Received: 10 March 2019 | Revised: 17 July 2019 | Accepted: 4 August 2019

DOI: 10.1002/spe.2748

SPECIAL ISSUE PAPER

WILEY

Jointgraph: A DAG-based efficient consensus algorithm for consortium blockchains

Fu Xiang[1,2] | **Wang Huaimin**[1,2] | **Shi Peichang**[1,2] | **Ouyang Xue**[1] | **Zhang Xunhui**[1,2]

[1]School of Computer, National University of Defense Technology, Changsha, China

[2]Key Laboratory of Software Engineering for Complex System, Changsha, China

Correspondence
Shi Peichang, School of Computer, National University of Defense Technology, Changsha 410073, China.
Email: shipeichang@kylinos.cn

Funding information
National Key R&D Program of China, Grant/Award Number: 2016YFB1000100; National Natural Science Foundation of China, Grant/Award Number: 61772030; GF Innovative Research Program

Summary

The blockchain is a distributed ledger that records all transactions and operations in a shared manner. Public blockchains such as Bitcoin realize decentralization at the cost of mining overhead, which is not suitable for real-life scenarios requiring high throughput. Techniques such as the consortium blockchain improve efficiency through partial decentralization. However, the consensus algorithms used in the existing state-of-the-art consortium blockchains face many challenges when dealing with commercial applications. For example, the high communication overhead hinders the scalability of PBFT-based consensus algorithms even though they are efficient at small scale. Hashgraph, one of the most popular Directed Acyclic Graph-based (DAG-based) consensus algorithms, achieves good performance in scalability; however, it does not allow users' dynamic participation. To deal with these challenges, we propose Jointgraph, a Byzantine fault-tolerance consensus algorithm for consortium blockchains based on DAG. In Jointgraph, transactions are packed into events and validated by no less than 2/3 of all members. A supervisor is introduced in our design, who monitors member behaviors and improves consensus efficiency. Simulation results demonstrate that Jointgraph outperforms Hashgraph in both throughput and latency.

KEYWORDS

blockchain, byzantine fault-tolerant, consensus algorithm, DAG, supervisory

1 | INTRODUCTION

In general, current blockchain systems can be broken down into three categories: the public blockchain, the private blockchain, and the consortium blockchain.[1-3] The significant differences among these three types are the requirements toward participants: for the public blockchain, everyone can take part in the consensus process while the private blockchain only allows participants from a particular organization. The consortium blockchain is in the middle of these two: usually, a group of pre-selected users from several organizations will dominate the consensus process.[4,5] The private and consortium blockchain are regarded as not fully decentralized because only particular users can take part in managing the chain.

For public blockchains, the more nodes, the higher the security and fairness of the system, which leads to the reduced efficiency of the system. As the most representative public blockchain system, Bitcoin[6,7] can only process seven transactions per second, and Ethereum[8,9] can merely process 10-20 transactions per second. Both Bitcoin and Ethereum

encounter challenges of high transaction costs, long confirmation times, and poor scalability. According to BTC.com, in 2017, the number of transactions of Bitcoin was only about 30 million, while the real-life e-commerce systems normally require much more capacity than that. For example, on November 11, 2017, Alipay completed 1.48 billion transactions.[10] Moreover, many public blockchain projects build platforms first and then look for application scenarios, which extended the gap between their system and practical applications of public blockchains.

The consortium blockchain weighs between decentralization and efficiency. It is partially decentralized and with higher throughput. The number of participants for the consensus process is specified, and the identity of the participants is known to each other. Compared with the public blockchain, the consortium blockchain has advantages in high availability, high performance, programmable ability, and privacy protection. The consortium blockchain makes the system more efficient and cost-effective.[11,12] Moreover, it is easier to apply in real-life scenarios. In addition, one of the essential features of the consortium blockchain, compared with the public blockchain, is its support toward node access control and national security standards. As a fundamental component supporting distributed commerce, the consortium blockchain is capable of dealing with multiparty co-operation. An internal ecosystem with significantly increased efficiency can be built by adopting the forms of consortium blockchains. For example, the consortium blockchain is suitable for the commercial transactions between organizations, similar to the transfer and payment between banks. There are several representative consortium blockchain systems such as Hyperledger[13] and the R3 (www.r3.com). Hyperledger is an open source project launched by the Linux Foundation in 2015, attracting Huawei and Tencent Cloud, Baidu Finance, Samsung, IBM, Intel, Fujitsu, Cisco, Redhat, Oracle, and many other companies. At present, there are more than 200 member companies in Hyperledger. The R3 is established in September 2015, which is an enterprise software firm working with a network of over 200 financial institutions, regulators, trade associations, professional services firms, and technology companies to develop on Corda,[14] the blockchain platform designed specifically for businesses.[15]

Despite the popularity, the consensus algorithms used in the existing state-of-the-art consortium blockchains are still far from successfully meeting all the needs of commercial applications. For example, the unaffordable communication overhead of the PBFT-based consensus algorithms restricts system performance in large-scale environment.[16-18] To solve the scalability problem, Algorand[19] uses a verifiable random function to select a committee of nodes that participate in a novel Byzantine consensus protocol called BA*. This consensus protocol requires a high quality of network connectivity. PHANTOM[20] utilizes a DAG data structure of blocks, and it is a generalization of Satoshi's Proof-of-Work (PoW)–based blockchain with faster block generation and larger blocks. Conflux[21] consensus protocol represents relationships between blocks as a DAG. It also operates with PoW mechanism. The block generator attempts to find solutions for PoW problems to generate blocks. Thus, both in PHANTOM and Conflux, the energy wasting PoW mechanism is stilled used. Avalanche[22] is a DAG-based PoW-free BFT protocol that does not incur quadratic message cost and can work without precise membership knowledge. However, for every single transaction in a node, votes should be sent through the network. Meanwhile, in Hashgraph,[23] one of the most popular DAG-based consensus algorithms, the consensus is determined by the DAG data structure. Once a node holds the DAG, it can make a judgment on whether an event has reached consensus, which means it does not need any more network communication for the consensus. However, there are some strong assumptions in Hashgraph, for example, participants cannot be replaced flexibly.

Furthermore, the lack of supervisors makes existing blockchain applications face some thorny issues. For example, Bitcoin anonymity can hide the identity of the user; WannaCry is a ransomware cryptoworm, which targeted computers running the Microsoft Windows operating system by encrypting data and demanding ransom payments in the Bitcoin.[24] Furthermore, Bitcoin is used for black market transactions such as buying drugs and arms. However, for the consortium blockchain, the participants' access mechanism creates conditions for adding supervisors. A supervisor monitors malicious behaviors. Moreover, he is able to improve the consensus efficiency.

In this paper, we propose Jointgraph, a mechanism for a supervisor to join the consortium blockchain to improve both security and efficiency. The supervisor node in the proposed design is different from the traditional fully centralized center. It is only a weakly centralized node that is responsible for collecting votes, taking snapshots, and replacing members. At the same time, a DAG-based consensus algorithm is adopted in our design. The throughput of the consensus algorithm increases a lot with the help of the supervisor, and the consensus delay is reduced as well, compared with Hashgraph. The security of Jointgraph relies on the principle that the number of malicious nodes is less than $n/3$ (n is the number of all nodes), which is a common assumption adopted by Byzantine fault-tolerant systems.

The major contributions of this paper are as follows:

- **Consensus algorithm:** We present a novel, efficient, and scalable DAG-based Byzantine tolerant[25] consensus algorithm, Jointgraph, for the consortium blockchain to build a partial order for the transactions.

- **Supervision protocol:** We set a supervisor, which is a unique participant in Jointgraph. The supervisor can improve the consensus efficiency, and he is mainly responsible for the members' behavior monitoring and replacing malicious members. To release memory, the supervisor takes snapshots of the system state periodically.
- **The results are validated via simulation:** We present several sets of simulation experiments by comparing the throughput and the consensus latency of the Hashgraph and Jointgraph in different scenarios (different number of nodes, different number of failure nodes, and different network latency). Results show that Jointgraph outperforms Hashgraph in both throughput and latency. Moreover, Jointgraph has better scalability than Hashgraph.

The rest of this paper is organized as follows. Section 2 illustrates the overall system model of Jointgraph, details including gossip for communication, how to reach consensus, how to take a snapshot, and when to replace ordinary nodes. The simulations and results are discussed in Section 3. Section 4 discusses the consistency, liveness, and security of Jointgraph. Section 5 presents an overview of the related work, and Section 6 concludes the paper.

2 | SYSTEM MODEL

The Jointgraph consensus is a Byzantine fault-tolerant algorithm, which is used for the nodes to reach an agreement of their transactions. In Jointgraph consensus, transactions are packaged into events. The events are sent through a gossip protocol, which means anyone could send events to a random node. There are two types of the nodes, namely the ordinary node and the supervisory node. The ordinary node sends events and is responsible for the validation of other nodes' events. They vote "YES" if the event passes the verification by adding the event to their Jointgraph. The supervisory node is responsible for the collection of votes in the consensus process and determines the finality of events. Every member has a copy of the Jointgraph, so one can calculate what vote the others would have sent to him. Members do not need to send their votes, so zero bandwidth is used, beyond simply gossiping the events. The event finality is formally determined when the supervisory node collected more than 2/3 of the nodes' "YES" votes. At the same time, it supervises the behaviors of the ordinary nodes, identifies the malicious behavior, and is responsible for the system snapshot generation, ordinary nodes replacement, etc.

2.1 | Gossip for communication

A node will send all the events it has so far to a random node. Every node repeats sending events to different random nodes all the time. In this way, if a single node issues an event, it will spread exponentially fast through the network until every node gets it. Figure 1 is a Jointgraph, which is actually a DAG. Every member of the community keeps a copy of it in memory. The vertexes in the Jointgraph are the events, which is a container for transactions. The event can optionally contain zero or more transactions that the sender wants to issue. Each member in the community has one column of vertexes. When member A receives events from member B, A issues a new event (event 1 in Figure 1), which is represented by a vertex in A's column, with two edges going downward to the immediately preceding gossip events by A (event 2 in Figure 1) and B (event 3 in Figure 1). There are two hashes in an event, which are the hashes of the self-parent and the

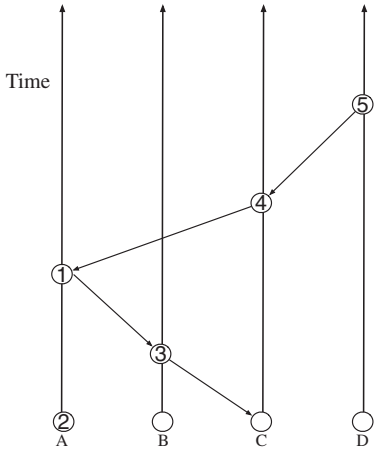

FIGURE 1 A Jointgraph

other-parent. Event 2 and event 3 are the self-parent and other-parent of event 1, respectively. The Jointgraph is the gossip history of all the members. It grows upward over time. Thus, lower vertices represent earlier events in history.

2.2 | Consensus process

The normal situation: In Jointgraph, an edge from event *A* to event *B* means event *A* has verified event *B*. Moreover, event *B* is a parent of event *A*. When a member receives an event from another member, he will verify whether the event is valid. The verification includes whether the transactions are valid, whether the signature is valid, and whether the hashes of self-parent and other-parent are valid. If the events cannot pass the verification, the receivers will dump the events. As shown in Figure 1, event 1 is the other-parent of event 4 and the ancestor of event 5; we can say that both members *C* and *D* have verified event 1. Thus, members *C* and *D* both vote "YES" for event 1. The finality of an event can be confirmed if it is verified by more than 2/3 of all the nodes (including the supervisory node), and the votes are collected by the supervisory node. In Figure 2, the dark vertexes are part of the consensus, and the light ones are not because the dark vertexes are verified by at least three (the number of all nodes is four) members, and the votes are collected by the supervisory node. As time goes by, the new coming events will verify the older ones, and every time the supervisor issues an event, some more former event may become part of the consensus. The confirmation time of an event is mainly based on the gossip frequency. The higher the frequency of the nodes sending events, the faster for an event to become part of the consensus. The consensus algorithms run on the ordinary nodes, and the supervisory node are Algorithm 1 and Algorithm 2, respectively. The consensus process in Jointgraph does not contradict that of the Hashgraph because both of them are based on the gossip protocol. Moreover, once a node holds the DAG, it can both use these two processes to determine whether an event has reached consensus or not. However, the consensus process of Hashgraph is much more complicated than that of the Jointgraph. With the help of the supervisor, Jointgraph only needs one round voting, whereas Hashgraph needs no less three rounds. If an event reaches consensus from the perspective of Hashgraph, it must also reach consensus once it is sent to the supervisory node from the perspective of Jointgraph. In most instances, the supervisory node becoming a malicious node is not considered because it is a weak centralized node that is responsible for the supervision of the system. However, it may be hacked down and stop working. In this case, the consensus process can still work with Hashgraph algorithm, which means if a node finds that the supervisory node is down, he can switch to Hashgraph process to determine the finality of events until the supervisory node is online again.

Dealing with a fork: Firstly, we define the fork: The pair of events (x, y) is a fork if x and y have the same creator, but neither is a self-ancestor of the other. If the creator sends event x to a neighbor and sends y to another neighbor, it may cause double spending[26] because the creator may spend his token twice in these two events, respectively. We call the node that generates a fork a Byzantine fault node. Once a fork is found by any node, the node will not communicate with this Byzantine fault node anymore. There are two situations of the fork when it is found: (1) neither of the two events in this fork reaches consensus and (2) only one of the two events reaches consensus. Section 4 will prove that it is impossible that both of the events in a fork can reach consensus. In the first situation, the node deletes the two events and all their

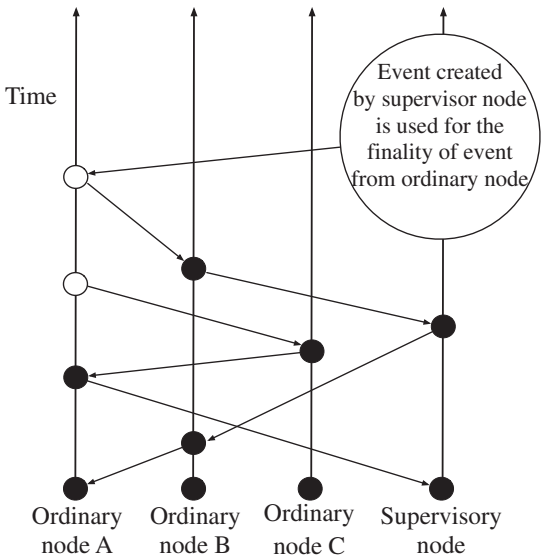

FIGURE 2 The consensus process

Algorithm 1 Algorithm runs on an ordinary node

```
run the following two loops in parallel threads
loop
    send all known events to a random node
end loop
loop
    receive events from its neighbor node N
    create a new event and set the other-parent as the last event from node N
    if any one of the received events is from a supervisor node then
        find new events that reach consensus
    end if
end loop
```

Algorithm 2 Algorithm runs on a supervisory node

```
run the following three loops in parallel threads
loop
    send all known events to a random node
end loop
loop
    receive events from its neighbor node N
    create a new event and set the other-parent as the last event from node N
    find new events that reach consensus
end loop
loop
    monitoring system's memory and the ordinary nodes' behavior
    if the system memory occupancy rate is more than n% then
        take a snapshot and release the memory
    end if
    if node M is found malicious or down for time t then
        replace node M
    end if
end loop
```

children events. Meanwhile, in the second situation, the node only deletes the event that does not reach consensus and all its children events. In both of the above situations, the events from honest nodes would be deleted with the fork too. Thus, if a malicious node continues to create forks, although this attack could be found, the system would stop working because the events from honest nodes would be deleted and never reach consensus. However, in Jointgraph, which is a consortium blockchain, every participant knows each other. Moreover, they have no motive to attack the system because they will get nothing but be caught easily. Once a participant is found to have launched a fork attack, he will be kicked out of the consortium (an honest one can do this simply by removing this malicious node from his neighbor list), and all the honest participants will not communicate with him anymore. According to the assumption, there are no more than 1/3 of malicious participants in total. Thus, in the worst case, all the malicious participants are kicked out, and then, the system runs normally with guaranteed liveness.

2.3 | Take a snapshot

The system state is represented as the tokens' amount that every account holds. Every node may have different system state from his view because some nodes may not receive the latest event. However, from the perspective of one same event, the system state before that event would be the same for every node. As time goes by, the Jointgraph will keep growing. A snapshot of the Jointgraph must be taken periodically to release memory. It can be done by the supervisory node. As shown in Figure 3, there are two types of special events that are issued by the supervisor node for snapshots, namely the snapshot event and the storage event. The snapshot event is the point at which the supervisory node takes a snapshot of

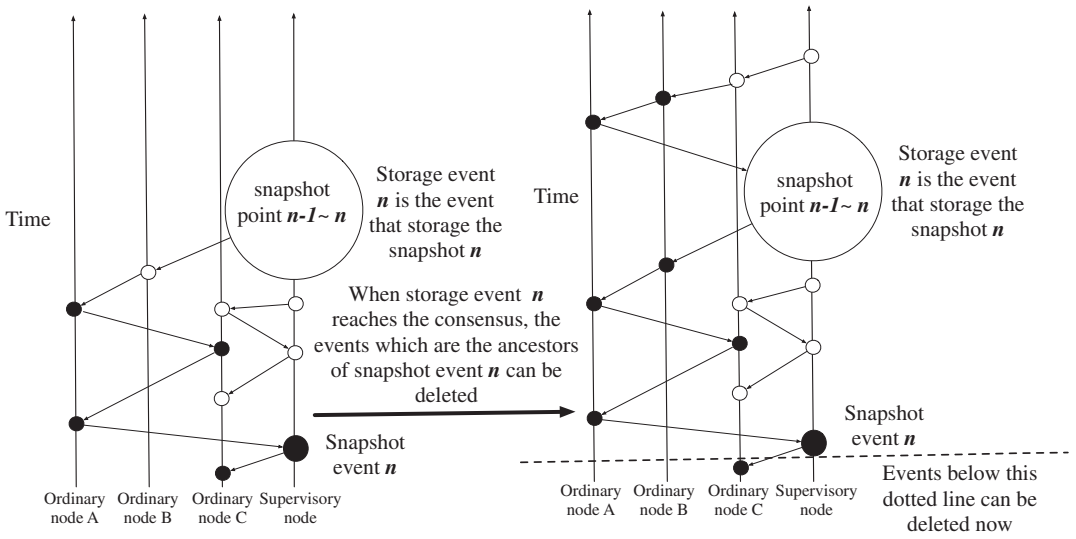

FIGURE 3 Take a snapshot

the state of the system from the perspective of this event, every node has the same system state). The snapshot event is not part of the consensus at the beginning it is chosen, so nodes cannot store the snapshot in the snapshot event itself. The supervisory node will wait for it to be part of the consensus. Once the snapshot event is confirmed by the supervisor node through issuing another event, the supervisory node immediately stores the snapshot in that event and calls it the storage event. Moreover, after the storage event becomes part of the consensus, we can delete the events that are the ancestors of the snapshot event. Thus, the memory is released. The Jointgraph (just a graph) kept in memory does not contain the transactions. It only contains the event header, which includes the event ID and the hashes of its parents. Furthermore, with the help of the snapshot mechanism, the Jointgraph kept in memory will not grow infinitely. Therefore, the space complexity is O(m), in which m is the number of the events on the Jointgraph after the last snapshot.

2.4 | Replace an ordinary node

When the supervisory node finds an ordinary node A needs to be replaced by another node (maybe the ordinary node A becomes a malicious node or for some other reasons), it will issue a special event called a member replacing event to claim the replacement. Once the ordinary nodes receive this special event, they will never send events to node A or receive events from it. Moreover, they will add the new node to their neighbor list. As shown in Figure 4, after ordinary node A

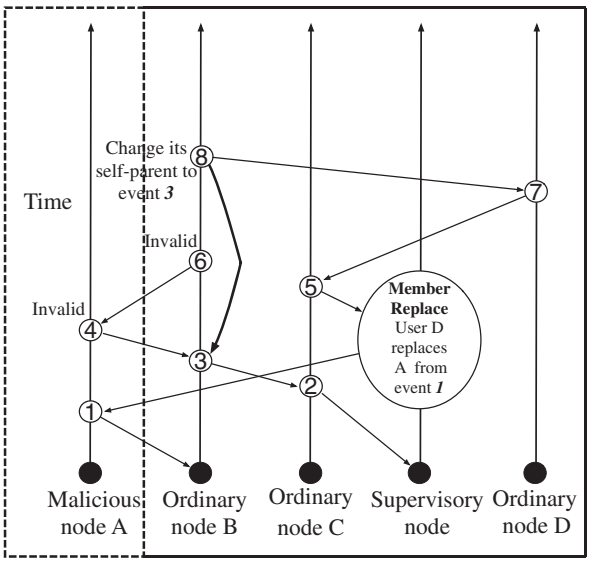

FIGURE 4 Replace an ordinary node

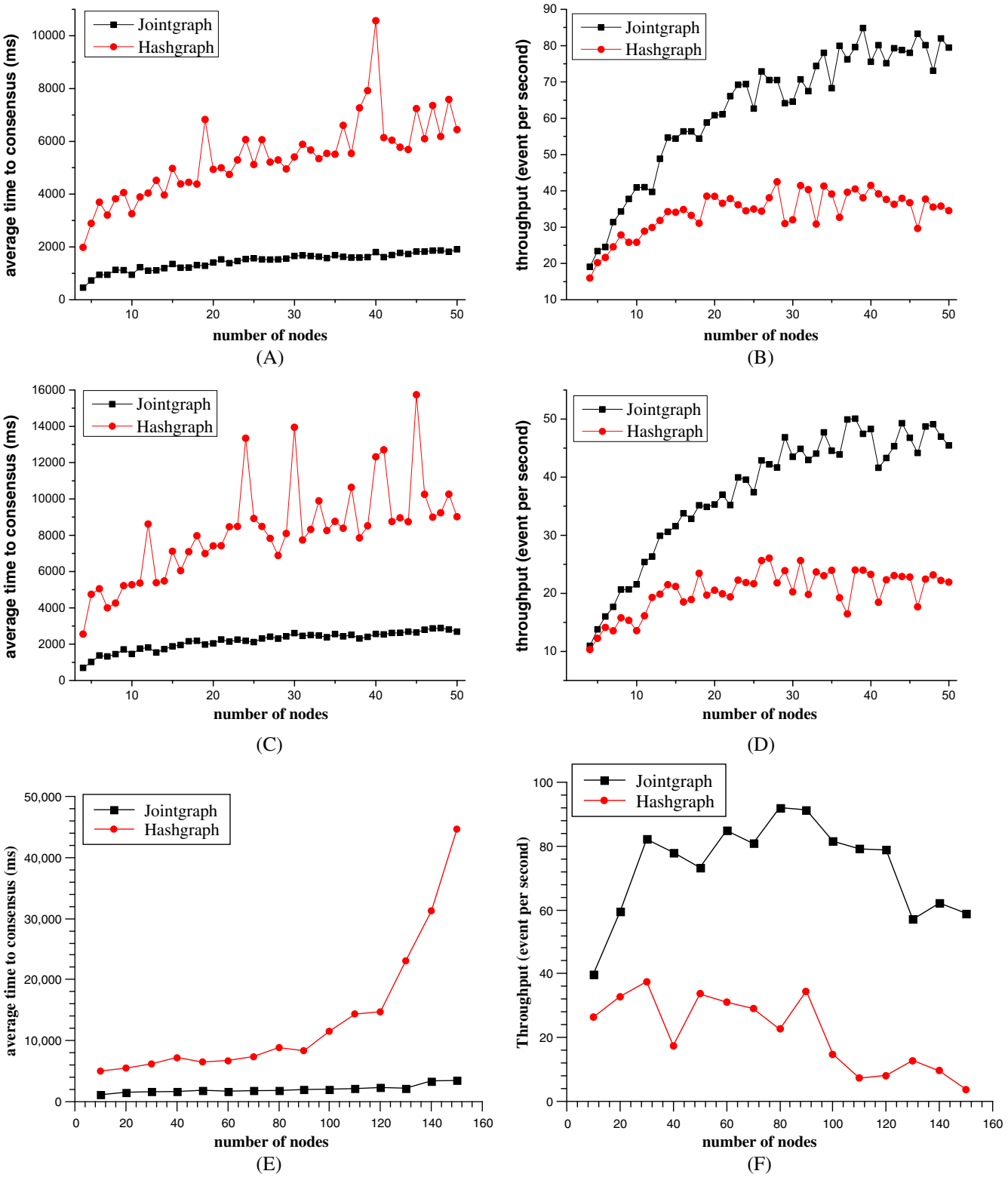

FIGURE 5 Impact of the number of all nodes. A, Latency with event sending interval = 200 ms; B, Throughput with event sending interval = 200 ms; C, Latency with event sending interval ranging from 200 to 500 ms; D, Throughput with event sending interval ranging from 200 to 500 ms; E, Latency with event sending interval = 200 ms; F, Throughput with event sending interval = 200 ms [Colour figure can be viewed at wileyonlinelibrary.com]

issues event 1, it is replaced by ordinary node *D*. Event 1 is the last valid event ordinary node *A* sends. However, before ordinary node *B* receives the member replace event, he receives event 4, which is an invalid (but he does not know that it is invalid) event from ordinary node *A*. He then issues a new event (event 6) and set its other-parent as event 4. Then, he receives event 5 and event 7 from ordinary node *D* and knows that ordinary node *A* has been replaced by ordinary node *D* since event 1. Thus, he sets the self-parent of event 8 as event 3 instead of event 6 and deletes the invalid event 4 and event 6 from his Jointgraph.

3 | SIMULATION EXPERIMENTS

This section contains several sets of simulation experiments by comparing the throughput (eps, event per second) and the latency (ltc, the average time for an event to reach the consensus) of Jointgraph and Hashgraph in different situations including different number of nodes, different number of failure nodes, different event sending interval and different network latency. Each node is simulated as a Java thread, and we set different event sending intervals to represent the event propagation time.

- **Impact of the number of all nodes**
  We first study the impact of the number of all nodes on the throughput and latency of Jointgraph and Hashgraph. We run our simulator for different numbers of nodes ranging from 4 to 50. Each simulation runs independently. Firstly, the event sending interval is set to 200 milliseconds. Then, the event sending interval of each node is set to a random value ranging from 200 to 500 milliseconds. The results are shown in Figures 5A to 5D. Our results suggest that the less the event sending interval, the higher the throughput and the lower the latency for both Jointgraph and Hashgraph. To analyze the scalability of Jointgraph, we add the nodes up to 150 because we think that 150 nodes are enough for a consortium blockchain. In this simulation, the number of nodes is set to 10, 20, 30, … , 150, respectively. The results are shown in Figures 5E and 5F, which suggest that with the number of nodes increases, the latency is higher. The latency of Jointgraph is relatively stable, while when the number of nodes is greater than 120, the latency of Hashgraph has a dramatic increase. The throughput of Jointgraph reaches a peak when the number of nodes is around 90. When the number of nodes is greater than 90, the throughput of Hashgraph has a sharp drop, while Jointgraph can still work normally even when the number of nodes is 150 (the throughput of Jointgraph is about 60 eps while Hashgraph is only about 3 eps). Moreover, in a relatively stable network without the failure nodes, Jointgraph outperforms Hashgraph in both throughput and latency in all the above simulations.
- **Impact of the number of failure nodes**
  We then study the impact of the number of failure nodes on the throughput and latency of Jointgraph and Hashgraph. The number of the nodes is set to 50, and we run our simulator for different numbers of failure nodes ranging from 1 to 15 ($15 < 50/3$). The event sending interval is 200 milliseconds. Each simulation runs independently. The results

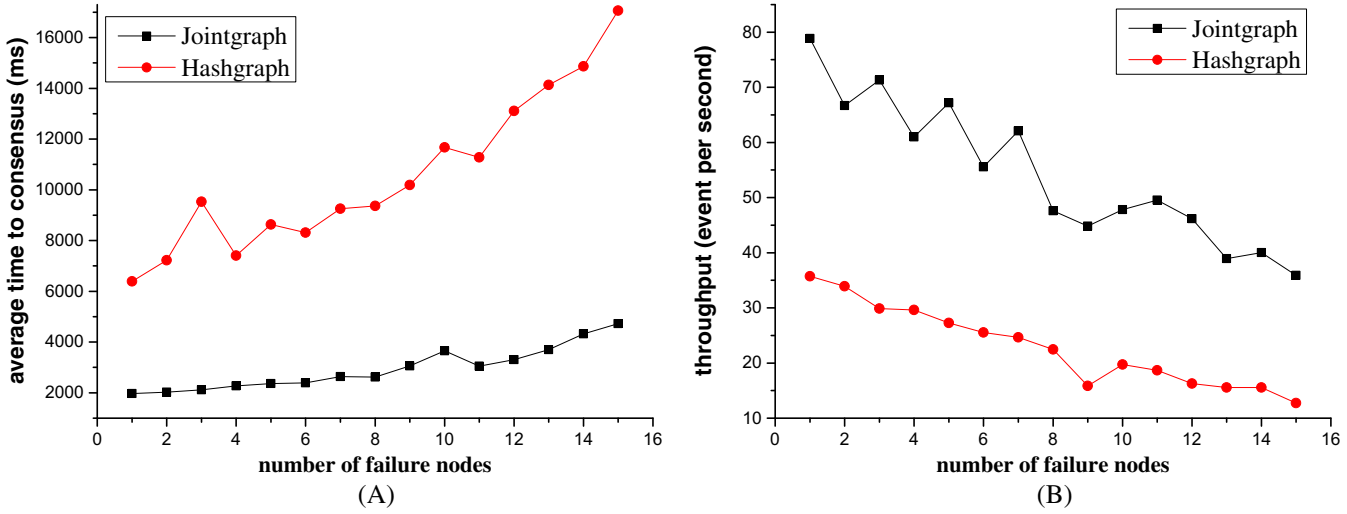

FIGURE 6 Impact of the number of failure nodes. A, Latency with event sending interval = 200 ms; B, Throughput with event sending interval = 200 ms [Colour figure can be viewed at wileyonlinelibrary.com]

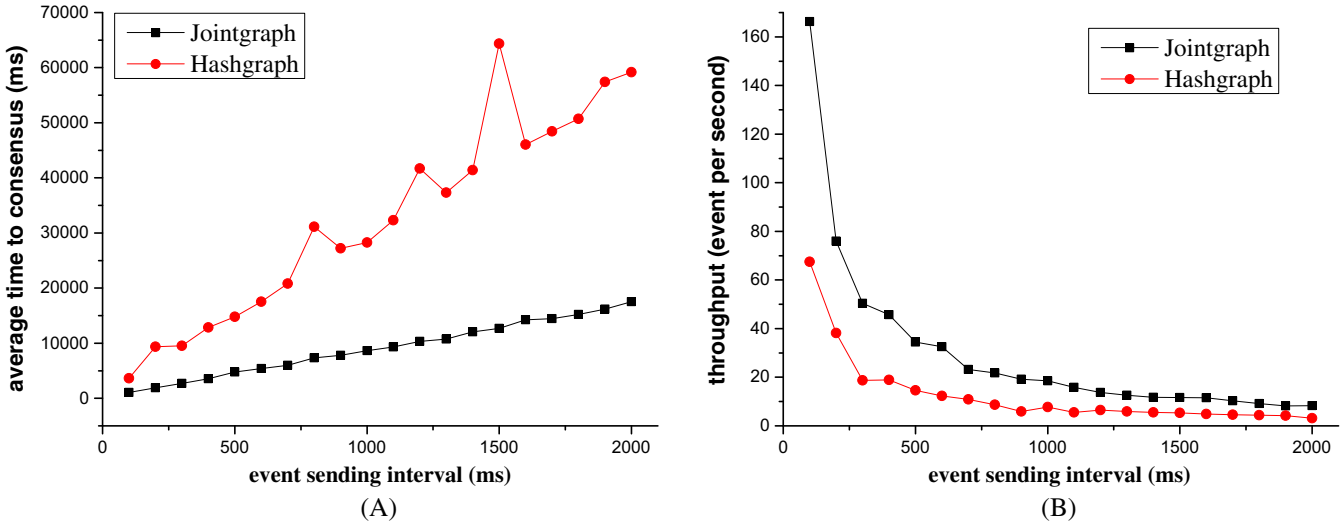

FIGURE 7 Impact of the event sending interval. A, Latency with 50 nodes; B, Throughput with 50 nodes [Colour figure can be viewed at wileyonlinelibrary.com]

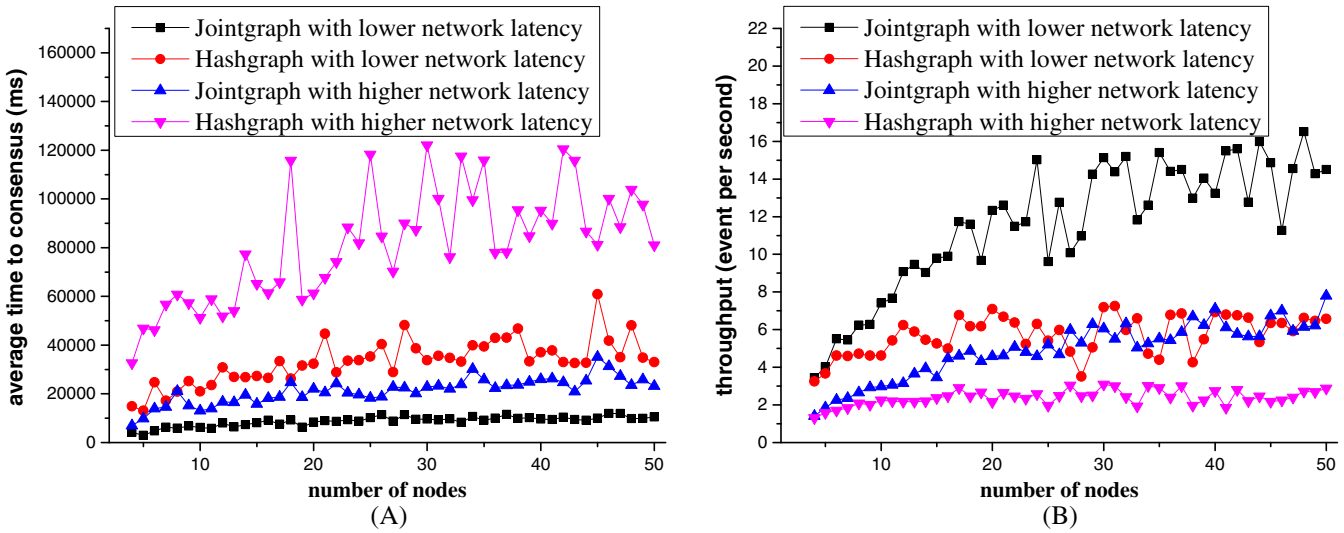

FIGURE 8 Result of simulations in more complex situations. A, Latency with different network delay; B, Throughput with different network delay [Colour figure can be viewed at wileyonlinelibrary.com]

are shown in Figure 6. Our results suggest that the more the failure nodes, the lower the throughput and the higher the latency. Moreover, Jointgraph is much less affected than Hashgraph in latency. With the failure of several nodes, Jointgraph outperforms Hashgraph in both throughput and latency.

- **Impact of the event sending interval**
  We now study the impact of the event sending interval. The number of the nodes is set to 50, and we run our simulator for different event sending interval ranging from 100 to 2000 milliseconds. Each simulation runs independently. The results are shown in Figure 7. Our results suggest that the latency is positively related to the event sending interval and Jointgraph is much less affected than Hashgraph in latency. Moreover, Jointgraph outperforms Hashgraph in both throughput and latency no matter what the event sending interval is.

- **Simulations in more complex situations**
  At last, we perform simulation experiments in more complex situations. The numbers of nodes range from 4 to 50. If a node waits for events more than 500 milliseconds from a neighbor, he will ignore these events and request events from another node. In the first situation, the event sending interval of each node is set to a random value ranging from 100 to 1100 milliseconds, while in the second situation, it is set to a random value ranging from 100 to 1600 milliseconds

(the network delay of the second situation is larger than the first). The results are shown in Figure 8, which suggest that as the network delay increases, the Jointgraph is less affected and outperforms Hashgraph in both throughput and latency.

4 | DISCUSSION

In this section, we first discuss the correctness of Jointgraph, including the consistency and liveness. Then, we discuss the security by analyzing the resilience to several attacks including double spending attacks, Dos attacks,[27] Sybil attacks,[28] and Eclipse attacks.[29]

Consistency: We will prove that the double spending can never succeed, and only one of the events in a fork could be valid, which proves the safety of Jointgraph. In this proof, we assume that there are n members ($n > 4$), more than 2/3 of which are honest and less than 1/3 of which are not honest (could be offline for a long time or make forks). It is also assumed that the asymmetric encryption algorithm for digital signatures is secure, so signatures cannot be forged. We also assume that the hash function is secure and hash collisions can never be found.

The proof is by contradiction. Suppose event (x, y) is a fork and both x and y are accepted by more than 2/3 of all the ordinary nodes, and the finality is confirmed by the supervisory node. In this case, if there is a double spending in this fork, the attack succeeds because both of the two events are part of the consensus. However, for honest nodes, they will either accept x or y, while for the malicious nodes, they may both accept x and y. Suppose the number of malicious is f and the number of the honest nodes that accept event x is h_1, the number of the nodes that accept event y is h_2. Then,

$$\begin{cases} h_1 + h_2 + f \leq n \\ h_1 + f > 2/3n \\ h_2 + f > 2/3n. \end{cases} \tag{1}$$

From Equation (1), it is figured out that $f > n/3$, which contradicts the assumption that $f < n/3$. Thus, it is impossible that both event x and y are valid.

According to the proof above, only one event from the fork can be valid, which means that Jointgraph can prevent the double spending attacks, and it is consistent. Furthermore, if the fork is found by the supervisory node, the fork creator would be kicked out of the consortium or replaced by another ordinary node.

Liveness: Liveness means that valid events made by honest nodes will always reach consensus. In Jointgraph, an event must reach the supervisory node first, and then, the supervisory node can determine whether it has reached consensus by collecting the votes it gets. The event can get a "YES" vote from the node it reaches. Thus, if an event can reach all the nodes, it can get supermajority "YES" votes from all the honest nodes. The supervisory node collects these votes when the nodes send events to it. Actually, the gossip-based dissemination can make the events get spread exponentially fast through the network. The time complexity for the supervisory collecting votes is related to the gossip protocol. According to our design, the supervisory node collects votes by receiving events from the ordinary nodes through the gossip. In general, it takes O(log$N$) rounds to reach all nodes, where N is the number of nodes.[30] Therefore, the time complexity for the supervisory collecting votes is also O(log$N$). The gossip protocol guarantees that any event from an honest node can reach all the honest nodes eventually, which proves that those valid events made by honest nodes will always reach consensus. As described in Section 2.2, Jointgraph is a consensus algorithm for the consortium blockchain, and the liveness can still be guaranteed even there exists fork attacks.

Security: To launch a double spending attack, the malicious node needs first to make a fork in Jointgraph and then spend the same token in this fork twice. However, in Section 2.2, we have explained how to deal with the fork to guarantee the consistency of Jointgraph, which indicates that double spending attacks can be prevented in Jointgraph. For Dos attacks, the attacker makes a node unavailable to its intended users by temporarily or indefinitely disrupting services. Therefore, if the attacker aims at an ordinary node, the consensus process will not be affected because we could treat the victim as a failure node, and the simulation about the "impact of the number of failure nodes" above has proved that if the number of failure nodes is less than 1/3 of all the nodes, Jointgraph can still work and outperform Hashgraph. If the supervisory node is hacked down and stop working by Dos attacks, the ordinary nodes cannot determine the finality of an event by the votes collected by the supervisory. However, the consensus process of Jointgraph does not contradict that of Hashgraph, which means the ordinary nodes can switch Jointgraph to Hashgraph before the new supervisory is online again. For Sybil attacks and Eclipse attacks, the attackers need to manipulate a large number of nodes. While

under the assumption that the number of malicious nodes is less than 1/3 of all the nodes and in a consortium blockchain environment, the attackers cannot have so many identities because there is always an access mechanism for choosing ordinary nodes. Therefore, Jointgraph can resist Sybil and Eclipse attacks.

5 | RELATED WORK

DAG-based blockchains: Researchers have proposed several consensus algorithms over DAG-based blockchains, including Hashgraph, Byteball,[31] IOTA,[32] PHANTOM, Conflux, etc. Hashgraph is proposed for replicated state machines with guaranteed Byzantine fault-tolerance. The nodes send transactions through a gossip protocol, in which the participants not only gossip about transactions but also gossip about the gossip history (This is called gossip about gossip). Moreover, every node builds a Hashgraph reflecting all of the gossip events. Nodes can reach Byzantine agreement on a total order for all transactions locally (according to the Hashgraph they hold) without any more communication. We also use the "gossip about gossip" protocol in Jointgraph. However, it is challenging to take a snapshot for the Hashgraph to release the memory because of its complex confirmation protocol. Furthermore, if some node is found malicious, there is no rule for the node replacement. The consensus algorithm of Byteball is also based on DAG. Each user should trust 12 witnesses, which are responsible for the transactions' finality. Every transaction is packed into a unit, and units are linked to each other such that each unit includes one or more hashes of earlier units. Byteball's security relies on the principle that the number of malicious witnesses is less than a half. However, the throughput of Byteball is very slow, and it must take several minutes for a transaction to be confirmed. Tangle is a DAG for IOTA to store transactions. For IOTA, the efficiency and security can be improved with the number of nodes. Moreover, users are not required to pay any transaction fees. Thus, IOTA is explicitly designed for the IoT industry. In IOTA, when a new transaction arrives, it must approve two previous transactions. These approvals are represented by directed edges. In the case where there are conflicting transactions, the nodes need to decide which transaction will become valid by running the Markov Chain Monte Carlo algorithm. However, in the current version of IOTA, there is a fully centralized role called "coordinator" to validate the transactions. [33,34] PHANTOM follows Bitcoin's model in almost every respect, including PoW, computationally bounded attacker, probabilistic security guarantees, etc. The "only" difference is that a block references (possibly) several predecessors rather than a single one. Conflux is a fast, scalable, and decentralized blockchain system, in which the consensus protocol is no longer the throughput bottleneck. The bottleneck is at the processing capability of individual nodes. Similar to Bitcoin, Conflux also operates with PoW mechanism. While in a consortium blockchain environment, it is unnecessary to use the energy wasting PoW algorithm to reach consensus.

Blockchain with weak centralized nodes: Hyperledger Fabric is an open-source project within the Hyperledger umbrella project. It is a flexible permissioned blockchain platform. Version 1.0 was launched in early 2017 and a critical shortage of which is its extensibility. To overcome this limitation, researchers have designed, implemented, and evaluated a BFT ordering service, which is provided by several weak centralized ordering nodes. The ordering service can achieve up to ten thousand transactions per second.[35] Our Jointgraph also sets a weak centralized node called a supervisor to improve the consensus efficiency.

6 | CONCLUSION AND FUTURE WORK

In this paper, we propose Jointgraph, which is a DAG-based efficient consensus algorithm with proved Byzantine fault-tolerance for consortium blockchains. In Jointgraph, transactions are packed into events, and a new coming event must be validated by more than 2/3 of all the members. There is a supervisor who can monitor the members' behavior and join the consensus process to improve efficiency. The supervisor is also responsible for the memory release by taking snapshots of system states. The supervisor can also replace a malicious member without affecting the process of consensus. The simulation results demonstrate that Jointgraph outperforms Hashgraph, which is one of the most popular DAG-based algorithms, in both throughput and latency in many different situations. Our future work will focus on the implementation of Jointgraph in the real-life environment. Moreover, we are also interested in the cross-chain technologies,[36-38] which are promised to break the information isolated island among different blockchains.

ACKNOWLEDGEMENTS

The research is supported by the National Key R&D Program of China under grant 2016YFB1000100, the National Natural Science Foundation of China under grant 61772030, and the GF Innovative Research Program.

REFERENCES

1. Zheng Z, Xie S, Dai H, Chen X, Wang H. An overview of blockchain technology: architecture, consensus, and future trends. In: Proceedings of the IEEE International Congress on Big Data (BigData Congress); 2017; Honolulu, HI.
2. Yuan Y, Wang F-Y. Blockchain: the state of the art and future trends. *Acta Autom Sin*. 2016;42(4):481-494.
3. Pilkington M. 11 blockchain technology: principles and applications. In: *Research Handbook on Digital Transformations*. Cheltenham, UK: Edward Elgar Publishing; 2016:225.
4. Chen Z, Dong AQ, Sun H. Research on private blockchain based on crowdfunding [j]. *J Inf Secur Res*. 2017;3(3):227-236.
5. Pongnumkul S, Siripanpornchana C, Thajchayapong S. Performance analysis of private blockchain platforms in varying workloads. In: Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN); 2017; Vancouver, Canada.
6. Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. 2008.
7. Gervais A, Karame GO, Wüst K, Glykantzis V, Ritzdorf H, Capkun S. On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16); 2016; Vienna, Austria.
8. Wood G. Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Proj Yellow Pap*. 2014;151:1-32.
9. Sixt E. Ethereum. In: *Bitcoins und andere dezentrale Transaktionssysteme*. Wiesbaden, Germany: Springer; 2017:189-194.
10. Alibaba. Alibaba group generated us$25.3 billion (RMB168.2 billion) of GMV during the 2017 11.11 global shopping festival. 2017. https://www.alibabagroup.com/en/news/article?news=p171112
11. Crain T, Gramoli V, Larrea M, Raynal M. (Leader/randomization/signature)-free Byzantine consensus for consortium blockchains. arXiv preprint arXiv:1702.03068. 2017.
12. Vukolić M. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In: *Open Problems in Network Security: IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*. Cham, Switzerland: Springer; 2015:112-125.
13. Androulaki E, Barger A, Bortnikov V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the 13th EuroSys Conference (EuroSys '18); 2018; Porto, Portugal.
14. Brown RG, Carlyle J, Grigg I, Hearn M. Corda: an introduction. *R3 CEV*. 2016.
15. Yoo S. Blockchain based financial case analysis and its implications. *Asia Pac J Innov Entrepreneursh*. 2017;11(3):312-321.
16. Castro M, Liskov B. Practical Byzantine fault tolerance. In: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99); 1999; New Orleans, LA.
17. Gramoli V. From blockchain consensus back to Byzantine consensus. *Future Gener Comput Syst*. 2017. In press.
18. Dinh TTA, Wang J, Chen G, Liu R, Ooi BC, Tan K-L. Blockbench: a framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17); 2017; Chicago, IL.
19. 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; 2017; Shanghai, China.
20. Sompolinsky Y, Zohar A. PHANTOM: a scalable BlockDAG protocol. *IACR Cryptol ePrint Arch*. 2018;2018:104.
21. Li C, Li P, Zhou D, Xu W, Long F, Yao A. Scaling Nakamoto consensus to thousands of transactions per second. arXiv preprint arXiv:1805.03870. 2018.
22. Rocket T. Snowflake to avalanche: a novel metastable consensus protocol family for cryptocurrencies. 2018.
23. Baird L. *The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance*. Swirlds Technical Reports SWIRLDS-TR-2016-01. 2016.
24. Chen Q, Bridges RA. Automated behavioral analysis of malware: a case study of WannaCry ransomware. In: Proceedings of the 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA); 2017; Cancun, Mexico.
25. Lamport L, Shostak R, Pease M. The Byzantine generals problem. *ACM Trans Program Lang Syst*. 1982;4:382-401.
26. Karame GO, Androulaki E, Capkun S. Two Bitcoins at the price of one? Double-spending attacks on fast payments in Bitcoin. *IACR Cryptol ePrint Arch*. 2012;2012(248).
27. Li X, Jiang P, Chen T, Luo X, Wen Q. A survey on the security of blockchain systems. *Future Gener Comput Syst*. 2017. In press.
28. Douceur JR. The sybil attack. In: *Peer-to-Peer Systems: First International Workshop, IPTPS 2002 Cambridge, MA, USA, March 7-8, 2002 Revised Papers*. Berlin, Germany: Springer; 2002.
29. Singh A. Eclipse attacks on overlay networks: threats and defenses. In: Proceedings of the IEEE INFOCOM; Barcelona, Spain; 2006.
30. Kermarrec A-M, van Steen M. Gossiping in distributed systems. *ACM SIGOPS Oper Syst Rev*. 2007;41(5):2-7.
31. Churyumov A. Byteball: a decentralized system for storage and transfer of value. 2016. https://obyte.org/Byteball.pdf
32. Popov S. The tangle. *Cit On*. 2016:131.
33. Benčić FM, Žarko IP. Distributed ledger technology: blockchain compared to directed acyclic graph. In: Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS); 2018; Vienna, Austria.
34. Schueffel P. Alternative distributed ledger technologies blockchain vs. tangle vs. hashgraph - a high-level overview and comparison. *SSRN Electron J*. 2018.
35. Bessani A, Sousa J, Vukolić M. A Byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In: Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers (SERIAL '17); 2017; Las Vegas, NV.
36. Hueber O. The blockchain and the sidechain innovations for the electronic commerce beyond the Bitcoin's framework. *Int J Transitions Innov Syst*. 2018;6(1):88-102.

37. Hope-Bailie A, Thomas S. Interledger: creating a standard for payments. In: Proceedings of the 25th International Conference Companion on World Wide Web; 2016; Montreal, Canada.
38. Buchman E. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains* [PhD thesis]. 2016.

How to cite this article: Xiang F, Huaimin W, Peichang S, Xue O, Xunhui Z. Jointgraph: A DAG-based efficient consensus algorithm for consortium blockchains. *Softw: Pract Exper*. 2019;1–13. https://doi.org/10.1002/spe.2748