Buterin's Scalability Trilemma viewed through a State-change-based Classification for Common Consensus Algorithms

Amani Altarawneh
Computer Science and Engineering
University of Tennessee at Chattanooga
Chattanooga, TN
jwh247@mocs.utc.edu

Tom Herschberg
Computer Science and Engineering
University of Tennessee at Chattanooga
Chattanooga, TN
mgr992@mocs.utc.edu

Sai Medury
Computer Science and Engineering
University of Tennessee at Chattanooga
Chattanooga, TN
sai-medury@mocs.utc.edu

Farah Kandah
Computer Science and Engineering
University of Tennessee at Chattanooga
Chattanooga, TN
farah-kandah@utc.edu

Anthony Skjellum
Computer Science and Engineering
University of Tennessee at Chattanooga
Chattanooga, TN
tony-skjellum@utc.edu

***Abstract*—Consensus algorithms in distributed systems have attracted much attention in recent studies. However, there is a need for a classification that leads to better understanding and helps direct the deployment of such algorithms. In this paper, we classify common consensus algorithms based on how they decide the order of system state changes. We then determine the extent to which each category prioritizes scalability, decentralization, and security. As with other engineering design scenarios, this is a choose-two tradeoff. Our key contribution is that, based on this taxonomy of tradeoffs, we are able to discern the types of consensus algorithms that work well within the application area(s) for a given distributed system. We find that a dichotomy of algorithms between leader-based and voting-based consensus algorithms emerges from this taxonomy. Applications of this classification scheme include several different categories of distributed ledgers such as blockchains and directed acyclic graphs (DAGs).**

Index Terms*— *Consensus algorithms, Blockchain, Distributed ledger technology (DLTs), Directed acyclic graphs, Scalability Trilemma, Decentralization, Blockchain security, Leader-based consensus, Voting-based consensus, Gossip voting, Byzantine Fault Tolerance, Internet-of-Things (IoT)

I. Introduction

Many of the systems that modern society depends on such as hospitals, banks, stock exchanges, and airports need highly reliable and available systems to function properly. These systems, however, often experience single points of failure in some part of their architecture, which makes them susceptible to crashes, downtime, and/or hacking. Redundancy can be achieved by creating a network of connected machines and distributing work and resources across that network. For such distributed systems to function properly, all machines must be able to agree on key aspects of the current state of the system; otherwise, the system would not be able to work as a coherent group and any benefits associated with distributing the system would be lost. Furthermore, the distributed system must be able to withstand the failure of a subset of its members in order to be resilient enough to be highly available. Distributed systems must also be safeguarded against insider threats arising from circumstances of machine compromise. The process of agreeing on the current state and deciding what the next state should be is called reaching consensus [1], which in the case of distributed systems, is typically carried through the use of consensus algorithms. Distributed systems that reach consensus on the current state are known as replicated state machines [2], which can be used to resolve problems related to fault tolerance that distributed systems encounter.

Blockchains are a type of distributed system in which each server, typically called a node, contains a state machine whose state must match all other nodes' state, without any node necessarily knowing or trusting the other nodes in the system. In order to reach consensus, each blockchain's consensus algorithm must be able to guide the process of validating the current blockchain, help decide which node should create the next state (i.e., the next block), assist in validating the next state, and ensure that all nodes agree on the next state [3]. An effective consensus algorithm must carry out these procedures in an objective and mathematically rigorous manner to guarantee that all nodes agree on the state of the system, while preventing the system from failing because of faulty or malicious behavior [4]. Many different consensus algorithms have been created to perform these actions as quickly and securely as possible in a decentralized environment. Because of this set of possible solutions, it is valuable to categorize each consensus algorithm so they can be compared and contrasted. Additionally, identifying each category's strengths and weaknesses can both assist an individual or entity in choosing a more vs. less appropriate consensus algorithm for their specific use case and help researchers identify a need for new consensus algorithm(s). However, a classification for consensus algorithms from different types of distributed systems that examines system state ordering evidently does not exist at present. Therefore, we seek to close that gap by presenting a simpler classification scheme that can be used to categorize consensus algorithms from several different types of distributed systems based on how the order of state changes is decided.

The remainder of this paper is organized as follows: Background and related work are described in Section II. Our classification scheme for consensus algorithms, based on how system state changes are ordered, is described in Section III. The extent to which each category of consensus algorithms prioritizes scalability, decentralization, and security, as well as the benefits, drawbacks, and use cases of each category are described in Section IV. Finally, Section V offers conclusions.

II. Background and Related Work

First, it is important to differentiate distributed ledger technology (DLT) from blockchains and directed acyclic graphs (DAGs). A DLT is a set of data that is replicated, shared, synchronized, and spread across multiple sites globally with no central administration or centralized data storage [5]. DLTs have several desirable properties such as immutability, transparency, anonymity, trust, and decentralization [6], [7]. Immutability means that once a transaction exists within the ledger, it cannot be changed. DLTs are transparent because they provide a transparent record of transactions without any intermediates or third party, and all the participants in the network have access to the information that is recorded in the ledger. While the identities of the participants in the network in DLTs are anonymous, DLTs offer a platform that can enable trusted communication between parties that may not necessarily trust each other [7]. Additionally, DLTs have a single source of truth where the data is edited in one place only [6], so all participants have the same copy the ledger. Finally, most DLTs are decentralized geographically but are owned and maintained by a central authority [5].

Blockchains and DAGs are both types of DLTs, each with their own unique characteristics [8]. A blockchain is a data structure in which state changes, which are called transactions, are grouped together into larger structures called blocks [9]. Each block is created using information about the previous block, which links the two blocks together. This ensures that all the data is immutable; if the information in one block is changed, it will not match what is stored in the block immediately after it, and the discrepancy will be easily detected [10]. A DAG is a data structure that is similar to a blockchain, but with some key differences. While blockchains have a single chain of blocks, DAGs allow for several chains to exist and connect simultaneously. To achieve this behavior, individual state changes are added to the chains as they are validated, as opposed to grouping state changes together into blocks and validating the block as a whole. Because state changes are added as they are validated and transactions are validated by individual nodes, DAGs have a tree-like structure in which each state change has an edge connecting it to another state change that validates it. Despite their differences, blockchains and DAGs both rely on consensus algorithms to ensure that all nodes agree on the state of the system [11].

Several classification schemes for both blockchains and consensus algorithms have been proposed before [12]–[15]. Our classification scheme is designed for consensus algorithms applied in broad range of distributed systems including, but not limited to, blockchain technology and distributed ledger technology. Furthermore, our survey includes a list of recently proposed consensus algorithms like LibraBFT [16], Hotstuff [17] and HashGraph [18]. We acknowledge the related work we came across, studied, and built upon in what follows.

Sankar et al. outlined a classification in [12] for blockchains based on the degree of centralization. Nguyen and Kim proposed a binary classification [13] in which blockchain consensus algorithms are either proof-based where a miner prove, either through effort or chance, that they deserve to mine the next block. Or, vote-based where the nodes communicate with each other before adding the proposed block to the blockchain. Wang et al. [14] focused on consensus protocols used in blockchains that are open for anyone to join, categorizing them as BFT-based protocols, Nakamoto protocols, and virtual mining or hybrid protocols based on the incentive and strategies provided by each algorithm. Cachin and Vukolic categorized blockchains in [15] to be permissioned or permissionless based on whether participation in the network is open to the public or to only a specific list of individuals. They also distinguish consensus algorithms as crash tolerant or Byzantine fault tolerant depending on the types of adverse behaviors the system can tolerate. Sankar et al. [19] and Wahab and Memood [20] survey consensus algorithms, but do not attempt to classify them.

III. State-Change-Based Classification

As discussed in Section II, there are already several ways of classifying both blockchains and consensus algorithms. However, there evidently is not a classification scheme that can be used to categorize consensus algorithms of blockchains as well as other kinds of distributed systems based on how the order of state changes is determined. We outline two categories, leader-based consensus and voting-based consensus, that can be used to describe and compare consensus algorithms from a variety of distributed systems.

A. Leader-Based Consensus

In leader-based consensus algorithms, one node is chosen to be the leader. The leader is responsible for correctly ordering the state changes, broadcasting that order all other nodes, and committing those state changes to the data structure. Though other nodes may validate the order that the leader provides, only the leader can change the order and number of state changes being committed. The leader can be selected in a

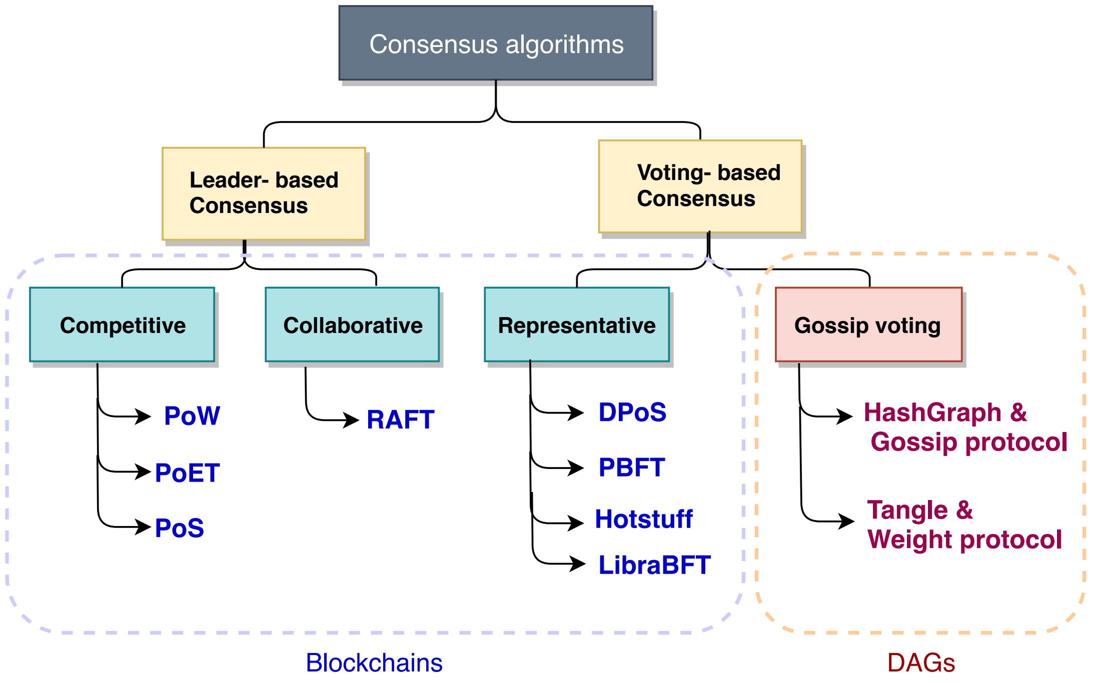

Fig. 1. Classification of Consensus Algorithms

variety of ways, but there is always at most one leader at any given time. Leader-based consensus algorithms are further classified into:

1) Competitive Leader Selection: In competitive leader selection consensus algorithms, nodes compete for the right to become the next leader, who is responsible for compiling, ordering, and committing the next group of state changes, which usually provides a monetary reward. Typically, there is a task that each node is trying to complete, and the first node to complete the task is selected as the next leader. Regardless of if completing the task requires effort or luck, each node does whatever it can to increase its chances of completing the task and being selected as leader. A few of the common algorithms that fit this definition are:

Proof of Work (PoW): PoW is the most prevalent consensus algorithm used by blockchains presently, mostly due to its use by Bitcoin [21]. Invented in 1993 by Cynthia Dwork and Moni Naor [22], PoW's simple design and rigorous security makes it an attractive solution for the consensus problem that all blockchains must solve. In PoW, nodes must generate a piece of data which satisfies given requirements that is computationally difficult to produce and easy to verify. Most PoW algorithms implement this by creating difficult puzzles, which can only be solved through brute force [21]. For example, Bitcoin's PoW algorithm, called Hashcash, requires each node to concatenate a given string with a number called a nonce. The node then hashes the resulting string and compares it to the current difficulty, which is the number that the hash must be smaller than in order to count as a correct answer. If the resulting hash is smaller than the difficulty, the node has found a correct answer. Otherwise, the node increments their nonce and repeats the process. Once a node finds a correct answer, they create a new block containing the nonce they used to calculate the answer, add the block to their copy of the blockchain, and broadcast their updated blockchain to all other nodes in the network. The other nodes see that a new block has been added, and add the new block to their copy of the blockchain once they calculate the hash using the winning nonce and verify that the resulting hash is lower than the difficulty. Every node that is currently mining Bitcoin is competing to find a nonce to concatenate to the hash of the previous block in the blockchain that hashes to a value lower than the current difficulty. Therefore, once a new block has been added, all mining nodes must restart the process of looking for a correct nonce because the hash they are concatenating the nonce with has changed [21].

Proof of Elapsed Time (PoET): One of the main criticisms of PoW is that, due to the difficulty of solving the puzzle, it consumes a lot of power without generating anything useful [23]. To address this issue, Intel created PoET [24] as an alternative to PoW that still selects its leaders competitively but does not waste large amounts of power to do so. Instead of solving a difficult puzzle, nodes in PoET are each given a random amount of time they must wait before creating a new block. Therefore, the node that is assigned the smallest amount of time will be allowed to create the next block and broadcast it to all other nodes in the network. To ensure that all random numbers are assigned fairly, Intel requires that each node has specialized hardware called the Software Guard Extensions (SGX). This hardware component ensures that the algorithm is running with trusted code, and can verify that the values it generates come from a protected environment. By using verifiably random numbers, PoET ensures that each node has an equal chance of being selected as the next leader without requiring the nodes to waste energy generating solutions to puzzles. Because each node has an equal chance of being the next leader, there is nothing that any node can do to increase its chances of being selected, which differs from the meritocracy of PoW networks [24].

Proof of Stake (PoS): As blockchain technology evolved following the release of Bitcoin, it became obvious that PoW's scalability limitations and power consumption were too large for most applications [25]. To address these issues, Sunny King created PoS and implemented it in Peercoin [26], and PoS has since emerged as one of the main alternatives to PoW. In PoS, a node's likelihood to be selected to create the next block is proportional to the amount of the blockchain's cryptocurrency they own. This eliminates the need for nodes to complete complex puzzles, so PoS is more scalable and eco-friendly than PoW. Nodes in PoS algorithms compete economically rather than computationally, which raises its own set of problems that must be addressed to prevent malicious behavior. For example, certain safeguards must be put in place to ensure that the blockchain is not dominated by a small number of wealthy nodes [26]. Peercoin prevents this by basing a node's stake on their coin age, a number that considers both the amount of cryptocurrency the node has and the amount of time that node has owned the cryptocurrency. Coin age resets once a node is selected to create a block, which prevents wealthy nodes from dominating the block creation process. Additionally, while nodes may increase their likelihood of being selected by owning more coins, the leader selection has a random element to ensure that the selection process is fair [27].

2) Collaborative Leader Selection: In collaborative leader selection consensus algorithms, nodes collaborate to elect their leader. The leader then receives all state changes and broadcasts them to the other nodes in the system. The other nodes, called followers, replicate the state changes broadcast by the leader and store the result in their individual copy of the data structure. The following is an example of a collaborative leader selection algorithm:

Reliable, Replicated, Redundant, And Fault-Tolerant (RAFT): Since its creation by Leslie Lamport in 1998, the Paxos algorithm [28] has been the consensus algorithm that all others look to for inspiration. However, it is difficult to understand and implement, which led Diego Ongaro and John Ousterhout to develop RAFT as an alternative to Paxos that is simple and practical [29]. In RAFT, one node is the leader and the rest are followers, unless a new leader is in the process of being elected. The leader has finite amount of time they are allowed to lead, which is called the term. Each node has its own individual set of all of the state changes the system has received, which is called their log. When a new state change is received, RAFT collapses consensus into two phases: log propagation and leader election. Leader election occurs if there is no current leader, the leader is unavailable, or the current leader's term has ended. Each node is assigned a random amount of time they must wait. The first node to wait their assigned amount of time becomes a candidate and votes for itself. That node then asks all other nodes to vote for it, which they do if they have not already cast a vote for the current term and the candidate's log is either matches or is longer than their own. Once one node receives a vote from a majority of other nodes, that node is elected as the leader for the next term. This approach guarantees that each node has an equal chance of being elected as the next leader. Once a leader has been elected, the log propagation phase begins. During this phase, the leader receives any state changes in the system, adds the new state changes to their log, and broadcasts their log to the followers. The followers then replicate the leader's log to their own log. Once a majority of the followers responds confirming they have replicated the leader's log, the leader commits the state changes and notifies the person who requested the state change. If there is a discrepancy between the leader's log and a follower's log, the follower will send a rejection message. The leader must then go through each previous entry of their log and broadcast it to the followers until the discrepancy is resolved. Once the leader's term ends, the leader election phase begins and the process repeats [29].

B. Voting-Based Consensus

In voting-based consensus algorithms, nodes vote for state changes or blocks that they think are valid directly, rather than voting for leaders or competing for the right to determine the order of the state changes. Therefore, while the actual creation of the block or state change may be done by one node, the order, contents, and validity of the block or state change is decided by multiple nodes. This can happen synchronously, as with most blockchains, or asynchronously, as seen in DAGs.

In pure voting based algorithms, each node communicates its vote to all other nodes in the network to reach consensus on one state change. Therefore, every round of voting requires $O(n)$ messages where n is the number of nodes in the system. Because each node must acknowledge all votes received, a single round of voting requires the communication of $O(n^2)$ messages. When reaching consensus on something that requires multiple rounds of voting such as the ordering of transactions, the amount of messages required to be sent throughout the network causes the system to be inefficient. Additionally, the number of required messages grows exponentially as the number of nodes in the system increases, making pure voting based algorithms poor at scaling. For these reasons, pure voting is never used as a consensus algorithm. However, voting can be used in conjunction with other methods of reaching consensus to greatly increase its scalability [18]. The following are two categories of consensus algorithms that use voting to ultimately agree on system state changes, but use additional mechanisms to combat the major issues associated with pure voting based algorithms:

1) Representative Voting: For the reasons listed above, voting based consensus algorithms that require each node to vote on the validity of every block are not typically practical and scalable in large distributed environments. However, a group of representatives may be elected as representatives responsible for proposing blocks. To maintain a certain degree of decentralization, the following conditions must apply: 1) All nodes must be eligible to be elected as representatives 2) All active nodes must participate in the election procedure 3) The election procedure must be conducted periodically to avoid centralization. Based on this definition, we identify the following well-known consensus algorithms that fit this category.

Delegated Proof of Stake (DPoS): The Delegated Proof of Stake algorithm was proposed and developed by Dan Larimer, who implemented it in the EOS blockchain [30]. Slightly different versions of DPoS are also implemented in BitShares [31] and Steemit [32]. It is variant of the Proof of Stake (PoS) consensus algorithm, in which participants stake an amount of cryptocurrency in order to qualify as a candidate to produce the next block.

The DPoS consensus algorithm creates a technological democracy among [30] the participating nodes by creating a group of block producers [32], who are designated with the authority to propose new blocks. Users of the underlying cryptocurrency with an account can register as voters [33] to become stakeholders and vote for desired block producers. The algorithm functions in two phases: election of block producers and scheduling production of blocks [32]. The election process is straightforward. Registered voters may login to the voting portal and vote for a specific number of nodes, which is typically higher than the fixed number of block producers. For example, EOS has 21 block producers at any given time and stakeholders may vote for as many as 30 nodes in a session [33]. The 21 nodes with the highest number of votes are elected as block producers. They are then responsible for scheduling the production of blocks and receive a reward for their services.

Since DPoS requires staking of underlying cryptocurrency and rewards block producers for their services and good behavior, it is most suitable for a cryptocurrency based public network. DPoS is Byzantine Fault Tolerant and can withstand upto $1/3(N) - 1$ of malicious nodes, where N is the total number of active nodes. Forks are possible with DPoS and the longest chain rule applies. It implements Transaction as Proof of Stake (TaPoS) [30] where all transactions include a hash of most recent block and if the newly proposed block is considered invalid if the hash of recent block doesn't exist on the longest chain. It is also resilient against attacks like Double Spending and Network Fragmentation, and it can effectively resolve troublesome scenarios like lack of quorum among block producers and corruption of majority of block producers.

Practical Byzantine Fault Tolerance (PBFT): Lamport et al. [34] proved that $2m + 1$ properly functioning processors are required to reach consensus in a system with m ill-functioning processors. Liskov et. al. [35] designed the PBFT mechanism to maintain consensus in a distributed computing system functioning asynchronously. PBFT builds on Lamport's work by assuming that at least $2/3 + 1$ nodes are required to act non-maliciously for all the nodes to effectively reach an agreement.

In PBFT implemented distributed systems, nodes do not require a leader in the network and they can communicate with each other. However, one node is considered as the primary and the others are regarded as replicas. This property ensures that there are no forks in the global state of ledger. PBFT implements the state machine approach where transactions lead to state transitions, a client must wait for $f + 1$ nodes to reply with the same resulting transition to regard a transaction as successful. In case the primary node fails to respond, nodes communicate with each other to decide which node will replace as primary and reach an agreement asynchronously [35].

The non-changing position of the primary node makes the PBFT algorithm extremely fast in comparison with other consensus algorithms. PBFT also requires $O(n^k)$ [36] communication between nodes where n is the messages and k is the number of nodes. This increases exponentially with addition of new nodes to the network which is not scalable across large network of nodes. Furthermore, PBFT is susceptible to sybil attacks [36] where one entity controls many different nodes with different identities. The possibility of carrying out a Sybil attack becomes almost negligible in large distributed systems with hundreds of nodes.

Hotstuff: Yin et al. proposed Hotstuff [17], a leader-based PBFT variant consensus algorithm for distributed systems. Hotstuff was the first partially synchronous BFT replication protocol, so it exhibits a set of unique properties.

There are 2 basic changes to the traditional PBFT implemented by Hotstuff [37]. First, the communication network topology is transformed from a mesh network in PBFT to a star network in Hotstuff. Each node no longer multicasts messages to all nodes. Instead, they all communicate with each other through the leader. This significantly reduces communication complexity from $O(n^k)$ to $O(n)$, where n is the number of participating nodes. Secondly, Hotstuff combines the view change phase with the normal operation, therefore decreasing the overhead caused by view change phase. Nonetheless, this leads to an extra phase of confirmation, expanding the two-phase PBFT to three phases in Hotstuff. It also introduces a new cryptographic primitive called threshold signature [17], which is used to reduce the number of signatures used in achieving consensus.

Hotstuff overcomes most of the limitations experienced in traditional PBFT and is significantly more decentralized, given that the view change occurs periodically. Like in PBFT, there is no fork possibility in Hotstuff, but it is still susceptible to Sybil attacks. It has not yet been tested on a public network and would require strong integration with cryptocurrency to do so. It may also be implemented in combination with Proof-of-Stake with critical punishment mechanisms for malicious behavior. Otherwise, we believe, Hotstuff is a scalable and decentralized consensus algorithm that is secure only in a partial trust environment like a permissioned/ private network.

LibraBFT: Implemented in the Libra blockchain [38] project, LibraBFT [16] is a PBFT variant consensus algorithm that builds on the above mentioned Hotstuff consensus algorithm. It supports state machine replication based consensus in a "partially synchronous distributed systems", defined by Dwork, Lynch and Stockmeyer [39].

LibraBFT extends the concept of Hotstuff algorithm by designing the protocol to be more resilient to non-determinism bugs. This is achieved by making validators sign the resulting state of each block collectively [38]. Furthermore, a pacemaker [16] is designed that explicitly emits timeouts which enables validators to proceed to the next round without the requirement of a synchronized clock. The most important change was to implement an unpredictable leader election mechanism where the proposer of the latest block uses a verifiable random function to elect the leader of the next round. This technique limits the window of possibility for a Denial-of-Service attack and randomizes the leader election process. LibraBFT replaces the threshold signature concept with aggregate signatures to reduce complexity and provide incentives the validators.

Like in PBFT and Hotstuff, LibraBFT is Byzantine fault tolerant and secure only in a partial trust environment where nodes are authorized to propose and validate blocks. LibraBFT is resistant to Sybil attacks since nodes must be authorized by an off-band process to be able to validate blocks. Hotstuff consensus also avoids forking issue, which is also true for LibraBFT due to core protocol adaption. However, LibraBFT provides incentive to validators, a unique property that is not implemented in Hotstuff or core PBFT. There is also the 3-chain commit rule [16] for persistence and replication of data blocks. The block in round n is committed only if it is confirmed by 2 more blocks at rounds $n + 1$ & $n + 2$ [38]. LibraBFT exhibits similar high latency and high throughput transactions but sacrifices decentralization by enforcing authorized validators.

2) Gossip Voting: Unlike pure voting and representative voting, gossip voting is a voting mechanism with little communication overhead in the network. This type of voting does not require leaders, followers, or competitions to reach consensus. Instead, it uses the gossip protocol, where every node randomly picks a neighbor node and sends all the information it knows to this node. This process repeats until, eventually, all the nodes in the network will have the same information. Additionally, every node will know where every other node got their information from. For example, imagine there are three nodes: Node 1, Node 2, and Node 3. If Node 1 and Node 2 are neighbors and Node 2 and Node 3 are neighbors, Node 3 will eventually know everything that Node 1 knows as well as the fact that Node 2 got their information from Node 1. This historical record of the spread of information ensures a fair ordering of transactions in the same way a blockchain does without requiring consensus to be reached synchronously. Thus, there is no need to have an explicit vote in order to reach consensus. Instead, each node votes that a transaction is valid once it sends the information about the transaction and where it came from to another node. Therefore, the number of votes a transaction has is the number of times it has been sent from one node to another. all other nodes and it needs to confirm the received messages. In the gossip voting nodes don't have to send any messages, nodes can deduce what other nodes voted because they already send what they know.

Due to gossip voting's asynchronous and non-linear design, it is not suited for use with a blockchain data structure. Instead, this type of voting is done using DAGs. DAGs allow several different chains of transactions to exist simultaneously, which works well with gossip voting's tree-like structure. Despite their differences, though, both DAGs and blockchains are immutable, so there are not many security sacrifices associated with using a DAG as opposed to using a blockchain [18], [40]. The following are examples of gossip voting consensus algorithms, all of which are implemented using DAGs:
Swirlds Hashgraph: Developed by Leemon Baird in 2016 [18], Hashgraph was one of the first gossip voting algorithms to be implemented using a DAG. In Hashgraph, nodes send transactions randomly to other nodes in the network. Every time a node receives a transaction from another node, they create an event that records everything in the transaction. Additionally, the hash of both nodes' last event is recorded in the new event along with any new information that the receiving node wants to include along with the receiving node's digital signature and the timestamp. Over time, this builds a ledger of events in which events are linked by the hashes included when the event was created. Each node has a copy of this ledger, which may differ slightly from other nodes' ledgers if new gossip has not reached it, but after a certain point all the ledgers will be identical. Because of this, consensus can be determined by any node in the network; all nodes know almost everything that all other nodes know, so they how a node would vote if they were asked. This allows each node to be able to simulate elections through what is called virtual voting. It is called this because there is no communication happening during the voting process; it is all simulated by the individual node. This drastically reduces latency and communication overhead, which is one of Hashgraph's greatest strengths [18].
Tangle: IOTA [40] uses a similar consensus approach to Hashgraph with its gossip voting protocol Tangle [41]. IOTA developers implemented Tangle in 2015 to provide cryptocurrency for the internet of things systems without requiring competition or high overhead. In IOTA, transactions are called sites and links between two transactions are called edges. A transaction in Tangle must have edges to at least two other transactions in order to considered confirmed; if they have less than two edges, they are unconfirmed transactions, also called tips of the Tangle. When a new transaction is added, a Markov chain Monte Carlo algorithm is used to randomly select two or more tips of the Tangle to attach the new transaction to [41]. This means that every added transaction confirms two other transactions, which makes IOTA's Tangle scalable; the network does not slow down when there are a lot of new transactions. Tangle currently uses the help of PoW to reach consensus. When a node creates a new transaction, it does a small amount of PoW to determine the transaction's weight. When the transaction is added and becomes a site, its weight

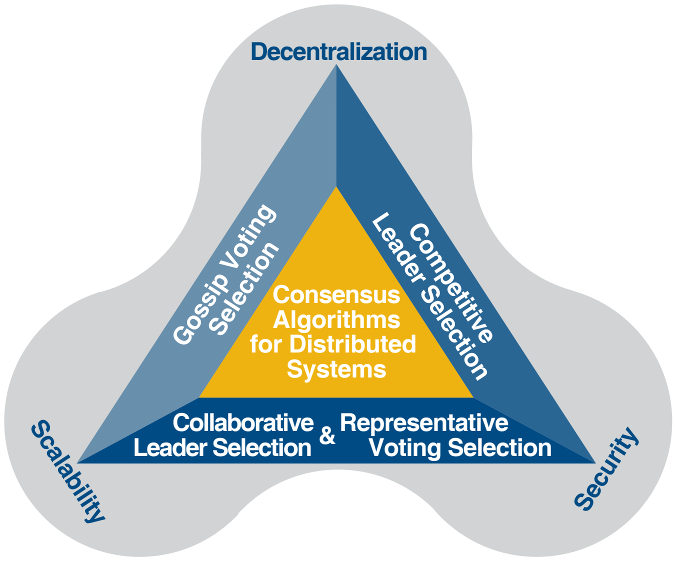
Decentralization
Gossip Voting Selection
Competitive Leader Selection
Consensus Algorithms for Distributed Systems
Collaborative Leader Selection
&
Representative Voting Selection
Scalability
Security

Fig. 2. The Consensus Classification Conflated with Buterin's 'Scalability Trilemma' illustrating the Choose-2 Tradeoff

represents how trustworthy it is. As the site gets confirmed by more sites, it's the confirming site's weight is added to the confirmed site's weight. Thus, the more a site is confirmed, the more trustworthy it will be [40].

IV. Determination of Each Category's Priorities

According to Vitalik Buterin's "Scalability Trilemma" [42], there are three qualities that every distributed system should strive to achieve: scalability, security, and decentralization. For a distributed system to be scalable, it must easily expand as the network grows without requiring excessive amounts of storage, even as the number of active nodes and transactions gets large. A distributed system is secure if it is fair, consistent, and available even under adverse conditions. Finally, there are two aspects of decentralization that a distributed system must satisfy in order to be fully decentralized [43]. First, it must consist of many individual nodes, and be able to handle some number of them being down at one time. Second, these nodes must not be owned by a relatively small number of individuals or organizations.

Although distributed systems are typically designed with these qualities in mind, this is a choose-two tradeoff. Thus, every consensus algorithm is designed to prioritize two of the three qualities. Figure 2 illustrates which two qualities are prioritized by each of our categories. Choosing which two qualities to prioritize is an important decision that depends on the specific application that the consensus algorithm will be used for. The benefits, drawbacks, and use cases for each of the categories in Section 3, relative to the three qualities listed above, are as follows:

A. Scalable and Secure

Representative Voting-Based Algorithms: As described in Sec-III-B1, the representative voting based consensus requires a subset of the total participating nodes to be delegated or authorized to propose blocks, therefore, reducing the computation and communication complexity to $O(r)$ involved in voting for new blocks, where $r \subsetneq N$ (Total number of participating nodes). This mechanism is more scalable than allowing all N participating nodes to be involved in block proposal procedure.

The Representative Voting-based algorithms are comparatively secure because the representatives are either delegated by a voting procedure (as in DPoS), or they are authorized by an off-band process (as in LibraBFT). As long as the delegation or authorization process remains secure, the representatives are trustworthy and non-malicious. Good behavior is also incentivized in the case of LibraBFT, which also means that there is punishment if malicious behaviour is detected. This will keep the validators motivated to order transactions appropriately and reach consensus as quickly as possible.

By selecting a subset of nodes to be authorized or delegated, naturally, this type of consensus algorithm trades decentralization for security and scalability. And as observed in such consensus algorithms, this results in high throughput and low latency of transactions.

Use Cases: Such consensus algorithms are best if used in a partial trust environment, which is also recommended by their authors. Partial trust environment may be implemented by access control and centralized authorization of validators, as observed in LibraBFT. Decentralized voting based authorization may also result in a partial trust environment, like in DPoS, this voting procedure, however, must be distinguishable from on-chain voting conducted on block proposals. Representative voting-based consensus algorithms are the best fit for distributed computing systems where high scalability can be prioritized over maximum decentralization.

Collaborative Leader Selection Algorithms: Because competition occurs relatively infrequently in collaborative leader selection algorithms, and because all followers must replicate any information broadcast by the leader, these algorithms are not under the risk of majority attacks or forking. Furthermore, consistency is guaranteed for the same reason; if there is ever an inconsistency, the leader will delete information and broadcast the updated set of information to be replicated by the followers until the inconsistency is resolved. Because a new leader is quickly elected if the leader is unavailable, collaborative leader selection algorithms are crash fault tolerant. This quality guarantees that the system will not fail as long as a majority of the nodes are available. In the event of an unavailable leader, all followers have an equal chance of being selected as the new leader, which guarantees that the algorithm is fair. Together, the availability, consistency, and fairness that characterize collaborative leader selection algorithms make them secure.

Because all followers must have identical sets of information, it can be assumed that information broadcast by the leader was successfully replicated in all nodes once a majority of followers confirm their replication. This contributes greatly to the scalability of collaborative leader selection algorithms; there is no need to wait for the response of every follower, so consensus can be reached even if some followers crash or experience network latency. While the number of responses needed to consider the replication successful will increase as the number of nodes in the network increases, the performance of the system will see negligible penalties as long as over half of all nodes are acting correctly due to the low overhead that follower messages incur. It is possible for the information set stored by each node to get large if the system runs for a long time or if there are a large number of transactions. This has the potential to limit the system's scalability if the information set becomes too large to feasibly store. However, mechanisms such as snapshotting, which periodically saves the entire information set to stable storage and clears each nodes information set, exist in all common collaborative leader selection algorithms to prevent such scalability limitations from occurring [29].

It is important to note that these security and scalability guarantees are contingent on all of the nodes acting in a non-Byzantine way, meaning they must all be trusted to not act maliciously. Collaborative leader selection algorithms are crash tolerant, so they can handle network delays and server crashes, but they are not Byzantine fault tolerant, which means they cannot handle malicious behavior. Because these algorithms require trust to function properly, there must be some central authority that determines which nodes can be trusted. Thus, collaborative leader selection algorithms are not decentralized. Instead, these algorithms sacrifice decentralization to prioritize security and scalability.

Use Cases: Because the leader is always assumed to be correct and any node can become the leader, collaborative leader selection algorithms require the higher amount of trust than any other type of algorithm in order to function properly. Therefore, this type of algorithm should only be used in private, permissioned systems where the identity and trustworthiness of each node can be verified. For closed systems where the sole purpose is to securely and quickly store information, the relative simplicity and low resource cost of these algorithms make them an attractive option. Similarly, systems that are trusted but unstable would benefit from the crash fault tolerance and fast leader election that these algorithms provide.

B. Secure and Decentralized

Competitive Leader Selection Algorithms: Because any node can compete to be selected as the next leader, algorithms that feature competitive leader selection are considered to be highly decentralized. There is no need for a central authority because it is easy for any node to prove that they were selected to create the next block. This ensures that every node has a chance to compete to become the leader and earn a reward, and that the network will not necessarily crash if a small number of nodes fail. However, participants will always do whatever they can to increase their likelihood of winning the competition so they can earn more reward. As a blockchain grows in popularity, certain users will invest more to ensure they have the highest chance of being selected. This can result in a barrier of entry that is too high for the average user, which means the blockchain has the possibility to become more centralized as it becomes more popular [44].

To ensure security, most competitive leader selection algorithms have a fixed block creation rate, meaning the difficulty of winning the competition and being elected as the leader is proportional to the number of nodes competing. This prevents mass cryptocurrency inflation as well as majority attacks like the 51% attack [45]. As long as the network remains synchronized, competitive leader selection algorithms are 50% fault tolerant, meaning the security of the blockchain will remain intact as long as less than half of all nodes are acting maliciously [46].

As stated above, for a blockchain to be scalable, it must be able to handle a high number of transactions in a small amount of time. This means that, as more nodes enter the network, blocks need to be either created more quickly or become larger in size. However, neither of these are easily done in competitive leader selection algorithms. Competitive leader selection algorithms must keep the block creation rate fixed in order to prevent majority attacks and other malicious behavior. If the block creation rate is fixed, the block size will have to increase as the number of nodes increases. This is a problem because each node must have their own copy of the blockchain so they can mine the next block and validate blocks broadcast by other nodes. As the block size grows, the amount of storage required to store the entire blockchain grows exponentially, making it impossible for the average user to compete to become the next block creator and compromising decentralization. Therefore, it is impossible for blockchains using competitive leader selection algorithms to be scalable without compromising security or decentralization.

Use Cases: Because of their emphasis on decentralization and security, competitive leader selection algorithms are the best choice in a trustless environment such as a public blockchain. The qualities listed above ensure that the network is fair for all nodes, and that each node has the ability to participate and earn rewards. Similarly, competitive leader selection algorithms are the best choice for any blockchain with an associated cryptocurrency.

C. Decentralized and Scalable

Gossip Voting-Based Algorithms: There are several characteristics of gossip voting algorithms that make them far more scalable than other consensus algorithms. The use of DAGs instead of blockchains allows transactions to be confirmed quickly and asynchronously. Similarly, because transactions are linked to other transactions, gossip voting algorithms can predict how any given node will vote, so voting can happen locally [18], [40]. This greatly reduces communication overhead and network latency. Because the transactions in these protocols are small and not grouped together, there is no risk of storage requirements limiting scalability. Decentralization is also maintained because transactions are verified by other nodes in the network. There is no need for a central authority because, as more transactions are added, the system becomes inherently more secure. Some gossip voting algorithms have used centralized nodes to bootstrap their networks [41], but most aim to have a fully decentralized system once it becomes large enough. In short, the more participants a gossip voting algorithm has, the faster and more secure it will be. This makes systems using gossip voting more scalable than the average blockchain while maintaining decentralization [18], [40].

However, systems that uses the gossip voting are susceptible to Sybil attacks [36], in which one entity controls many different nodes with different identities. In gossip voting algorithms, there is not yet a solution to this problem without compromising decentralization or scalability.

Use Cases: Because the communications between the nodes is fast and light overhead then this category is applicable for the applications that has a private permissioned network where the nodes need to communicate with each other in the fastest and cheapest way possible, and maintain a ledger for keeping track of their transactions. Systems with limited storage capacity and processing power, such as IoT systems, would benefit from the use of this type of consensus algorithm.

V. Conclusion

In this paper, we described a novel classification for common consensus algorithms based on how they decide the order of system state changes. Figure 1 offered a classification of consensus algorithms for distributed ledgers while Figure 2 showed a prioritization (tradeoffs) of the consensus algorithms. We determined the extent to which each category prioritizes scalability, decentralization, and security. We found that, as is common with other engineering design scenarios, that a choose-two tradeoff results among these three concerns. And, based on this taxonomy of tradeoffs, we were able to discern the types of consensus algorithms that work well within the application area(s) for a given distributed system. We found that a dichotomy of algorithms between leader-based and voting-based consensus algorithms emerged from this taxonomy. We presented certain applications under this classification scheme including several different categories of distributed ledgers such as blockchains and Directed Acyclic Graphs (DAGs). Overall, this classification scheme provides a useful basis for determining the appropriate consensus algorithms for a variety of distributed applications.

Acknowledgments

Research reported in this publication was supported by the 2020 Center of Excellence for Applied Computational Science and Engineering grant competition (CEACSE). It was also supported in part by the National Science Foundation under grants nos. 1812404 and 1821926. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

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

[2] Florian Idelberger, Guido Governatori, Régis Riveret, and Giovanni Sartor. Evaluation of logic-based smart contracts for blockchain systems. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, pages 167–183. Springer, 2016.

[3] Kenji Saito and Hiroyuki Yamada. What's so different about blockchain?—blockchain is a probabilistic state machine. In *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 168–175. IEEE, 2016.

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

[5] Anton Badev, Maria Baird, Timothy Brezinski, Clinton Chen, Max Ellithorpe, Linda Fahy, Vanessa Kargenian, Kimberley Liao, Brendan Malone, Jeffrey Marquardt, David Mills, Wendy Ng, Anjana Ravi, and Kathy Wang. Distributed ledger technology in payments, clearing, and settlement. *Finance and Economics Discussion Series*, 2016, 12 2016.

[6] Roger Maull, Phil Godsiff, Catherine Mulligan, Alan Brown, and Beth Kewell. Distributed ledger technology: Applications and implications. *Strategic Change*, 26(5):481–489, 2017.

[7] Phoebus L Athanassiou. *Digital Innovation in Financial Services: Legal Challenges and Regulatory Policy Issues*. Kluwer Law International BV, 2016.

[8] Nikos Fotiou and George C Polyzos. Decentralized name-based security for content distribution using blockchains. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 415–420. IEEE, 2016.

[9] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59(3):183–187, Jun 2017.

[10] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 557–564. IEEE, 2017.

[11] Federico Matteo Benčić and Ivana Podnar Žarko. Distributed ledger technology: Blockchain compared to directed acyclic graph. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1569–1570. IEEE, 2018.

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

[13] Giang-Truong Nguyen and Kyungbaek Kim. A survey about consensus algorithms used in blockchain. *Journal of Information processing systems*, 14(1), 2018.

[14] Wenbo Wang, Dinh Thai Hoang, Zehui Xiong, Dusit Niyato, Ping Wang, Peizhao Hu, and Yonggang Wen. A survey on consensus mechanisms and mining management in blockchain networks. *arXiv preprint arXiv:1805.02707*, pages 1–33, 2018.

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

[16] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State machine replication in the libra blockchain, 2019.

[17] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019.

[18] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep.*, 2016.

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

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

[21] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[22] Moni Dwork, Cynthiaand Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 139–147, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[23] Harald Vranken. Sustainability of bitcoin and blockchains. *Current Opinion in Environmental Sustainability*, 28:1 – 9, 2017. Sustainability governance.

[24] Intel. Poet 1.0 specification. https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html. [Online, Accessed: Nov/19/2019].

[25] Ghassan Karame. On the security and scalability of bitcoin's blockchain. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1861–1862. ACM, 2016.

[26] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

[27] Sunny King and Scott Nadal. Ppcoin: peer-to-peer crypto-currency with proof-of-stake (2012). *URL https://peercoin. net/assets/paper/peercoin-paper. pdf.[Online*, 2017.

[28] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[29] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm (extended version), 2013.

[30] Block.one. EOS.IO Technical White Paper v2. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md, 2018. [Online, Accessed: 10/27/2019].

[31] bitshares.org. Delegated proof-of-stake consensus. https://bitshares.org/technology/delegated-proof-of-stake-consensus/. [Online, Accessed: Nov/17/2019].

[32] Daniel Larimer. Dpos consensus algorithm - the missing white paper. https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper, 2017. [Online, Accessed: Nov/17/2019].

[33] EOS Authority. Eos block producer voting statistics. https://eosauthority.com/voting, 2019. [Online; Accessed: Nov/25/2019].

[34] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[35] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186, 1999.

[36] Geeksforgeeks.org. practical byzantine fault tolerance(pbft). https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/. [Online, Accessed: Nov/17/2019].

[37] The Ontology Team. Hotstuff: the consensus protocol behind facebook's librabft. https://medium.com/ontologynetwork/hotstuff-the-consensus-protocol-behind-facebooks-librabft-a5503680b151, 2019. [Online, Accessed: Nov/17/2019].

[38] Zachary Amsden et. al. The libra blockchain. https://developers.libra.org/docs/assets/papers/the-libra-blockchain.pdf, 2019. [Online, Accessed: Nov/18/2019].

[39] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[40] Roman Alexander. *IOTA - Introduction to the Tangle Technology: Everything You Need to Know About the Revolutionary Blockchain Alternative*. Independently published, 2018.

[41] Serguei Popov. The tangle. *cit. on*, page 131, 2016.

[42] Ethereum. Sharding faq. https://github.com/ethereum/wiki/wiki/Sharding-FAQ. [Online, Accessed: Nov/19/2019].

[43] Vitalik Buterin. The meaning of decentralization. https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274. [Online, Accessed: Nov/19/2019].

[44] Arthur Gervais, Ghassan O Karame, Vedran Capkun, and Srdjan Capkun. Is bitcoin a decentralized currency? *IEEE security & privacy*, 12(3):54–60, 2014.

[45] Martijn Bastiaan. Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin. In *Availab le at http://referaat. cs. utwente. nl/conference/22/paper/7473/preventingthe-51-attack-a-stochasticanalysis-oftwo-phase-proof-of-work-in-bitcoin. pdf*, 2015.

[46] Ethereum. Proof of stake faq. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ. [Online, Accessed: Nov/19/2019].