greater computational resources. Nonetheless, POW and POS continue to offer notable benefits, including strong security and resistance to tampering. Therefore, some researchers have used the advantages of POW and POS by combining them with BFT consensus protocols. There are many related protocols [58–61]. The PeerCensus [44] blockchain consensus protocol uses a mixture of PBFT and POW. To achieve consensus on a transaction block, the POW algorithm first selects a group of nodes to form a committee, which then utilizes the PBFT protocol. The committee members in the agreement are not always the same, as there will be new members joining and old members leaving. However, there is no clear provision in PeerCensus for how to reconfigure committee members. Ref. [62] proposed a fair selection protocol for reconfiguring committees. The fair selection protocol mainly includes the mining process and the confirmation of the new node list. Ref. [63] used delegated proof of stake (DPoS) to elect witness nodes, which were used to generate blocks, and then ran the PBFT consensus protocol on the alternative witness nodes to verify the generated blocks. The election committee can also be divided into electing a single committee and electing multiple committees. The parallel operation of multiple committees can improve the scalability of the network [60,61,64,65].

3.2. Multi-Leader Consensus Algorithm

The leader-based BFT protocol relies on a single leader node to order the requests in the network. However, this approach presents a potential weakness, as any mistakes made by the leader can have significant impacts on the entire protocol. Therefore, it is crucial for the protocol to detect and recover from any errors made by the leader in real time. Additionally, relying on a single node to order requests limits the protocol's security and robustness, prompting researchers to explore alternative solutions. For example, Boran and Schiper proposed a theoretical protocol that eliminates the need for a primary replica [66], but this approach lacks practical significance. Another solution proposed by some researchers is to use multiple primary replicas to order requests [67–70]. However, this approach still requires replicas to verify the correctness of received messages, resulting in longer wait times for the system. Overall, researchers are actively exploring ways to address these challenges and enhance the security and efficiency of BFT protocols. Table 2 exemplifies some multi-leader BFT consensus protocols that will be introduced in this section.

RBFT [71] proposed a plan to execute multiple instances in parallel in a BFT protocol. Each instance has a primary replica, that is, multiple primary replicas are used to support the correct operation of the protocol. The protocol requires $N \geq 3f + 1$ nodes to tolerate f Byzantine error nodes, which can effectively ensure robustness. The RBFT protocol differentiates between a main instance and a backup instance, with only the requests that are ordered by the main instance being eligible for execution by the node. The backup instance's role is to sort the requests and oversee the primary instance's work. In the event of an error, the primary node is replaced by other replicas, and if the primary instance is considered malicious, a corresponding change mechanism is in place to replace it. RBFT makes reasonable use of multiple leaders, which not only enhances the robustness of the consensus algorithm, but also strengthens its tolerance of errors. The performance of the algorithm does not drop too much when node failure occurs. However, in order to support the advantages brought by the RBFT consensus algorithm, the supervision mechanism (used to detect whether the master node is wrong) needs to consume a large computational and communication performance overhead.

To increase the efficiency of wide-area networks (WANs) that have a high density of nodes, Mir-BFT (Mir) proposed the utilization of multiple leaders for parallel and independent batch request processing [72]. While this approach offers benefits, there is a potential issue of performance attacks resulting from duplicated requests. To address this problem, Mir partitions the request hash space among several copies. Mir is a BFT protocol that enhances the PBFT by placing emphasis on both security and liveness, while also boosting network throughput. In order to improve security and eliminate the impact of mobile interruption, Mir adopts a client-side signature and verification mechanism, which increases the latency and is also one of the limitations of the algorithm. However, in order to reduce the impact of this limitation, Mir uses a signature verification sharding mechanism. Mir uses a broadcast communication method [73], and it requires a lot of communication complexity to run under normal circumstances. Furthermore, when using multiple leaders, many other factors need to be considered, so the complexity of the algorithm is relatively high. Similarly, BigBFT [74] also uses multiple leaders to overcome the communication limitations of single-leader BFT protocols, but it minimizes communication message complexity to $O(N)$. The message complexity of BigBFT remains unchanged even if there is overlap between the proposal voting and coordination phases. BigBFT leverages two communication rounds to sort blocks and achieves block consensus by pipelining blocks of the same round or through different rounds. FnF-BFT [75] is a state machine replication protocol applicable to partially synchronous models. It permits all nodes to act as leaders, allowing for parallel request processing and even the distribution of the load across the network. FnF-BFT outperforms HotStuff [76] in terms of throughput, and its scalability improves with an increase in replicas. However, its performance in terms of latency is subpar.

Table 2. Comparison of Multi-Leader Consensus Algorithms.

Protocol	N	Communication Complexity (Normal)	Mechanisms Related to Multi-Leader Election	Network Model
RBFT [71]	$3f+1$	$O(N^2)$	The monitoring mechanism and the protocol instance change mechanism	Partial synchronization
Mir-BFT [72]	$3f+1$	$O(N^2)$	Dynamically adjusting the number of leaders	Partial synchronization
BigBFT [74]	$3f+1$	$O(N)$	Coordinator	Partial synchronization
FnF-BFT [75]	$3f+1$	$O(N)$	Hash space division and primary rotation	Partial synchronization

N in the table is the total number of nodes, and f is the number of Byzantine nodes. The N listed in the table is the minimum value of N.

Using multiple leaders can not only effectively reduce the complexity of consensus, but also improve the throughput of the system. When the scale of nodes in the network is large, the above advantages are obvious. However, the following challenges also exist when using multiple leaders: selecting multiple different leaders, negotiation between different leaders, ensuring safety and integrity, and reducing latency. It is believed that only by fully considering the above factors can a high-performance multi-leader BFT consensus algorithm be designed.

3.3. Consensus Algorithm Based on Speculation and Optimism

Among the many methods to improve the performance of BFT consensus algorithms, one optimization approach involves the replica treating the request made by the client with a positive attitude and responding to the request in a timely manner. This is the speculation-based consensus algorithm approach that will be explained in this section. Table 3 exemplifies some speculation-based consensus algorithms that will be introduced in this section.

Table 3. Comparison of consensus algorithms based on speculation and optimism.

Protocol	N	Communication Complexity (Normal)	Optimistic Execution	Network Model
Zyzzyva [77]	$3f+1$	$O(N)$	Speculation	Asynchronous
AZyzzyva [78]	$3f+1$	$O(N)$	Speculation and lightweight	Asynchronous
MinZyzzyva [79]	$2f+1$	$O(N)$	Unique sequential identifier generator and speculation	Asynchronous
hBFT [80]	$3f+1$	$O(N^2)$	Three communication steps and speculation	Partial synchronization
SAZyzzva [81]	$3f+1$	$O(N)$	Monotonic counter and speculation	Partial synchronization
DBFT [82]	$3f+1$	$O(N^2)$	Double-response mechanism and speculation	Asynchronous

N in the table is the total number of nodes, and f is the number of Byzantine nodes. The N listed in the table is the minimum value of N.

Zyzzyva [77] proposed using speculation to reduce the complexity of designing Byzantine fault-tolerant state machine replication. In Zyzzyva, the primary server still needs to first sort the requests made by the client, but due to the existence of the speculative mechanism, the replica can speculatively execute the requests made by the client without waiting for the entire protocol to be completed and for an agreement to be reached on the order of the requests. This operation may lead to temporary inconsistencies in the state between replicas. Therefore, Zyzzyva was designed to let the client detect the inconsistency, that is, the submission process is left to the client to complete. The consensus process of Zyzzyva is shown in Figure 5 (there is a faulty node at this time). The client judges whether a speculative reply is reliable based on whether the reply and its corresponding history are stable. If there is no reliable answer, the client waits until the system converges to a stable answer and history. Zyzzyva uses speculative execution to effectively reduce latency and increase throughput. However, the speculative mechanism will also bring some security risks and waste computing resources.

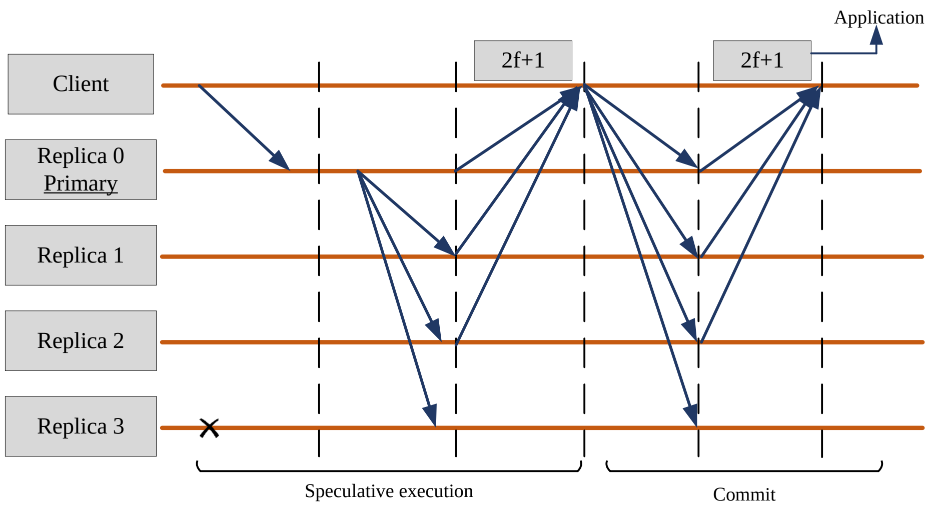

Figure 5. Zyzzyva's consensus process in the presence of a faulty node [77].

A new consensus protocol design method called Abstract was presented in [78], and two consensus algorithms were proposed based on this method: AZyzzyva and Aliph. Abstract's approach treats a BFT protocol as an abstract composition of instances, each of which can be individually designed to accomplish a task. AZyzzyva was a further improvement on Zyzzyva. The protocol uses different sub-protocols to complete tasks in different situations (AZyzzyva proposed two sub-protocols, ZLight and Backup, for the two different situations of asynchronous/fault and non-asynchronous/fault). In the absence of failures or asynchrony, ZLight mimics the operation of the Zyzzyva protocol to reach consensus. ZLight includes a checkpoint sub-protocol, which will be triggered when the number of messages reaches 128 to truncate historical messages. Backup is implemented based on PBFT. SAZyzz [81] adopted a tree-based communication model based on AZyzzyva, which further improved the fault scalability of AZyzzyva.

MinZyzzyva [79] added a trusted component service—the unique sequential identifier generator (USIG) based on Zyzzyva. USIG exists in the local service of each server, and it can assign identifiers to messages and sign messages, thereby improving the security of the protocol. By using the USIG trusted service component, in the presence of f Byzantine error nodes, in order to guarantee that the protocol provides safe and reliable services, MinZyzzyva needs the number of summary points to be at least $2f+1$ instead of $3f+1$ from Zyzzyva. However, it is difficult and unnecessary to equip each replica with a trusted component in practice. SACZyzzyva proposed that it is only necessary for the primary replica to have an active monotonic counter for a period of time [83]. Also, the protocol overcomes the performance penalty of the need for non-speculative fallbacks.

The utilization of a speculation mechanism enables hBFT [80] to decrease the cost associated with the BFT consensus protocol while still upholding optimal resilience. Essentially, hBFT is a hybrid Byzantine consensus protocol. To find a solution for state inconsistency between replicas brought about by speculative execution, hBFT designed a three-phase checkpoint sub-protocol similar to PBFT. Since hBFT delivers some key tasks to the client (such as switching between sub-protocols), in order to ensure the security of the system, hBFT adds the client suspicion mechanism, which can detect whether the client is correct in real time and take corresponding measures to cope with error situations. When using the speculation method, hBFT designs more mechanisms to improve the security and fault tolerance of the consensus algorithm. Despite the utilization of speculation in hBFT to further enhance the performance of the consensus algorithm, the algorithm's resistance to attacks remains sub-optimal, and its performance is constrained by the requirement for frequent exchanges between nodes, which can be resource-intensive.

In order to further prevent the threat of speculative execution, DBFT [82] proposed a double-response mechanism: responding deterministically to the client after the speculative execution phase and after the commit phase, respectively. By using the double-response mechanism, DBFT does not need to add additional stages to deal with the problem of inconsistency between replicas caused by speculative execution. Similar to other methods using speculation, DBFT also delivers some tasks to the client. However, DBFT delivers simple and lightweight tasks such as sending requests and receiving responses to the client, which can reduce the impact of client failures on the operation of the entire protocol. The protocol also designed a self-conflict checking mechanism and a rotating primary mechanism to deal with failures related to view changes and primary nodes, respectively. DBFT has excellent performance, including a high throughput, low latency, and better scalability. Moreover, its series of designs have raised speculation-based consensus algorithms to a higher level.

The core idea of the speculation-based consensus algorithms is to allow nodes to perform calculations and verifications in advance without fully reaching a consensus, thereby improving the performance of the entire network. There are several distinct benefits to using this approach. First, the speculation-based approach improves the processing performance of the entire network by allowing nodes to execute transactions ahead of time during the consensus process. This means that the network can process more transactions, reducing transaction confirmation times. Secondly, this algorithm has a high fault tolerance, because even if some nodes make mistakes during the consensus process, other nodes can still continue to process transactions. This helps maintain network stability and availability. Furthermore, since nodes can execute transactions in advance during the consensus process, the consensus algorithm based on speculation can reduce network latency. This means transactions can be confirmed faster, improving the user experience.

Despite these many benefits, there are also downsides to using a speculation-based approach. Since speculation-based consensus algorithms allow nodes to execute transactions in advance without reaching a consensus, they may lead to problems such as insufficient security, the consumption of more computing resources, and consensus uncertainty. This may limit the scalability of the network to some extent. Therefore, the consensus algorithms in this area remain to be further studied.

3.4. Consensus Algorithms Based on Trusted Hardware

The use of hardware can speed up the calculation process, which naturally reduces the time required to reach a consensus. Hence, incorporating reliable consensus protocols that leverage hardware assistance can enhance system throughput and enable the handling of a greater number of requests simultaneously. Trusted hardware can provide a certain degree of physical security isolation, which can better protect against malicious attacks and Byzantine faults, thereby improving the security of the BFT consensus protocol. The addition of hardware to the consensus protocol can enhance its efficiency, throughput, and security, as can be observed. Table 4 summarizes some trusted-hardware-based consensus protocols mentioned in this section.

Table 4. Comparison of trusted-hardware-based consensus algorithms.

Protocol	N	Communication Complexity (Normal)	Trusted Hardware Name	Network Model
CNV [84]	$2f+1$	$O(N^2)$	Trusted multicast ordering service (TMO)	Asynchronous
CheapBFT [85]	$2f+1$	$O(N^2)$	Counter assignment service in hardware (CASH)	Partial synchronization
ByzID [86]	$2f+1$	$O(N)$	Specification-based IDS	Asynchronous
FastBFT [87]	$2f+1$	$O(N)$	Trusted execution environment (TEE)	Partial synchronization
Avocado [88]	$2f+1$	$O(N^2)$	Trusted execution environment (TEE)	Asynchronous
Damysus [89]	$2f+1$	$O(N)$	Checker and accumulator services	Partial synchronization
SplitBFT [90]	$3f+1$	$O(N^2)$	Trusted execution environment (TEE)	Partial synchronization

N in the table is the total number of nodes, and f is the number of Byzantine nodes. The N listed in the table is the minimum value of N.

Correia, Neves, and Verissimo (CNV) leveraged a new trusted timely computing base (TTCB) service, the trusted multicast ordering service (TMO), to develop an atomic multicast protocol that could withstand the presence of f Byzantine nodes using only $N \geq 2f+1$ replicas [84]. The function of the TMO is to allocate a sequential number to the message while also verifying its authenticity, after which the verified message can be transmitted through the communication channel. In this consensus protocol, malicious replicas have no control over the channel, they cannot be ambiguous, and thus they cannot act arbitrarily [91]. CheapBFT is a Byzantine fault-tolerant (BFT) system that utilizes a reliable sub-system, known as CASH, which is based on FPGA technology, to verify messages and achieve high efficiency in resource usage [85]. The role of CASH is to verify and authenticate messages. Compared with traditional BFT consensus algorithms, the security is improved. CheapBFT utilizes a composite consensus protocol comprising three sub-protocols: CheapTiny, CheapSwitch, and MinBFT. CheapTiny is responsible for the normal case protocol, CheapSwitch handles the conversion protocol, and MinBFT operates as the fallback protocol [79]. The number of replicas required for different sub-protocols is different. In typical scenarios, having $N \geq f+1$ replicas is sufficient to withstand f malicious replicas, but as a general rule, the system requires $N \geq 2f+1$ replicas to tolerate f malicious replicas.

ByzID [86] uses a lightweight intrusion detection system (IDS) based on specifications to develop a fault detection component that continuously monitors the behavior of replicas in real time to detect any instances of misbehavior among them. Through this trusted component, ByzID has stronger robustness against attacks, and its efficiency is also improved to a certain extent. Like other consensus protocols that rely on trusted components, the ByzID consensus protocol also requires $N \geq 2f+1$ replicas to tolerate f Byzantine fault replicas. However, ByzID is able to maintain security and functionality even if the IDS crashes, with the correct functioning of the protocol primarily relying on the replicas.

FastBFT utilized Intel SGX as a hardware-based solution to develop a novel message aggregation technique that leverages trusted execution environments (TEEs) and lightweight secret sharing [87,92,93]. The unique aspect of this protocol is that it achieves message aggregation without the need for any public key operations. FastBFT not only improves scalability, efficiency, and throughput but also reduces message complexity from $O(N^2)$ to $O(N)$. Avocado was designed to offer robust security, fault tolerance, consistency, and high performance in untrusted cloud environments that utilize TEE technology [88]. The replication protocol in this system was built on the basis of the multi-writer ABD [94] protocol. Compared with BFT, the replication protocol in Avocado has stronger security and better performance. SplitBFT [90] combines a trusted execution environment (such as SGX) with a PBFT consensus protocol to improve the security and confidentiality of

BFT systems. Its core idea is to divide the PBFT protocol into different parts, with each part being an independent area protected by trusted execution. Although the performance of SplitBFT has not shown substantial improvements compared to other BFT consensus algorithms, the security and integrity have been greatly improved. The enhanced security and integrity make SplitBFT suitable for cloud service providers, as this is exactly what they need. In the future, further improvements can be made in the fine-grained division of SplitBFT to provide better performance.

HotStuff-M and VABA-M [95] proposed using a trusted log as the trusted component to enhance the fault tolerance of the consensus algorithm. Each protocol stage is served by two trusted logs, one for protocol uptime and one for trace view. Also, to improve the performance of thin consensus protocols like HotStuff, Damysus [89] uses checkers and accumulators as trusted components. The two components perform some key operations in the protocol, where the job of the checker is to maintain the state of the replica in the corresponding view, and the accumulator is responsible for the related operations of the view change. The protocol improves throughput relative to HotStuff and achieves performance improvements in terms of resiliency, latency, and communication complexity. Damysus only needs $N \geq 2f + 1$ replicas to tolerate f Byzantine fault replicas.

3.5. Other Consensus Algorithms

If the number of Byzantine faulty nodes grows, the performance of the consensus algorithm will correspondingly deteriorate, potentially resulting in network failure and rendering it inoperable. The majority of Byzantine fault-tolerant consensus algorithms define a threshold for the maximum number of nodes that can be faulty while still maintaining system functionality. For instance, they may require that the proportion of Byzantine nodes be no more than $1/2$, $1/3$, $1/5$, or $1/7$ among the complete set of nodes in the network. Hence, there have been significant research efforts aimed at enhancing consensus algorithms' throughput and expanding the count of Byzantine fault-tolerant nodes that can be accommodated. Inspired by the quorum system, some consensus algorithms have also been derived. The Q/U protocol [96] offers a fault-scalable Byzantine fault-tolerant service by requiring $N \geq 5f + 1$ servers (where f is a Byzantine fault server) to tolerate up to f Byzantine fault servers. This protocol integrates innovative techniques such as version control, arbitration, optimism, and cryptography to achieve scalable performance and fault tolerance. Although the Q/U protocol maintains good performance when there is no contention between replicas, a considerable quantity of replicas is required to complete tasks under contention. In contrast, HQ uses a lightweight quorum-based protocol to resolve non-contentious situations and BFT consensus to resolve contentious ones [97]. Compared to the Q/U protocol, HQ requires only $N \geq 3f + 1$ replicas to ensure normal system operation with up to f Byzantine fault replicas. By utilizing a hybrid approach of a quorum-based protocol and BFT, HQ improves the efficiency of solving the scalability problem caused by failed nodes. Table 5 lists the BFT consensus algorithms that will be introduced in this section.

HotStuff [76] is a consensus algorithm that provides Byzantine fault tolerance in a distributed system. As mentioned in [76], if the complete sum of replicas is $3f + 1$ or more, the system is capable of admitting a limit of f Byzantine faults. The algorithm achieves this by modifying the broadcast vote mechanism in the PBFT algorithm, which is commonly utilized for achieving consensus in conventional distributed systems. In HotStuff, the leader replica is responsible for collecting votes from all other replicas before broadcasting the result. This modification reduces the algorithm's complexity from $O(N^2)$ to $O(N)$, making it more efficient and scalable. Figure 6 illustrates the consensus process, which depicts a typical scenario under normal circumstances. Furthermore, Chained HotStuff optimizes the algorithm by introducing the concept of a pipeline and using the genericQC approach in multiple stages. By adopting this approach, many comparable tasks are streamlined, and the consensus process is enhanced in terms of efficiency. By dividing the consensus process into multiple stages, each with its own set of replicas, the algorithm achieves a higher level of parallelism and can handle larger-scale systems. Overall, HotStuff and its Chained variant provide an efficient and scalable solution for achieving consensus in a distributed system, even with the presence of Byzantine faults. The modifications made to the PBFT algorithm mean that it is more pragmatic and suitable in practical applications. However, HotStuff has relatively high expectations for the master node, so it replaces the master node in each round in order to prevent the frequent failure of the master node, leading to a lot of time spent on replacing the master node.

Table 5. Comparison of other BFT consensus algorithms.

Protocol	N	Communication Complexity (Normal)	Design Goals	Network Model
Q/U [96]	$5f+1$	$O(N^2)$	Better throughput and fault scalability	Asynchronous
HQ [97]	$3f+1$	$O(N)$	Hybrid to handle different fault situations	Asynchronous
HotStuff [76]	$3f+1$	$O(N)$	Provide a simpler and more practical copy service	Partial synchronization
Dynamic PBFT [98]	$3f+1$	$O(N)$	Replicas and nodes dynamically join or exit the consensus network	Partial synchronization
WRBFT [99]	$3f+1$	$O(N)$	Combines node consensus workload and verifiable random function	Partial synchronization
Flexible BFT [100]	$3f+1$	$O(N^2)$	Greater resilience and diversity	Partial synchronization
Tendermint [101]	$3f+1$	$O(N^2)$	To simplify the view change mechanism and timeouts in PBFT	Partial synchronization
HoneyBadgerBFT [102]	$3f+1$	$O(N^3)$	Cases where BFT consensus protocols that address partially synchronous network assumptions rely heavily on network assumptions	Asynchronous
DispersedLedger [103]	$3f+1$	$O(N^3)$	Address the issue of block retrieval traffic slowing down scatter traffic, causing system throughput to decrease	Asynchronous
Dumbo [104]	$3f+1$	$O(N^3)$	Reduce latency and increase the throughput of HoneyBadgerBFT	Asynchronous
Dumbo-NG [105]	$3f+1$	$O(N^3)$	Resolve the latency–throughput tension	Asynchronous
BeauForT [106]	$3f+1$	$O(N^2)$	Consensus in a Byzantine fault-tolerant manner that is lightweight, leaderless, and centered on the client	Partial synchronization

N in the table is the total number of nodes, and f is the number of Byzantine nodes. The N listed in the table is the minimum value of N. The design goal column may include the innovative mechanism of the BFT consensus algorithm.

Dynamic PBFT [98] builds upon the PBFT protocol and enhances its functionality by enabling replicas to join and leave the consensus network dynamically, thus breaking the original PBFT's complete closure. This modification not only allows for the removal of faulty replicas but also introduces a new notion of "participation" to ensure that the nodes are sufficiently active, thereby improving the system's robustness and security. In order to ensure the dynamic joining and exiting of nodes, each copy in Dynamic PBFT needs to maintain a node information table, which records some information of the node and the corresponding status (Benign, Absent, Malicious). Moreover, a corresponding join and exit mechanism must be designed. This algorithm was improved from PBFT, which can provide better performance but requires more computing power. The WRBFT [99] algorithm was designed primarily to address several issues in relation to the PBFT consensus algorithm. These issues include the fixed nature of the primary node selection, the high communication overhead during the consensus process, and the inability of nodes to dynamically join or exit. To overcome these problems, the WRBFT algorithm utilizes a verifiable random function (VRF) in conjunction with the node consensus workload, resulting in a more flexible and efficient consensus process.

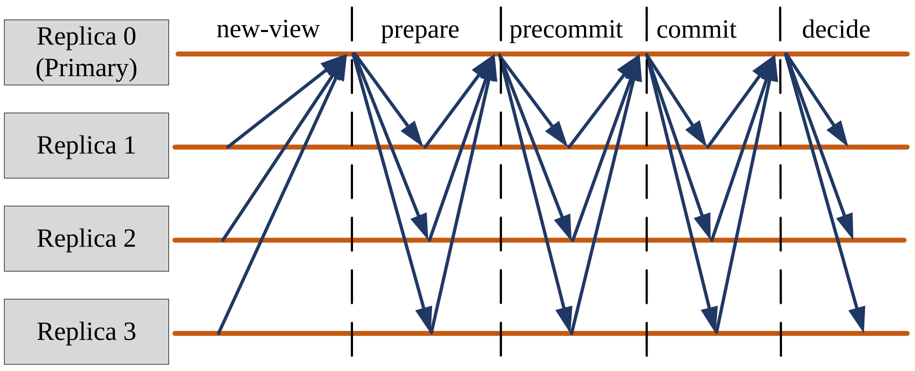

Figure 6. HotStuff consensus process [76].

In general, when designing a BFT protocol, one first puts forward the premise of the protocol, but this will also cause the protocol to fail to operate normally once the protocol is used in different settings. An innovative approach to separate fault model and protocol design was proposed in Flexible BFT [100]. Flexible BFT features two major innovations to support its high performance. The first innovation of this algorithm lies in the design of a brand new fault model, alive-but-corrupt faults, meaning that this model can bring stronger resistance to faults compared to the BFT consensus protocol. The second innovation is that Flexible BFT uses a specific method to separate the failure model from the consensus protocol. This separation enables Flexible BFT to deal with various situations, that is, to have diversity. Moreover, Flexible BFT provided a new understanding of the arbitration group (such as $2f + 1$) in PBFT, introduced the concept of the learner, and proposed locking and unlocking when conflicts arise. There is no doubt that the protocol has very good performance, including higher security, improved efficiency, super-adaptability, and resistance, and the algorithm can be used in scenarios that require strong applicability and resistance. However, Flexible BFT involves quite a lot of assumptions. Without these assumptions, the performance of the protocol may decrease. Therefore, maintaining these advantages without these assumptions is one of the directions for future research and improvement.

Tendermint [101] employs an agreement procedure that resembles the PBFT consensus algorithm, made up of three steps: pre-vote, pre-commit, and commit. This procedure is employed to attain consensus in scenarios where the quantity of Byzantine nodes is at most 1/3 of the total number of nodes. Tendermint's three-stage optimization process is shown in Figure 7. In Tendermint, nodes that participate in the voting process are designated as verifier nodes. Verifier nodes are chosen randomly, and a proposer is then selected through proof of stake (POS) to oversee and gather all network transactions. The algorithm determines blocks by epoch, running a round-robin protocol at each epoch to determine the next block. All information in Tendermint is stored in the blockchain, and the view change mechanism is also eliminated. When the network times out or the primary replica fails and the primary replica needs to be replaced, Tendermint's processing flow is the same as the normal consensus process. Tendermint is a high-throughput consensus algorithm that can handle thousands of transactions per second across multiple nodes distributed worldwide with a latency of approximately 1 second. The main distinction between Tendermint and PBFT is that Tendermint utilizes a specialized empty block when processing timeouts, whereas PBFT uses a view change mechanism to replace the primary node. However, when faced with malicious attacks, Tendermint's performance may be slightly degraded.

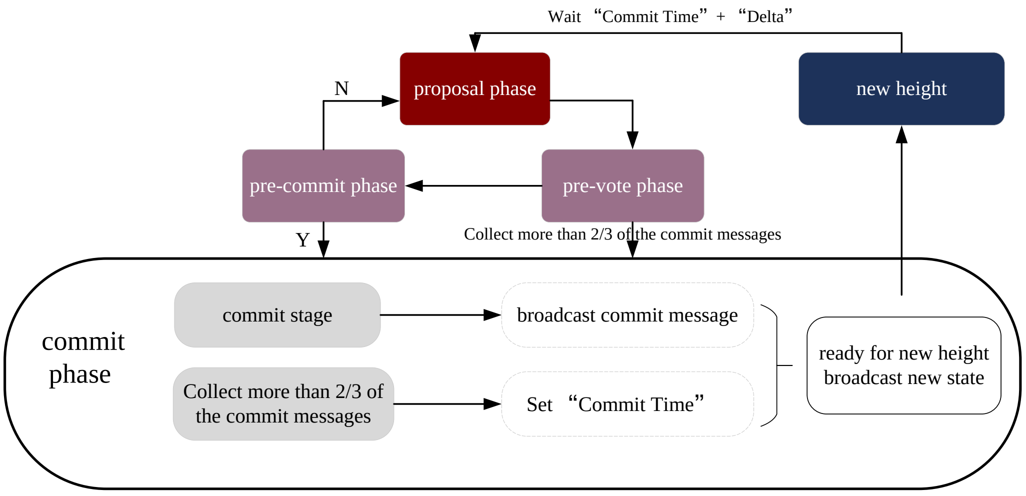
Wait "Commit Time" + "Delta"
proposal phase
new height
N
pre-commit phase
pre-vote phase
Y
Collect more than 2/3 of the commit messages
commit phase
commit stage
broadcast commit message
Collect more than 2/3 of the commit messages
Set "Commit Time"
ready for new height broadcast new state

Figure 7. Tendermint's three-stage optimization process [101].

Asynchronous networks, where message delivery times are unpredictable, have posed a significant challenge for achieving consensus in distributed systems. Without making any assumptions about time, the FLP impossibility theorem [107] demonstrates that achieving deterministic consensus in asynchronous networks with even one faulty node is impossible. However, the HoneyBadgerBFT (HBBFT) protocol [102] has made significant progress in overcoming this challenge. HBBFT is a BFT consensus protocol that provides a practical and high-throughput solution for asynchronous networks, guaranteeing liveness without any time assumptions. HBBFT achieves this by utilizing an innovative atomic broadcast protocol that can tolerate up to f faulty nodes, provided the total number of nodes is no less than $3f + 1$. HBBFT's practicality and efficiency make it an attractive option for implementing distributed systems that require BFT consensus in asynchronous networks.

Despite the success of HBBFT, further improvements have been made in the form of the BEAT protocol [108]. BEAT offers superior performance compared to HBBFT, particularly in terms of delay and throughput. BEAT comprises a family of five asynchronous BFT protocols (BEAT0-BEAT5), each tailored for specific use cases. BEAT0-BEAT2 were designed for state machine replication, while BEAT3 and BEAT4 are better suited for BFT storage systems. The BEAT protocols provide a more efficient and robust solution for achieving consensus in asynchronous networks, making them a promising option for implementing large-scale distributed systems. In summary, the development of the HBBFT and BEAT protocols has significantly improved the practicality and efficiency of achieving Byzantine fault-tolerant consensus in asynchronous networks. These protocols provide robust solutions that are suitable for different use cases, offering promising options for implementing large-scale distributed systems in the future.

DispersedLedger [103] is a novel restructuring that utilizes the BFT protocol to separate the bandwidth-intensive task of block downloading from the protocol. This approach results in improved throughput and reduced latency when compared to HBBFT. In addition, AVID-M, an asynchronous VID protocol based on [102], was proposed in DispersedLedger to enhance communication efficiency. Guo's proposed solution, Dumbo [104], not only achieved superior throughput and lower latency, but also exhibited greater tolerance towards larger system sizes. HBBFT's main idea is to use Ben-Or's asynchronous common subset protocol [109] for batch consensus, as well as the reliable broadcast protocol (RBC) with the number of m and asynchronous binary agreement (ABA) with the number of *m*. Dumbo introduced two novel atomic broadcast protocols (Dumbo1 and Dumbo2) that employ the fundamental concept of HBBFT, both of which exhibit better efficiency in both theory and practice. The core idea of Dumbo-NG [105] is a non-trivial direct reduction from asynchronous atomic broadcasting to multi-valued verifiable Byzantine agreement (MVBA) with quality properties (ensuring that MVBA outputs come from honest nodes with a probability of 1/2). The protocol supports the complete parallelism of transaction propagation and the asynchronous protocol. While effectively resolving the tension between throughput and delay, it also allows any honest node to broadcast a transaction that can agree on the output [105]. In addition to Dumbo and Dumbo-NG, there are many studies focused on exploring whether asynchronous protocols can really be achieved [110–114].

BeauForT is a platform that is entirely web-based and is utilized for achieving decentralized BFT consensus in applications that are driven by community needs and focused on clients. It optimizes for unstable network conditions with high latency, i.e., it is not necessary for all replicas to be directly linked to one another, and the consensus algorithm does not rely on the presence of a leader. Not relying on a leader means that a lot of the cost of leader election and change can be saved. BeauForT uses an aggregated signature scheme called BLS [115] to reduce storage and bandwidth requirements while also reducing computational costs [106]. The algorithm is robust to smaller-scale networks, where it can cope well with the effects of short-term outages and outages caused by mobility. However, its scalability is limited due to the absence of a master node.

In addition to the BFT consensus algorithms listed above, there are many other excellent consensus algorithms in each category, though this article will not list them one by one. Figure 8 shows a summary of the BFT consensus algorithms mentioned in this paper.

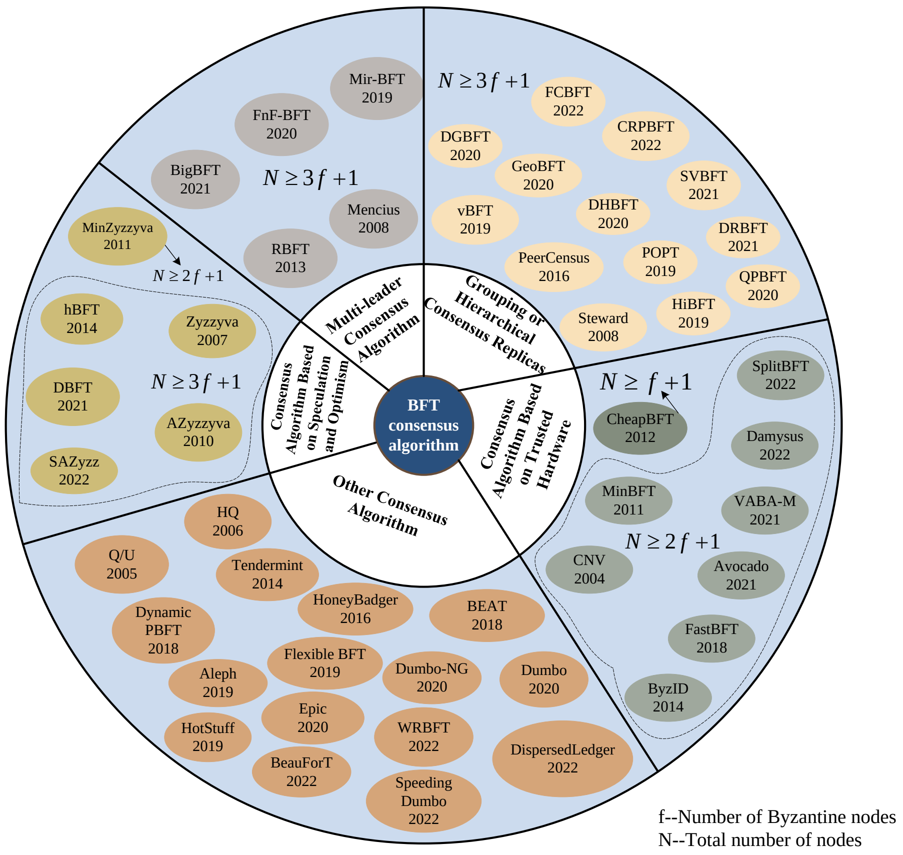

Figure 8. Summary of the BFT consensus algorithms.

4. Discussion

Based on the current situation, there is still much research to be carried out on BFT consensus algorithms. This section will discuss the results of this survey and the technical updates that can be brought about by the advancement of BFT consensus algorithms.

(1) PBFT is the most classic BFT consensus algorithm. This algorithm has strong performance when the number of nodes is small, but as the number of nodes increases, the performance of the algorithm drops sharply, that is, the scalability of the algorithm is low. In order to achieve consensus, PBFT also requires huge performance overheads, such as the need to store a large number of communication messages. In addition, PBFT also requires stable network conditions. Once this condition is not met, it will cause high latency problems. PBFT also has other limitations, such as a lack of the reasonable evaluation of nodes and strong closure. Although this algorithm has existed for a while, according to the current survey, adding other methods on the basis of PBFT can greatly improve the performance of the BFT consensus algorithm. There are still many BFT consensus algorithms that have been improved based on PBFT, so researchers can consider conducting in-depth research on the basis of PBFT. (2) The method of layering or grouping the nodes that participate in the consensus process can reduce the complexity of the BFT consensus algorithm and improve the elasticity to a certain extent. This is a rather effective approach that has been studied a lot in recent years. The focus of this type of consensus algorithm is still on selection or allocation and subsequent organizational management. A selection algorithm without a comprehensive consideration will directly reduce the performance of this type of consensus algorithm. On the contrary, a good selection algorithm will greatly improve its performance. Furthermore, a layered or grouped approach can introduce load imbalances, increase the cost and complexity of managing each group, and lead to insufficient security. (3) The use of multiple leaders can enhance the efficiency and scalability of the BFT consensus algorithm, but it will also bring problems such as difficulties achieving effective synchronization between multiple leaders, increased system security risks, and increased management complexity. These are the reasons for which multi-leader consensus algorithms are limited in practical applications. For the purpose of further strengthening the degree of decentralization of BFT consensus algorithms, some researchers have also focused on not using leaders in recent years. The method of not using a leader can improve the fairness, decentralization, and fault tolerance of the system, but it has an obvious disadvantage in that it requires more network bandwidth and computing resources. (4) As mentioned in Section 3.3, the method of using speculation can greatly augment the efficiency of BFT consensus algorithms, but the focus is still on how to reduce the security threat caused by the replica directly replying to the client. (5) Adding trusted components to the general BFT consensus algorithm can effectively raise its performance. Increasing the efficiency and security of the BFT consensus algorithm is thus possible. However, because the cost of using hardware is very high, balancing hardware overhead and performance improvement is a key aspect of the method of using hardware.

So far, the development of the BFT consensus algorithm has advanced to a certain extent. Even the BFT consensus problem in an asynchronous network, which many algorithms could not solve in the early days, now has a practical solution algorithm. However, the performance of consensus algorithms in solving the Byzantine generals problem has not reached its peak. Even the newly proposed BFT consensus algorithms have certain defects. Therefore, for researchers, the following two directions can be considered for further research: (1) Improving the consensus algorithms proposed by others to enhance their performance—one can refer to the defects and limitations of some of the algorithms we mentioned above for targeted research. (2) Distributed systems—due to their powerful role and functions, they are also a popular research direction now. Whether distributed computing or distributed storage is considered, the requirements for security and integrity are relatively high. Therefore, the application of BFT consensus algorithms in various distributed systems or scenarios can be considered. For example, the combination of federated learning and BFT consensus algorithms is currently being studied by many researchers.

Studying BFT consensus algorithms is of great benefit to current technological innovation. Blockchain is one of the most well-known applications of BFT consensus algorithms. Many blockchain projects use a BFT consensus algorithm as their core consensus mechanism, such as Libra [116] and Algorand [117]. BFT consensus algorithms can also help distributed database systems ensure data consistency and availability. In emerging cloud computing and edge computing, multiple nodes need to work together to provide elasticity, fault tolerance, and high availability. BFT consensus algorithms can be used in these scenarios to achieve coordination and consistency among multiple nodes. Internet of Things (IoT) systems can use BFT consensus algorithms to help achieve inter-device collaboration and data consistency. In summary, BFT consensus algorithms have broad application potential in various technical fields and have become a key component of distributed system design. With further research, we can expect more innovations and applications of BFT consensus algorithms in the future.

5. Conclusions

This paper comprehensively analyzed BFT consensus protocols by surveying the existing literature, focusing on the improvement and innovation of methods to improve the performance of BFT consensus algorithms. We classified some existing BFT consensus algorithms according to different design concepts, reviewed the core ideas of these design methods, and conducted in-depth research into them. For each category, we also investigated the development process of BFT consensus algorithms using the core ideas of the category and compared their improvements and shortcomings. By reviewing the development of BFT consensus algorithms and analyzing the methods used, we summarized the impact of different design methods on BFT consensus algorithms, looked forward to the future development of BFT consensus algorithms, and discussed the challenges of different design methods and the directions for further research.

Author Contributions: Investigation, W.Z.; resources, W.Z.; modification of English grammar, N.X.; validation, N.X.; writing—original draft preparation, W.Z.; writing—review and editing, N.X., C.Y., W.L., L.C., J.L. and J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created.

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

References

1. Sherman, A.T.; Javani, F.; Zhang, H.; Golaszewski, E. On the origins and variations of blockchain technologies. *IEEE Secur. Priv.* **2019**, *17*, 72–77. [CrossRef]
2. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 25 July 2023).
3. Ante, L. Smart contracts on the blockchain—A bibliometric analysis and review. *Telemat. Inform.* **2021**, *57*, 101519. [CrossRef]
4. Yin, J.; Xiao, Y.; Pei, Q.; Ju, Y.; Liu, L.; Xiao, M.; Wu, C. SmartDID: A novel privacy-preserving identity based on blockchain for IoT. *IEEE Internet Things J.* **2022**, *10*, 6718–6732. [CrossRef]
5. Liang, W.; Yang, Y.; Yang, C.; Hu, Y.; Xie, S.; Li, K.C.; Cao, J. PDPChain: A consortium blockchain-based privacy protection scheme for personal data. *IEEE Trans. Reliab.* **2022**, *72*, 586–598. [CrossRef]
6. Fu, J.; Cao, B.; Wang, X.; Zeng, P.; Liang, W.; Liu, Y. BFS: A blockchain-based financing scheme for logistics company in supply chain finance. *Connect. Sci.* **2022**, *34*, 1929–1955. [CrossRef]
7. Wang, Q.; Zhu, X.; Ni, Y.; Gu, L.; Zhu, H. Blockchain for the IoT and industrial IoT: A review. *Internet Things* **2020**, *10*, 100081. [CrossRef]
8. Hu, W.J.; Fan, J.; Du, Y.X.; Li, B.S.; Xiong, N.; Bekkering, E. MDFC–ResNet: An agricultural IoT system to accurately recognize crop diseases. *IEEE Access* **2020**, *8*, 115287–115298. [CrossRef]
9. Xu, Z.; Liang, W.; Li, K.C.; Xu, J.; Jin, H. A blockchain-based roadside unit-assisted authentication and key agreement protocol for internet of vehicles. *J. Parallel Distrib. Comput.* **2021**, *149*, 29–39. [CrossRef]
10. Qu, J. Blockchain in medical informatics. *J. Ind. Inf. Integr.* **2022**, *25*, 100258. [CrossRef]
11. Kassen, M. Blockchain and e-government innovation: Automation of public information processes. *Inf. Syst.* **2022**, *103*, 101862. [CrossRef]

12. Pandey, V.; Pant, M.; Snasel, V. Blockchain technology in food supply chains: Review and bibliometric analysis. *Technol. Soc.* **2022**, *69*, 101954. [CrossRef]
13. Belchior, R.; Vasconcelos, A.; Guerreiro, S.; Correia, M. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. *ACM Comput. Surv.* **2021**, *54*, 1–41. [CrossRef]
14. Van Renesse, R.; Schneider, F.B. Chain Replication for Supporting High Throughput and Availability. *OSDI* **2004**, *4*, 91–104.
15. Sang, Y.; Shen, H.; Tan, Y.; Xiong, N. Efficient protocols for privacy preserving matching against distributed datasets. In Proceedings of the Information and Communications Security: 8th International Conference, ICICS 2006, Raleigh, NC, USA, 4–7 December 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 210–227.
16. Ongaro, D.; Ousterhout, J. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 305–319.
17. Yang, Y.; Xiong, N.; Chong, N.Y.; Defago, X. A decentralized and adaptive flocking algorithm for autonomous mobile robots. In Proceedings of the 2008 The 3rd International Conference on Grid and Pervasive Computing-Workshops, Kunming, China, 25–28 May 2008; pp. 262–268.
18. Xiong, N.; Vasilakos, A.V.; Wu, J.; Yang, Y.R.; Rindos, A.; Zhou, Y.; Song, W.Z.; Pan, Y. A self-tuning failure detection scheme for cloud computing service. In Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, Shanghai, China, 21–25 May 2012; pp. 668–679.
19. Guru, A.; Mohanta, B.K.; Mohapatra, H.; Al-Turjman, F.; Altrjman, C.; Yadav, A. A Survey on Consensus Protocols and Attacks on Blockchain Technology. *Appl. Sci.* **2023**, *13*, 2604. [CrossRef]
20. Zhou, S.; Li, K.; Xiao, L.; Cai, J.; Liang, W.; Castiglione, A. A Systematic Review of Consensus Mechanisms in Blockchain. *Mathematics* **2023**, *11*, 2248. [CrossRef]
21. Sayeed, S.; Marco-Gisbert, H. Assessing blockchain consensus and security mechanisms against the 51% attack. *Appl. Sci.* **2019**, *9*, 1788. [CrossRef]
22. Nguyen, G.T.; Kim, K. A survey about consensus algorithms used in blockchain. *J. Inf. Process. Syst.* **2018**, *14*, 101–128.
23. Pease, M.; Shostak, R.; Lamport, L. Reaching agreement in the presence of faults. *J. ACM* **1980**, *27*, 228–234. [CrossRef]
24. Lamport, L. The part-time parliament. In *Concurrency: The Works of Leslie Lamport*; ACM Books: New York, NY, USA, 2019; pp. 277–317.
25. Lamport, L.; Shostak, R.; Pease, M. The Byzantine generals problem. In *Concurrency: The Works of Leslie Lamport*; ACM Books: New York, NY, USA, 2019; pp. 203–226.
26. Wang, Z.; Li, T.; Xiong, N.; Pan, Y. A novel dynamic network data replication scheme based on historical access record and proactive deletion. *J. Supercomput.* **2012**, *62*, 227–250. [CrossRef]
27. Zhang, G.; Pan, F.; Dang'ana, M.; Mao, Y.; Motepalli, S.; Zhang, S.; Jacobsen, H.A. Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms. *arXiv* **2022**, arXiv:2204.03181.
28. Wensley, J.H.; Lamport, L.; Goldberg, J.; Green, M.W.; Levitt, K.N.; Melliar-Smith, P.M.; Shostak, R.E.; Weinstock, C.B. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proc. IEEE* **1978**, *66*, 1240–1255. [CrossRef]
29. Hopkins, A.L., Jr.; Lala, J.H.; Smith, T.B., III. The evolution of fault tolerant computing at the Charles Stark Draper Laboratory, 1955–1985. In *The Evolution of Fault-Tolerant Computing: In the Honor of William C. Carter*; Springer: Vienna, Austria, 1987; pp. 121–140.
30. Driscoll, K. Multi-Microprocessor Flight Control System, 1982. In Proceedings of the 5th Digital Avionics Systems Conference, Seattle, WA, USA, 31 October–3 November 1983; p. 11.
31. Castro, M.; Liskov, B. Practical byzantine fault tolerance. *OsDI* **1999**, *99*, 173–186.
32. Lamport, L. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: The Works of Leslie Lamport*; ACM Books: New York, NY, USA, 2019; pp. 179–196.
33. Zhou, Y.; Zhang, Y.; Liu, H.; Xiong, N.; Vasilakos, A.V. A bare-metal and asymmetric partitioning approach to client virtualization. *IEEE Trans. Serv. Comput.* **2012**, *7*, 40–53. [CrossRef]
34. Thai, Q.T.; Yim, J.C.; Yoo, T.W.; Yoo, H.K.; Kwak, J.Y.; Kim, S.M. Hierarchical Byzantine fault-tolerance protocol for permissioned blockchain systems. *J. Supercomput.* **2019**, *75*, 7337–7365. [CrossRef]
35. Wang, H.; Guo, K. Byzantine fault tolerant algorithm based on vote. In Proceedings of the 2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Guilin, China, 17–19 October 2019; pp. 190–196.
36. Yu, G.; Wu, B.; Niu, X. Improved blockchain consensus mechanism based on PBFT algorithm. In Proceedings of the 2020 2nd International Conference on Advances in Computer Technology, Information Science and Communications (CTISC), Suzhou, China, 20–22 March 2020; pp. 14–21.
37. Zhang, Z.; Zhu, D.; Fan, W. Qpbft: Practical byzantine fault tolerance consensus algorithm based on quantified-role. In Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 29 December 2020–1 January 2021; pp. 991–997.
38. Gupta, S.; Rahnama, S.; Hellings, J.; Sadoghi, M. Resilientdb: Global scale resilient blockchain fabric. *arXiv* **2020**, arXiv:2002.00160.
39. Chen, Y.; Li, M.; Zhu, X.; Fang, K.; Ren, Q.; Guo, T.; Chen, X.; Li, C.; Zou, Z.; Deng, Y. An improved algorithm for practical byzantine fault tolerance to large-scale consortium chain. *Inf. Process. Manag.* **2022**, *59*, 102884. [CrossRef]
40. Xiang, F.; Huaimin, W.; Peichang, S. Proof of previous transactions (PoPT): An efficient approach to consensus for JCLedger. *IEEE Trans. Syst. Man Cybern. Syst.* **2019**, *51*, 2415–2424. [CrossRef]

41. Li, F.; Liu, K.; Liu, J.; Fan, Y.; Wang, S. DHBFT: Dynamic hierarchical Byzantine fault-tolerant consensus mechanism based on credit. In Proceedings of the Web and Big Data: 4th International Joint Conference, APWeb-WAIM 2020, Tianjin, China, 18–20 September 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 3–17.
42. Zhan, Y.; Wang, B.; Lu, R.; Yu, Y. DRBFT: Delegated randomization Byzantine fault tolerance consensus protocol for blockchains. *Inf. Sci.* **2021**, *559*, 8–21. [CrossRef]
43. Li, C.; Zhang, J.; Yang, X.; Youlong, L. Lightweight blockchain consensus mechanism and storage optimization for resource-constrained IoT devices. *Inf. Process. Manag.* **2021**, *58*, 102602. [CrossRef]
44. Decker, C.; Seidel, J.; Wattenhofer, R. Bitcoin meets strong consistency. In Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, 4–7 January 2016; pp. 1–10.
45. Amir, Y.; Danilov, C.; Dolev, D.; Kirsch, J.; Lane, J.; Nita-Rotaru, C.; Olsen, J.; Zage, D. Steward: Scaling byzantine fault-tolerant replication to wide area networks. *IEEE Trans. Dependable Secur. Comput.* **2008**, *7*, 80–93. [CrossRef]
46. Lamport, L.; Massa, M. Cheap paxos. In Proceedings of the International Conference on Dependable Systems and Networks, Florence, Italy, 28 June–1 July 2004; pp. 307–314.
47. Zeng, Y.; Sreenan, C.J.; Xiong, N.; Yang, L.T.; Park, J.H. Connectivity and coverage maintenance in wireless sensor networks. *J. Supercomput.* **2010**, *52*, 23–46. [CrossRef]
48. Yang, J.; Xiong, N.; Vasilakos, A.V.; Fang, Z.; Park, D.; Xu, X.; Yoon, S.; Xie, S.; Yang, Y. A fingerprint recognition scheme based on assembling invariant moments for cloud computing communications. *IEEE Syst. J.* **2011**, *5*, 574–583. [CrossRef]
49. Wu, C.; Ju, B.; Wu, Y.; Lin, X.; Xiong, N.; Xu, G.; Li, H.; Liang, X. UAV autonomous target search based on deep reinforcement learning in complex disaster scene. *IEEE Access* **2019**, *7*, 117227–117245. [CrossRef]
50. Qi, J.; Guan, Y. Practical Byzantine fault tolerance consensus based on comprehensive reputation. *Peer Peer Netw. Appl.* **2023**, *16*, 420–430. [CrossRef]
51. Wu, X.; Ling, H.; Liu, H.; Yu, F. A privacy-preserving and efficient byzantine consensus through multi-signature with ring. *Peer Peer Netw. Appl.* **2022**, *15*, 1669–1684. [CrossRef]
52. Rivest, R.L.; Shamir, A.; Tauman, Y. How to leak a secret. In Proceedings of the Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, 9–13 December 2001; Springer: Berlin/Heidelberg, Germany, 2001; pp. 552–565.
53. Schnorr, C.P. Efficient signature generation by smart cards. *J. Cryptol.* **1991**, *4*, 161–174. [CrossRef]
54. Maxwell, G.; Poelstra, A.; Seurin, Y.; Wuille, P. Simple schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptogr.* **2019**, *87*, 2139–2164. [CrossRef]
55. Li, Y.; Qiao, L.; Lv, Z. An optimized byzantine fault tolerance algorithm for consortium blockchain. *Peer Peer Netw. Appl.* **2021**, *14*, 2826–2839. [CrossRef]
56. Gao, Y.; Xiang, X.; Xiong, N.; Huang, B.; Lee, H.J.; Alrifai, R.; Jiang, X.; Fang, Z. Human action monitoring for healthcare based on deep learning. *IEEE Access* **2018**, *6*, 52277–52285. [CrossRef]
57. Deng, Y.; Hu, H.; Xiong, N.; Xiong, W.; Liu, L. A general hybrid model for chaos robust synchronization and degradation reduction. *Inf. Sci.* **2015**, *305*, 146–164. [CrossRef]
58. Pass, R.; Shi, E. Hybrid consensus: Efficient consensus in the permissionless model. *Cryptol. Eprint Arch.* **2016**. Available online: https://eprint.iacr.org/2016/917 (accessed on 26 July 2023).
59. Pass, R.; Shi, E. Thunderella: Blockchains with optimistic instant confirmation. In Proceedings of the Advances in Cryptology—EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, 29 April–3 May 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 3–33.
60. Luu, L.; Narayanan, V.; Zheng, C.; Baweja, K.; Gilbert, S.; Saxena, P. A secure sharding protocol for open blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 25–27 October 2016; pp. 17–30.
61. Al-Bassam, M.; Sonnino, A.; Bano, S.; Hrycyszyn, D.; Danezis, G. Chainspace: A sharded smart contracts platform. *arXiv* **2017**, arXiv:1708.03778.
62. Liu, Y.; Liu, J.; Zhang, Z.; Yu, H. A fair selection protocol for committee-based permissionless blockchains. *Comput. Secur.* **2020**, *91*, 101718. [CrossRef]
63. Ma, F.Q.; Fan, R.N. Queuing Theory of Improved Practical Byzantine Fault Tolerant Consensus. *Mathematics* **2022**, *10*, 182. [CrossRef]
64. Fang, W.; Li, Y.; Zhang, H.; Xiong, N.; Lai, J.; Vasilakos, A.V. On the throughput-energy tradeoff for data transmission between cloud and mobile devices. *Inf. Sci.* **2014**, *283*, 79–93. [CrossRef]
65. Shu, L.; Zhang, Y.; Yu, Z.; Yang, L.T.; Hauswirth, M.; Xiong, N. Context-aware cross-layer optimized video streaming in wireless multimedia sensor networks. *J. Supercomput.* **2010**, *54*, 94–121. [CrossRef]
66. Borran, F.; Schiper, A. Brief announcement: A leader-free byzantine consensus algorithm. In Proceedings of the Distributed Computing: 23rd International Symposium, DISC 2009, Elche, Spain, 23–25 September 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 479–480.
67. Barcelona, C.S. Mencius: Building efficient replicated state machines for WANs. In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 08), San Diego, CA, USA, 8–10 December 2008.

68. Sandor, V.K.A.; Lin, Y.; Li, X.; Lin, F.; Zhang, S. Efficient decentralized multi-authority attribute based encryption for mobile cloud data storage. *J. Netw. Comput. Appl.* **2019**, *129*, 25–36. [CrossRef]
69. Milosevic, Z.; Biely, M.; Schiper, A. Bounded delay in byzantine-tolerant state machine replication. In Proceedings of the 2013 IEEE 32nd International Symposium on Reliable Distributed Systems, Braga, Portugal, 1–3 October 2013; pp. 61–70.
70. Lin, C.; He, Y.X.; Xiong, N. An energy-efficient dynamic power management in wireless sensor networks. In Proceedings of the 2006 Fifth International Symposium on Parallel and Distributed Computing, Timisoara, Romania, 6–9 July 2006; pp. 148–154.
71. Aublin, P.L.; Mokhtar, S.B.; Quéma, V. Rbft: Redundant byzantine fault tolerance. In Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems, Philadelphia, PA, USA, 8–11 July 2013; pp. 297–306.
72. Stathakopoulou, C.; David, T.; Vukolic, M. Mir-bft: High-throughput bft for blockchains. *arXiv* **2019**, arXiv:1906.05552.
73. Junqueira, F.P.; Reed, B.C.; Serafini, M. Zab: High-performance broadcast for primary-backup systems. In Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), Hong Kong, China, 27–30 June 2011; pp. 245–256.
74. Alqahtani, S.; Demirbas, M. Bigbft: A multileader byzantine fault tolerance protocol for high throughput. In Proceedings of the 2021 IEEE International Performance, Computing, and Communications Conference (IPCCC), Austin, TX, USA, 28–30 October 2021; pp. 1–10.
75. Avarikioti, Z.; Heimbach, L.; Schmid, R.; Vanbever, L.; Wattenhofer, R.; Wintermeyer, P. Fnf-bft: Exploring performance limits of BFT protocols. *arXiv* **2020**, arXiv:2009.02235.
76. Yin, M.; Malkhi, D.; Reiter, M.K.; Gueta, G.G.; Abraham, I. HotStuff: BFT consensus with linearity and responsiveness. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, Toronto, ON, Canada, 29 July–2 August 2019; pp. 347–356.
77. Kotla, R.; Alvisi, L.; Dahlin, M.; Clement, A.; Wong, E. Zyzzyva: Speculative byzantine fault tolerance. In Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, Stevenson, DC, USA, 14–17 October 2007; pp. 45–58.
78. Guerraoui, R.; Knežević, N.; Quéma, V.; Vukolić, M. The next 700 BFT protocols. In Proceedings of the 5th European Conference on Computer Systems, Paris, France, 13–16 April 2010; pp. 363–376.
79. Veronese, G.S.; Correia, M.; Bessani, A.N.; Lung, L.C.; Verissimo, P. Efficient byzantine fault-tolerance. *IEEE Trans. Comput.* **2011**, *62*, 16–30. [CrossRef]
80. Duan, S.; Peisert, S.; Levitt, K.N. hBFT: Speculative Byzantine fault tolerance with minimum cost. *IEEE Trans. Dependable Secur. Comput.* **2014**, *12*, 58–70. [CrossRef]
81. Sohrabi, N.; Tari, Z.; Voron, G.; Gramoli, V.; Fu, Q. SAZyzz: Scaling AZyzzyva to Meet Blockchain Requirements. *IEEE Trans. Serv. Comput.* **2022**, *16*, 2139–2152. [CrossRef]
82. Zhang, J.; Rong, Y.; Cao, J.; Rong, C.; Bian, J.; Wu, W. DBFT: A Byzantine fault tolerance protocol with graceful performance degradation. *IEEE Trans. Dependable Secur. Comput.* **2021**, *19*, 3387–3400. [CrossRef]
83. Gunn, L.J.; Liu, J.; Vavala, B.; Asokan, N. Making speculative BFT resilient with trusted monotonic counters. In Proceedings of the 2019 38th Symposium on Reliable Distributed Systems (SRDS), Lyon, France, 1–4 October 2019. [CrossRef]
84. Correia, M.; Neves, N.F.; Verissimo, P. How to tolerate half less one Byzantine nodes in practical distributed systems. In Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, Florianopolis, Brazil, 18–20 October 2004; pp. 174–183.
85. Kapitza, R.; Behl, J.; Cachin, C.; Distler, T.; Kuhnle, S.; Mohammadi, S.V.; Schröder-Preikschat, W.; Stengel, K. CheapBFT: Resource-efficient Byzantine fault tolerance. In Proceedings of the 7th ACM European Conference on Computer Systems, Bern, Switzerland, 10–13 April 2012; pp. 295–308.
86. Duan, S.; Levitt, K.; Meling, H.; Peisert, S.; Zhang, H. ByzID: Byzantine fault tolerance from intrusion detection. In Proceedings of the 2014 IEEE 33rd International Symposium on Reliable Distributed Systems, Nara, Japan, 6–9 October 2014; pp. 253–264.
87. Liu, J.; Li, W.; Karame, G.O.; Asokan, N. Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Trans. Comput.* **2018**, *68*, 139–151. [CrossRef]
88. Bailleu, M.; Giantsidi, D.; Gavrielatos, V.; Quoc, D.L.; Nagarajan, V.; Bhatotia, P. Avocado: A Secure In-Memory Distributed Storage System. In Proceedings of the USENIX Annual Technical Conference, Virtual, 14–16 July 2021; pp. 65–79.
89. Decouchant, J.; Kozhaya, D.; Rahli, V.; Yu, J. DAMYSUS: Streamlined BFT consensus leveraging trusted components. In Proceedings of the Seventeenth European Conference on Computer Systems, Rennes, France, 5–8 April 2022; pp. 1–16.
90. Messadi, I.; Becker, M.H.; Bleeke, K.; Jehl, L.; Mokhtar, S.B.; Kapitza, R. SplitBFT: Improving Byzantine Fault Tolerance Safety Using Trusted Compartments. In Proceedings of the 23rd Conference on 23rd ACM/IFIP International Middleware Conference, Quebec, QC, Canada, 7–11 November 2022; pp. 56–68.
91. Wang, X.; Duan, S.; Clavin, J.; Zhang, H. Bft in blockchains: From protocols to use cases. *ACM Comput. Surv. (CSUR)* **2022**, *54*, 1–37. [CrossRef]
92. Xie, S.; Liang, W.; Xu, J.; Tang, M.; Weng, T.H.; Li, K.C. A novel bidirectional RFID identity authentication protocol. In Proceedings of the 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Guangzhou, China, 8–12 October 2018; pp. 301–307.
93. Zhao, J.; Huang, J.; Xiong, N. An effective exponential-based trust and reputation evaluation system in wireless sensor networks. *IEEE Access* **2019**, *7*, 33859–33869. [CrossRef]

94. Lynch, N.A.; Shvartsman, A.A. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In Proceedings of the IEEE 27th International Symposium on Fault Tolerant Computing, Seattle, WA, USA, 24–27 June 1997; pp. 272–281.
95. Yandamuri, S.; Abraham, I.; Nayak, K.; Reiter, M. Brief announcement: Communication-efficient BFT using small trusted hardware to tolerate minority corruption. In Proceedings of the 35th International Symposium on Distributed Computing (DISC 2021), Freiburg, Germany, 4–8 October 2021; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Wadern, Germany, 2021.
96. Abd-El-Malek, M.; Ganger, G.R.; Goodson, G.R.; Reiter, M.K.; Wylie, J.J. Fault-scalable Byzantine fault-tolerant services. *ACM SIGOPS Oper. Syst. Rev.* **2005**, *39*, 59–74. [CrossRef]
97. Cowling, J.; Myers, D.; Liskov, B.; Rodrigues, R.; Shrira, L. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In Proceedings of the 7th symposium on Operating Systems Design and Implementation, Seattle, WA, USA, 6–8 November 2006; pp. 177–190.
98. Hao, X.; Yu, L.; Zhiqiang, L.; Zhen, L.; Dawu, G. Dynamic practical byzantine fault tolerance. In Proceedings of the 2018 IEEE Conference on Communications and Network Security (CNS), Beijing, China, 30 May–1 June 2018; pp. 1–8.
99. Huang, B.; Peng, L.; Zhao, W.; Chen, N. Workload-based randomization byzantine fault tolerance consensus protocol. *High-Confid. Comput.* **2022**, *2*, 100070. [CrossRef]
100. Malkhi, D.; Nayak, K.; Ren, L. Flexible byzantine fault tolerance. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 1041–1053.
101. Buchman, E. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains. Ph.D. Thesis, University of Guelph, Guelph, ON, Canada, 2016.
102. Miller, A.; Xia, Y.; Croman, K.; Shi, E.; Song, D. The honey badger of BFT protocols. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 31–42.
103. Yang, L.; Park, S.J.; Alizadeh, M.; Kannan, S.; Tse, D. {DispersedLedger}:{High-Throughput} Byzantine Consensus on Variable Bandwidth Networks. In Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), Renton, WA, USA, 4–6 April 2022; pp. 493–512.
104. Guo, B.; Lu, Z.; Tang, Q.; Xu, J.; Zhang, Z. Dumbo: Faster asynchronous bft protocols. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020; pp. 803–818.
105. Gao, Y.; Lu, Y.; Lu, Z.; Tang, Q.; Xu, J.; Zhang, Z. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November 2022; pp. 1187–1201.
106. Jannes, K.; Beni, E.H.; Lagaisse, B.; Joosen, W. BeauForT: Robust Byzantine Fault Tolerance for Client-Centric Mobile Web Applications. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 1241–1252. [CrossRef]
107. Fischer, M.J.; Lynch, N.A.; Paterson, M.S. Impossibility of distributed consensus with one faulty process. *J. ACM* **1985**, *32*, 374–382. [CrossRef]
108. Duan, S.; Reiter, M.K.; Zhang, H. BEAT: Asynchronous BFT made practical. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 2028–2041.
109. Ben-Or, M.; Kelmer, B.; Rabin, T. Asynchronous secure computations with optimal resilience. In Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, USA, 14–17 August 1994; pp. 183–192.
110. Abraham, I.; Malkhi, D.; Spiegelman, A. Asymptotically optimal validated asynchronous byzantine agreement. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, Toronto, ON, Canada, 29 July–2 August 2019; pp. 337–346.
111. Liu, C.; Duan, S.; Zhang, H. Epic: Efficient asynchronous bft with adaptive security. In Proceedings of the 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Valencia, Spain, 29 June–2 July 2020; pp. 437–451.
112. Gągol, A.; Leśniak, D.; Straszak, D.; Świętek, M. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies, Zurich, Switzerland, 21–23 October 2019; pp. 214–228.
113. Keidar, I.; Kokoris-Kogias, E.; Naor, O.; Spiegelman, A. All you need is dag. In Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, Virtual, 26–31 July 2021; pp. 165–175.
114. Guo, B.; Lu, Y.; Lu, Z.; Tang, Q.; Xu, J.; Zhang, Z. Speeding dumbo: Pushing asynchronous bft closer to practice. *Cryptol. Eprint Arch.* **2022**. Available online: https://eprint.iacr.org/2022/027 (accessed on 25 July 2023).
115. Boneh, D.; Lynn, B.; Shacham, H. Short signatures from the Weil pairing. *J. Cryptol.* **2004**, *17*, 297–319. [CrossRef]
116. Baudet, M.; Ching, A.; Chursin, A.; Danezis, G.; Garillot, F.; Li, Z.; Malkhi, D.; Naor, O.; Perelman, D.; Sonnino, A. State machine replication in the libra blockchain. *Libra Assn. Tech. Rep.* **2019**. Available online: https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-in-the-libra-blockchain/2019-09-19.pdf (accessed on 25 July 2023).
117. Gilad, Y.; Hemo, R.; Micali, S.; Vlachos, G.; Zeldovich, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, 28–31 October 2017; pp. 51–68.