PAS: Enable Partial Consensus in the Blockchain

Zihuan Xu, Siyuan Han, and Lei Chen(✉)

The Hong Kong University of Science and Technology, Hong Kong, China
{zxuav,shanaj,leichen}@cse.ust.hk

Abstract. Permissioned Blockchain enables distributed collaboration among organizations that may not trust each other. However, existing systems cannot efficiently support the ordering and execution of transactions in different workflows parallelly, which seriously affects system scalability and performances in terms of throughput and latency.

In this paper, we present a partial consensus mechanism named PAS to achieve fault tolerance and parallelism of transaction processing. In PAS, transactions in different workflows only need to be confirmed by the involved subset of nodes, which significantly enhances the system performance and scalability. Specifically, we introduce a novel data structure, called the hierarchical consensus tree (HCT). It is maintained in each node and used to coordinate the consensus process. HCT guarantees that the consistency reached in different sets of nodes is eventually agreed by all nodes without conflicts and rollbacks. Since there are many valid HCTs with different system improvements, we introduce an optimization problem, named OHCT, to obtain an HCT with respect to the optimal enhancement. We prove OHCT is NP-hard and propose a general framework with efficient algorithms to address it. Finally, we implement PAS on PBFT-based Hyperledger fabric and conduct extensive experiments to show the performance and scalability of PAS.

1 Introduction

The *permissioned Blockchain* (*e.g.*, Hyperledger [2], Multichain[1], and Tendermint [5]), where the node identities are controlled and known by each other, builds a dedicated environment to prompt accountable interactions among users. However, its performance and scalability issues caused by the underlying consensus mechanism arise many concerns [10].

Most existing permissioned chains require every node to maintain a single ledger and treat the system as a *replicated state machine* to reach global consistency by involving all nodes at any time [2], meanwhile, transactions are executed and ordered sequentially. Thus, it fails to parallelize the transactions that are not dependent on each other, which leads to low system performance and scalability. Consider a supply chain management example described in [24] where a role in the supply-chain workflows has multiple instances as shown in Fig. 1.

[1] https://www.multichain.com/.

© Springer Nature Switzerland AG 2021
C. S. Jensen et al. (Eds.): DASFAA 2021, LNCS 12683, pp. 375–392, 2021.
https://doi.org/10.1007/978-3-030-73200-4_26

Example 1. Suppose there are two under processing workflows: **Workflow 1:** A factory F_1 has produced some products P_1 and stored them in a warehouse W_2. Currently, a retailer R_3 places an order O_1 to purchase these products. **Workflow 2:** A retailer R_1 places an order to purchase product P_2 firstly. Now, factory F_1 confirms the order and places another order to P_2's material supplier S_1. Meanwhile, F_1 and S_1 agree to deliver the material M_2 by carrier C_1.

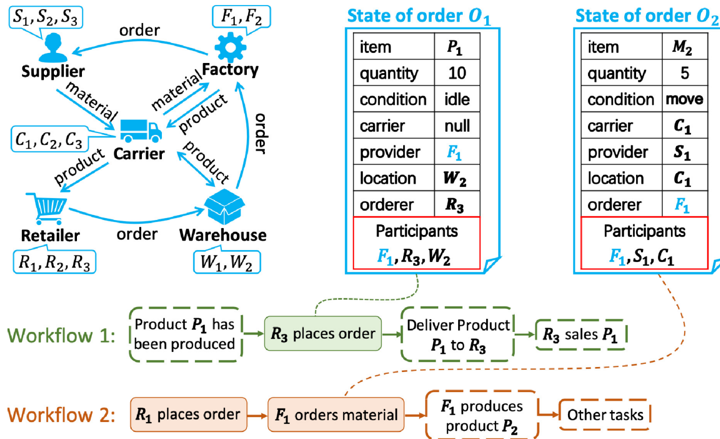


Fig. 1. Example in the supply-chain scenario

Here, companies collaborate to accomplish different supply-chain workflows which can be divided into several tasks with different service-level agreements (SLAs) agreed by related users to determine the data to read/write and the responsibility of each user. These SLAs can be represented in *smart contracts* [23]. Participants of each task modify the task data by transactions. Specifically, contracts O_1 and O_2 record data states of two tasks (R_3 orders product P_1 and F_1 orders material M_2). Since the data in O_1 and O_2 do not have overlap and dependency, participants of two tasks can order and execute task transactions internally and parallelly, because they are the only current valid modifiers of the task data. Notice that, each company can participate in multiple tasks simultaneously (*e.g.*, factory F_1 produces different products in both tasks) and join or leave a task at any time (*e.g.*, carrier C_1 only participates in O_2 for a while and after the delivery, later state of O_2 is irrelevant to C_1). Thus, to maintain a complete Blockchain modification history, the participation order of each user in each task needs to be preserved and globally agreed without conflicts.

In summary, to work in a more general scenario and obtain high performance and scalability, a Blockchain should satisfy at least the following three requirements: **(1)** Users can flexibly join or leave the modification processes of different data with separate SLAs. **(2)** Transactions modifying data without dependency can be ordered and executed in parallel. **(3)** The total order of transactions with

dependencies and the users' participation order of the data modifications can be eventually agreed by the entire system under a Byzantine fault environment.

Most existing works satisfy *requirement (2)* by introducing sharding of nodes. For example, Hyperledger fabric v1.0+ uses channels and CAPER [1] uses applications to separate nodes. Such a mechanism enables processing transactions in different shards in parallel. However the settings of shards are static, it is inflexible for a user to join or leave a shard that fails to efficiently support *requirement (1)*. Moreover, to fulfill *requirement (3)*, fabric adopts a trusted channel to deal with cross-channel transactions [3] which breaks the decentralization principle of the Blockchain. While, to order the cross-application transaction in CAPER, all system nodes are involved which brings high communication cost in running the BFT consensus protocol. Especially, when the cross-application transactions occupy the majority, the system latency increase dramatically [1].

To overcome the shortcomings of existing approaches, in this paper, we propose a novel consensus mechanism called PAS. In particular, to satisfy *requirement (1)*, we separate the transactions into tasks. In a period, specific data can only be updated by users in one task. We use a special transaction to globally specify the task participants, such that a user can join or leave a task at any time. To satisfy *requirement (2)*, we order and execute transactions in different tasks in parallel. A scope of involved nodes reach strong consistency by the BFT protocol (*e.g.*, PBFT [7]). To satisfy *requirement (3)*, we propose a data structure named *hierarchical consensus tree* (HCT) to coordinate the consensus process to ensure the eventual agreement on every partial consensus without conflicts and rollbacks. Besides, as different HCT constructions can affect the system performance and scalability, to get the optimal HCT, we define an optimization problem. Though the problem is NP-hard, we managed to propose efficient solutions to it. We summarize our contributions as follows:

- **(i)** We separate transaction types in the distributed collaboration scenario and identify the challenges to order them in parallel. Then we propose a partial consensus mechanism named PAS to address these challenges.
- **(ii)** We propose a structure named hierarchical consensus tree (HCT) to support our consensus mechanism and introduce the OHCT problem to obtain an optimal HCT with the maximum system performance and scalability improvement.
- **(iii)** We prove the NP-hardness and approximation hardness of the OHCT problem. Thus, we propose efficient algorithms to construct the HCT.
- **(iv)** We implement PAS on PBFT-based Hyperledger Fabric as an example, and conduct extensive experiments to evaluate the system improvement as well as the effectiveness and efficiency of HCT construction algorithms.

The rest of the paper arranged as follows: Sect. 2 reviews related works. Section 3 overviews the PAS mechanism. Section 4 and Sect. 5 introduce the consensus in PAS and propose the HCT to realize the mechanism. Section 6 introduces the OHCT problem and the general framework with efficient algorithms to address it. Section 7 shows experiments and evaluations. We conclude in Sect. 8.

2 Related Work

Sharding Techniques. To achieve *requirements 1 and 2*, the sharding technique divides and maintains system states in several shards. The maintainers of shards consent in parallel on the execution order of transactions updating the states within each shard. However, to fulfill the *requirement 3*, the biggest challenge is to deal with the cross-shard transactions updating states in different shards simultaneously. Existing solutions (*i.e.*, RapidChain [27], OmniLedger [14] and Elastico [17]) are limited to the UTXOs transaction model. A recent work [9] applies sharding with SGX [18] under general workloads. It relies on a dedicated committee running BFT protocol to deal with cross-shard transactions. However, without a well-designed system state sharding schema, the majority of transactions can be cross-shard that is costly to deal with. Thus, in the worst case, the performance is merely the same as the system without sharding.

Hyperledger shards the users in different workflows by channels which are partitions of the network. However, channels are isolated from each other. It is inflexible for a node to join or leave a workflow at any time since the configuration of channels is fixed. Moreover, interactions between channels rely on a trusted channel [3] or an atomic commit protocol [2], which either breaks the decentralization or still treat the system as an entirety. Meanwhile, CAPER [1] adopts a similar idea to shard the users into applications based on their collaboration workflows and process transactions of each application internally. To solve the cross-application transactions, additional BFT protocols are designed. However, the protocols still need to involve all nodes. Moreover, if such transactions occupy the majority, the system latency increase dramatically [1].

Directed Acyclic Graph (DAG). By changing the chain-like structure to a directed acyclic graph, transactions can be appended to multiple branches in parallel which satisfy *requirements 1 and 2*. For example, IOTA [20] and Byteball [8] are two DAG-based permissioned chains. To satisfy the *requirement 3*, IOTA enforces each new transaction to pick two existing transactions as its predecessors and use the number of transactions' descendants and a PoW nonce as the proof to prevent conflicts. However, similar to PoW, the security of such protocol is nondeterministic and it is limited to cryptocurrency applications. While Byteball relies on a set of privileged users to order the transactions which breaks the decentralization principle and can easily become the performance bottleneck.

3 Overview of PAS

To satisfy all the requirements, PAS shards users into scopes. We use tasks and a special transaction to specify valid modifiers of a set of system states. We organize the scopes into a tree-like structure with each tree node accompanied by a Blockchain ledger. For each transaction, we determine which user scopes are compulsory to reach the consensus based on modified states and current

valid state modifiers. Such that, the order of each transaction can be determined immediately after only a portion of all nodes reaching consensus. The result is eventually propagated to the system which significantly improves the parallelism.

Similar to prior works [9,14,19], we make two assumptions on the network: (i) Nodes are fully connected with each other. (ii) The message sent by an honest node can be eventually received by others within a maximum delay.

User and Validator. We denote *users* in PAS by $U = \{u_1, u_2, .., u_n\}$. Besides, like other permissioned chains, a set of nodes known as *validators* process transactions and ensure the consistency of states on behalf of the users. In PAS, we set the number of validators and users equally. Moreover, cryptography signatures are used to ensure the integrity and authenticity of a message sent by each node.

Consensus Scope and Ledger. With the similar idea of node partitions [12,13,25], in PAS, we partition validators to several disjoint sets named *consensus scope* with the cardinality in the range of $[k, 2k)$. Details will be discussed in Sect. 4. Validators in each scope maintain a ledger consisting of the transactions ordered within the scope. PAS works in epochs denoted by e. In each epoch, validators are shuffled to serve different scopes. Besides, each user is assigned to one scope to process transactions. Different from validators, the user-scope assignment is static. Therefore, the user-scope assignment also determines the structure of consensus scopes. We use $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_m\}$ to denote the consensus scopes where $\mathcal{N}_i$ also represents the users assigned to scope i ($\mathcal{N}_i \subset U$). The organization of the consensus scope is shown at the top of Fig. 2.

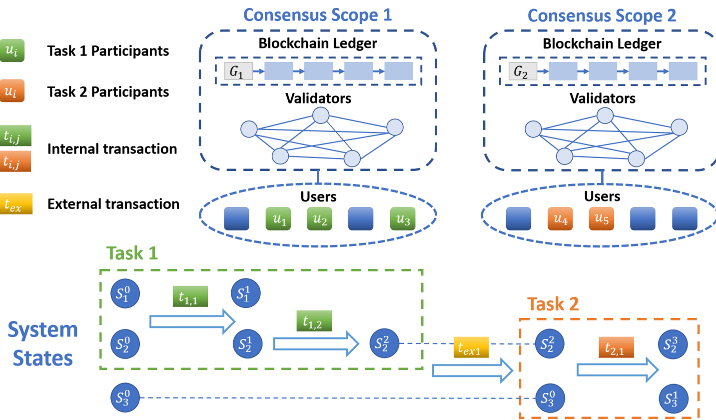


Fig. 2. Overview of the system

Data Model. Beyond UTXO-based model [19], we focus on the state-based model introduced in Ethereum [23]. Specifically, the system states denoted by $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, can be created, updated or deleted by transactions.

Task and Participant. We define a *task* as successive modifications on a set of system states within a group of users (task *participants*). Besides, modifications

may depend on other states within but not beyond the task (otherwise, those states should be involved in this task as well). Thus, the transaction execution order within a task is determined by its participants. In Fig. 2, there are two tasks with different users to update different states. For instance, once the system agrees on states (S_1, S_2) and participants (u_1, u_2, u_3) involved in *task 1*, until finishing the *task 1*, only its participants are authorized to generate transactions and determine their execution order to modify states in the task.

Transactions. We divide the transaction in PAS into two types based on the functionality. One is the ***internal transaction*** used to modify states in a task. In Fig. 2 *task 1*, u_1, u_2 and u_3 use transaction $t_{1,2}$ to modify S_2 from the original state S_2^1 to S_2^2. The other is the ***external transaction*** to change the task participants or create a new task. In Fig. 2, between *task 1* and *task 2*, an external transaction t_{ex1} changes the valid modifiers of S_2 with the confirmation of S_2's final state (S_2^2) after *task 1* which is also the initial state of S_2 in *task 2*. Importantly, the external transaction determines valid modifiers of each state.

Definition 1. ***Transaction.*** *A transaction is a tuple* $t = (id, S, op, i, P, P', \Sigma)$, *where id is the unique order of transactions modifying a set of states* S. *op is the operation with parameters. i is the initiator where* $\forall i, u_i \in U$. P *is the set of current valid modifiers of* S *where* $P \subseteq U$ *and* P' *is the new valid modifiers of* S *where* $P' \subseteq U$. Σ *is a set of signatures signed by validators who have ordered* t.

The *id* is to specify the order of transactions and the Σ is obtained during the consensus. For the validation of a transaction t, suppose a function $P(s)$ returns current valid modifiers of state s based on the records kept by each validator, if $\exists s \in t.S,\ P(s) \neq t.P \ or \ t.i \notin P(s)$, t is treated as invalid.

Threat Model. In this paper, we assume the malicious nodes are less than $\frac{1}{3}$, since the BFT protocol is one of our security guarantee. The attackers are adaptive like described in [9,15,17,27] where the corruption of the validator takes time to achieve. Different from other sharding works [9,15,27], instead of assuming all the shards perform honestly, we do not assume all the consensus scopes perform honestly. Instead, nodes in a consensus scope can perform maliciously together to cheat others outside the scope for their profit.

4 Partial Consensus in PAS

Achieve Partial Consensus. As we assign users to scopes and rely on validators to process transactions, given a transaction t, after determining which users are relevant to t, validators in corresponding scopes run the BFT protocol to order and execute t. Besides, we need to identify in which consensus scope t has been consented. We denote $\mathcal{PC}(t) \subseteq \mathcal{N}$ as scopes where t has been ordered. Since validators in each scope are periodically shuffled, after current validators in $\mathcal{PC}(t)$ reach strong consistency, they will sign on t accompanied by the current epoch index e and commit t on the Blockchain ledger of the scope. Others can verify the authority of t through $t.\Sigma$. Notice that all existing BFT protocols are

applicable for PAS. Examples are PBFT [7], Tendermint [5], Zyzzyva [16], Hot-Stuff [26] and MirBFT [21] where all of them can achieve $\lfloor \frac{n-1}{3} \rfloor$ fault-tolerant.

Validator Assignment. For safety, validators shuffling needs to be unbiased. Omniledger combines RandHound [22] with the verifiable random function (VRF) based leader election algorithm [11] to assign validators which can be used in PAS as well. Specifically, with a bounded message transmission delay δ, in each epoch, validators compute a hash value and gossip it to others for a time δ. Then, the one who gets the lowest value is selected as the leader to run the RandHound protocol to generate and broadcast a bias-resistant random number rnd_e with correctness proof. Finally, others can use rnd_e to get the validator-scope assignment. We refer the details and its security analysis to [15].

Consensus Scope Size. The size of a scope is the number of inside validators and we set it in the range of $[k, 2k)$ to seek a balance between security and performance. Fewer validators lead to lower latency and higher throughput [10]. However, it also reduces the safety, especially for the scope with the minimum size k. Since the validator assignment can be treated as random sampling without replacement, we consider the probability to form a fault scope directly. Suppose there are V validators with αV malicious, the random variable X denotes the number of malicious nodes in the scope with k validators. X should follow the hypergeometric distribution where $X \sim (k, V, \alpha V)$. Given a BFT protocol with f (*e.g.*, for PBFT $f = \frac{n}{3}$) malicious tolerance, the probability to form a fault scope is $Pr[X \geq f] = \sum_{x=f}^{k} \frac{\binom{\alpha V}{x}\binom{(1-\alpha)V}{k-x}}{\binom{V}{k}}$. For example, with $V = 128, \alpha = 0.2, k = \frac{V}{8}$, by using PBFT, $Pr[X \geq \frac{k}{3}] = 6\%$. As stated in [9], when k is large enough (*e.g.*, $k \geq 600$), the probability is considered negligible (*e.g.*, $\leq 2^{-20}$).

5 Eventual Consistency

After reaching partial consensus, validators propagate the result to others. Thus, every node will eventually receive all consensus results. However, if we want to finalize a transaction t when only a subset of nodes reaches partial consensus on its execution order, it is vital to prevent the subsequential transactions conflicting with t from being adopted. Therefore, we first analyze possible conflicts.

Definition 2. ***Conflict Transactions.*** *For two valid transactions t_1 and t_2 within the same task ($t_1.P = t_2.P \bigwedge t_1.S \cap t_2.S \neq \emptyset$)) where t_1 is the first to reach partial consensus, if one of the three conditions holds, t_2 conflicts with t_1. 1.* ***internal conflict:*** *$t_1.op \neq t_2.op \bigwedge t_1.P' = t_2.P'$. 2.* ***external conflict:*** *$t_1.op = t_2.op \bigwedge t_1.P' \neq t_2.P'$. 3.* ***dual-conflict:*** *$t_1.op \neq t_2.op \bigwedge t_1.P' \neq t_2.P'$.*

We only focus on the conflict within a task, because, if t_1, t_2 are from two tasks, there must be one external transaction t_{ex} in between of the tasks. As long as we ensure that t_{ex} can be eventually agreed by the system without conflict (has been covered in condition 2 in Definition 2), t_1, t_2 do not have conflict anymore.

Internal Conflict. This happens when the participants of a task concurrently modify the same state with different operations by two transactions with the same *id*. Since task participants remain the same, validators need to reach partial consensus on their order are the same as well.

External Conflict. This happens when two external transactions t_1, t_2 (with the same *id*) change modifiers of the same state S with the final value (say S^*) to different users. To order t_1 or t_2 both current and new modifiers need to be involved. Suppose the current modifiers are in scope $\mathcal{N}_0$. When validators in $\mathcal{N}_0$ act maliciously, they can reach two conflict partial consensus (switching the modification authority to users in scopes $\mathcal{N}_1$ and $\mathcal{N}_2$) with the validators in $\mathcal{N}_1$ and $\mathcal{N}_2$ simultaneously. Meanwhile, users and validators in $\mathcal{N}_1$ and $\mathcal{N}_2$ cannot detect the deviation respectively which leads to the system inconsistency.

Dual-conflict. This happens when the modifier change and state update happen simultaneously. When shifting the state modifiers, an external transaction must specify it is based on which internal transaction to explicitly inform the final state values to new modifiers. For example, in Fig. 2, t_{ex1} is generated after $t_{1,2}$ ($t_{ex1}.id = t_{1,2}.id + 1$) specifying the value of S_2 is S_2^2. For an internal transaction t_{in} ($t_{in}.id = t_{ex1}.id$) modifying S_2 based on the value S_2^2, if t_{ex1} reaches partial consensus first, t_{in} should not be accepted anymore, vice versa.

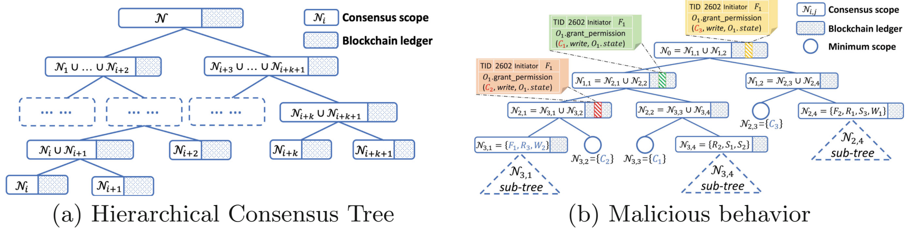
Hierarchical Consensus Tree. To prevent the conflicts, our idea is to control the consensus scopes where a transaction is ordered. We introduce a data structure named *hierarchical consensus tree* (HCT) kept by each node to organize the scopes and Blockchain ledgers and used to coordinate the consensus process. By leveraging a tree structure, any two scopes will share a common root. We restrict a transaction involving users in different scopes to be ordered by all validators under the common root of these scopes. Thus, the conflict can always be detected and we set rules to prevent the acceptance of conflicts.

For a transaction t, all possible combinations of $\mathcal{PC}(t)$ (scopes have reached consensus on t) form a join semi-lattice which is a partial order of the set include operation ($\subseteq$). Given a pair of scopes $\mathcal{N}_1$ and $\mathcal{N}_2$, a least upper bound (LUB) $\sqcup$ exists. $\bar{\mathcal{N}} = \mathcal{N}_1 \sqcup \mathcal{N}_2$ is a LUB of $\{\mathcal{N}_1, \mathcal{N}_2\}$ iff $\forall \mathcal{N}^*, \mathcal{N}_1 \subseteq \mathcal{N}^* \bigwedge \mathcal{N}_2 \subseteq \mathcal{N}^* \Rightarrow \mathcal{N}_1 \subseteq \bar{\mathcal{N}} \bigwedge \mathcal{N}_2 \subseteq \bar{\mathcal{N}} \bigwedge \bar{\mathcal{N}} \subseteq \mathcal{N}^*$. Based on LUB we can have the following definition.

Definition 3. ***Monotonic Consensus Semi-lattice (MCSL).*** *MCSL refers to $\mathcal{PC}(t)$ with the properties: (1) Forms a semi-lattice ordered by $\subseteq$. (2) Merging two scopes $\mathcal{N}_i$ and $\mathcal{N}_j$ involves consensus scopes included in the LUB of $\{\mathcal{N}_i, \mathcal{N}_j\}$. (3) Scope changing is non-decreasing ($\mathcal{PC}(t)$ can only accept new scopes).*

For a transaction t involving users in scopes $\mathcal{N}_1, \mathcal{N}_2$, it needs to be ordered at least by validators in $\mathcal{N}_1$ and $\mathcal{N}_2$. According to the MCSL, it equals to merge the partial consensus results in $\mathcal{N}_1$ and $\mathcal{N}_2$ and t should be ordered in all scopes included in the LUB of $\{\mathcal{N}_1, \mathcal{N}_2\}$. However, for another scope $\mathcal{N}_3$, it is still possible that $(\mathcal{N}_1 \sqcup \mathcal{N}_2) \cap (\mathcal{N}_1 \sqcup \mathcal{N}_3) = \mathcal{N}_1$ which means users in $\mathcal{N}_1$ can still generate conflict transactions without letting nodes in $\mathcal{N}_2$ and $\mathcal{N}_3$ be aware. Therefore, we further define the HCT to address this problem.

Definition 4. ***Hierarchical Consensus Tree (HCT).*** *HCT is a restricted MCSL with the constraint that each scope set can only have one ancestor. In an HCT, each leaf node is a single consensus scope. Each internal node is accompanied by a Blockchain ledger recording the transactions ordered by all validators covered by the consensus scope set of the internal node. Moreover, for two conflict transactions t_1 and t_2, if $\mathcal{PC}(t_2) \subset \mathcal{PC}(t_1)$, t_2 is treated as invalid.*



(a) Hierarchical Consensus Tree (b) Malicious behavior

Fig. 3. Hierarchical Consensus Tree examples

Figure 3a shows an HCT example. The represented consensus scope of a tree node is the union of its two children's scopes. Precisely, each tree node ledger is maintained by validators in the scope of the tree node. A transaction involving users in different scopes should be ordered on the ledger of the tree node representing the LUB of these scopes. Now we prove HCT can prevent the conflict transactions being adopted during the eventual consistency process.

Theorem 1. *By following the hierarchical consensus tree to reach partial consensus, when a transaction t fulfills $(t.P \bigcup t.P') \subseteq \mathcal{PC}(t)$, its conflict transaction t^* cannot be accepted by any correct node.*

Proof. Suppose a transaction t has reached partial consensus and its later generated conflict transaction is represented as t^* (by Definition 2, $t.P = t^*.P$). We use S_1 to denote the set of consensus scopes where users of $t.P$ are assigned. Also, we use S_2, S_3 to denote the scopes where $t.P'$ and $t^*.P'$ are assigned respectively. Based on the HCT, $\mathcal{PC}(t) = S_1 \sqcup S_2$. For different conflicts:

Internal conflict $(t.op \neq t^*.op \bigwedge S_2 = S_3)$**.** To order t^*, it must have $\mathcal{PC}(t^*) = S_1 \sqcup S_3 = S_1 \sqcup S_2 = \mathcal{PC}(t)$. Since the order of t has reached partial consensus in $\mathcal{PC}(t)$ before t' is generated, honest nodes will treat t' as invalid.

External conflict $(t.op = t^*.op \bigwedge S_2 \neq S_3)$**.** To order t^*, it must have $\mathcal{PC}(t^*) = S_1 \sqcup S_3$. If $S_3 \subset (S_1 \sqcup S_2) \Rightarrow (S_1 \sqcup S_3) \subset (S_1 \sqcup S_2) \Rightarrow \mathcal{PC}(t^*) \subset \mathcal{PC}(t)$. According to Definition 4, t^* is invalid. If $S_3 \not\subset (S_1 \sqcup S_2)$, there must be at least one consensus scope $\mathcal{N}$ where $\mathcal{N} \notin (S_1 \sqcup S_2) \bigwedge \mathcal{PC}(t) \subset (S_1 \sqcup \{\mathcal{N}\})$. It means to order t^*, validators who have ordered t must be involved in the process. Thus, validators in $\mathcal{PC}(t)$ can prove that $t.\Sigma$ contains their signatures during the execution of the BFT protocol such that other honest validators in $S_1 \sqcup S_3$ can deny t^*.

Dual-conflict $(t.op \neq t'.op \bigwedge S_2 \neq S_3)$**.** If t is an external and t^* is an internal transaction, there must be $S_1 \neq S_2 \bigwedge S_1 = S_3$, meanwhile, $S_1 = (S_1 \sqcup S_3) \subset (S_1 \sqcup S_2) \Rightarrow \mathcal{PC}(t^*) \subset \mathcal{PC}(t)$. According to Definition 4, t^* is invalid and will be discarded. Else if t is an internal transaction and t^* is an external transaction, there must be $S_1 = S_2 \bigwedge S_1 \neq S_3$, we also consider two cases where $S_3 \subset (S_1 \sqcup S_2)$ or $S_3 \not\subset (S_1 \sqcup S_2)$. Thus, the rest proof is the same as external conflict.

Based on Example 1, we give an HCT example shown in Fig. 3b, to illustrate how it can prevent the generation of conflict transactions.

Example 2. Suppose a transaction t is sent by F_1 to appoint O_1's carrier as C_1 and allow C_1 to modify the state of $O_1.condition$ and t has been ordered by validators in $\mathcal{N}_{1,1}$. Meanwhile, F_1, C_2 and C_3 are malicious and try to generate an external conflict transaction t^* granting the same permission to another user:

Case 1: F_1 grants permission to C_2 which needs to be ordered by the validators in $\mathcal{N}_{2,1}$. Since $\mathcal{N}_{2,1} \subset \mathcal{N}_{1,1}$ and F_1, C_2 know the existence of t during their partial consensus, by Definition 4, t^* is invalid and will be discarded by the validators.

Case 2: F_1 grants permission to C_3 which needs to be ordered by the validators in $\mathcal{N}_0$. Since the honest validators in $\mathcal{N}_{1,1}$ has obtained the confirmation signatures on t, they can prove the existence of t to deny t^* during the running of the BFT protocol and other honest validators will discard t^* as well.

Notice that, for each node, in the path from its position to the tree root, the ledger of each internal node will always be up-to-date. Because whenever an update is made on those ledgers, the nodes will be involved in the consensus process to reach strong consistency. For instance, in Example 2, C_1 always knows the latest transactions committed on the ledgers of $\mathcal{N}_{2,2}, \mathcal{N}_{1,1}$ and $\mathcal{N}$.

6 HCT Optimization

The bottleneck to scale the system is the communication cost to consent the transactions [10], while, ***transaction generation frequency*** **(TGF)** of users (the interaction frequency between users) contributes to the cost.

Definition 5. ***Interaction Frequency.*** *A matrix $\mathcal{F}_{n\times n}$ where $n = |U|$ records the interaction frequency in users. $f_{i,j}$ in $\mathcal{F}$ is the generation frequency of transactions involving u_i and u_j where $u_i, u_j \in U$ and $f_{i,j} = f_{j,i}$.*

For instance, in Example 1, if retailer R_3 often places order O_1, the interaction frequency between F_1, W_2, F_1, R_3 and W_1, R_3 will be high. Notice that, the TGF is an estimation result in a period or determined by actual applications. It is used to establish and reconfigure the system based on the demands of users.

Optimal HCT Problem. The consensus message complexity denoted by $T(\mu)$ where μ is the number of involved validators, is determined by the used BFT protocol (*e.g.*, for PBFT $T(\mu) \in O(n^2)$). Observe that $T(\mu) \propto \mu$. Suppose

users u_1 and u_2 are in scopes $\mathcal{N}_1$ and $\mathcal{N}_2$, the message complexity to consent transactions related to u_1 and u_2 is $f_{1,2}T(\mu_{1,2})$ where $\mu_{1,2}$ is the number of validators in the scopes under $\mathcal{N}_1 \sqcup \mathcal{N}_2$. Therefore, a well-structured HCT can further reduce the message complexity which leads to better system performance and scalability. We define the HCT optimization problem as below:

Definition 6. ***Optimal Hierarchical Consensus Tree (OHCT) Problem.*** *Given the interaction frequency matrix $\mathcal{F}_{n\times n}$, the complexity function $T(\cdot)$ of the BFT protocol and the scope validator cardinality constraint $[k, 2k)$. Our goal is:*

$$\begin{aligned} &\textit{minimize} \quad \sum_{u_i,u_j\in U} f_{i,j}\, T(\mu_{i,j}) \quad (\forall f_{i,j} : i \le j) \\ &\textit{subject to} \quad \min_{u_i,u_j\in U}\{\mu_{i,j}\} \ge k, \ \max_{u_i,u_j\in U}\{\mu_{i,j}\} < 2k \end{aligned}$$

Hardness Analysis. We prove the NP-hardness by studying a special case of OHCT and reducing the minimum bisection problem (MBP) [4] to it.

Theorem 2. *OHCT problem is NP-hard.*

Due to the space, we only show our proof sketch. Consider a special case of OHCT problem where $k = \frac{|U|}{2}$. In this case, we can only bisect users in two sets S_1, S_2 with all transactions completion cost as a constant $T(|U|)$ and minimize $T(|U|)\sum_{u_i\in S_1,u_j\in S_2} f_{i,j}$. Then, we can reduce the MBP to the case. Besides, we further analyze the hardness to get an approximation solution to OHCT.

Theorem 3. *There is no algorithm with constant approximation ratio for OHCT.*

The sketch of the proof is to use the conclusion in [6] that for a fixed $\epsilon > 0$, it is NP-hard to approximate the MBP problem with an additive term of $n^{2-\epsilon}$ [6].

Solution Framework. To solve the OHCT problem, a general framework is to **1. construct** an optimal HCT by regarding each user as a single consensus scope first and **2. pruning and merging** to fulfill the cardinality constraint.

Top-down Construction Algorithm. Intuitively, the greedy way to do the construction is to pair two users into a binary tree first. Then, each time we randomly pick one user from unassigned users. Starting from the tree root, we compare the normalized interaction frequency between the picked user and all users in the left and right sub-tree. Then we go into the root of sub-tree with higher normalized interaction frequency. We stop until we find the suitable leaf node position for all users. Details are shown in Algorithm 1.

Pruning And Merging Algorithm. After we obtain an HCT with each leaf to be one user, we perform DFS on the root. Each time, if the leaf number $|t|$ of an internal node is greater than $2k$, we continue. If $|t| \in [k, 2k)$, we group its leaves to one scope. If $|t| < k$, we reinsert each user in the leaf to the sibling

sub-tree using the top-down construction algorithm. Because, since one sub-tree and its sibling are grouped under the same LUB by the construction algorithm, it means they have more frequent interactions. By merging the sub-tree into its sibling, it will bring less additional completion cost. Details are in Algorithm 2.

Complexity Analysis. Since $|U| = n$, for *Construction* with Algorithm 1, it recurrently decides the position for each user. In an average case, the algorithm takes $O(n^2 \log n)$. For *Pruning and Merging*, the worst case of its DFS takes $O(n)$ while the worst case cost of merge is $O(k \log n)$. In total, it takes $O(nk \log n)$. Since $k \ll n$, the total complexity is $O(n^2 \log n)$.

Bottom-up Construction Algorithm. Although the Algorithm 1 is efficient, its performance is affected by the input order of users. Thus, we can enhance the HCT construction by always considering every users. The idea is that each time we merge two sub-trees with the highest average transaction completion cost. Then, we recompute the cost between the new sub-tree and other sub-trees. We terminate until all users are rooted in the same tree. The intuition is to merge sub-trees with higher average transaction completion cost as earlier as possible to reduce the involved consensus scopes to the most. Details are in Algorithm 3.

Algorithm 1: Top-down HCT Construction

```
Input  : Interaction frequency matrix F and Users U.
Output: tree root of a hierarchical consensus tree.
1  undetermined ← U;
2  find u1 and u2 with minimum interaction frequency;
3  HCTRoot.children ← [u1, u2];
4  undetermined.remove(u1 and u2);
5  foreach user ∈ undetermined do
6      tRoot ← HCTRoot;
7      F_left, F_right ← 0, 0;
8      while |tRoot.leaves| > 1 do
9          foreach leaf ∈ tRoot.leftChild.leaves do
10             F_left += F[user][leaf];
11         foreach leaf ∈ tRoot.rightChild.leaves do
12             F_right += F[user][leaf];
13         F_left /= tRoot.leftChild.leaves;
14         F_right /= tRoot.rightChild.leaves;
15         if F_left >= F_right then
16             tRoot ← tRoot.leftChild
17         else
18             tRoot ← tRoot.rightChild
19     tRoot.children ← [tRoot, user];
20 return HCTRoot;
```

Algorithm 2: HCT Pruning And Merging

```
Input  : root of a HCT and scope size constraint k.
Output: tree root of a hierarchical consensus tree.
1  push root in the search stack S1;
2  mark all internal nodes as unchecked;
3  while S1 is not empty do
4      // |t|: number of leaves of tree t
5      t ← S1.getTop();
6      if t.children are all checked then
7          S1.pop() and continue while loop;
8      t ← unchecked child node of t with fewer leaves;
9      if |t| ≥ 2k then
10         S1.push(t);
11     else if |t| ≥ k then
12         merge all leaf nodes into one scope;
13         mark t as checked;
14     else
15         Pruning branch t and save its leaves in L;
16         t.sibling.parent ← t.sibling.grandparent;
17         foreach l ∈ L do
18             find position for l by searching t.sibling.
19         S1.pop();
20 return HCTRoot;
```

Complexity Analysis. In $n - 1$ rounds merging, the most time consuming part is to maintain the heap which provides the highest interaction frequency among sub-trees. It takes $O(max\{|t_i| \times |t_j|, (n-t)log(n^2)\}) \in O(nlogn^2)$ in average. Thus, using Algorithm 3 makes the time complexity of the framework be $O(n^2 logn^2)$.

Algorithm 3: Bottom-up HCT Construction

```
Input : Interaction frequency matrix F, Users U and cost function T.
Output: tree root of a hierarchical consensus tree.
1  subTrees ← U;
2  foreach t_i ∈ subTrees do
3      foreach t_j ∈ subTrees do
4          avgC(t_i, t_j) ← F[i][j] × T(2); //avgC: average completion cost
5  while |subTrees| != 1 do
6      Merge t_i, t_j with the maximum avgC(t_i, t_j) into t* and remove t_i, t_j from subTrees;
7      foreach t ∈ subTrees do
8          avgC(t*, t) ← ( avgC(t_i,t)×|t_i| / T(|t_i|+|t|) + avgC(t_j,t)×|t_j| / T(|t_j|+|t|) ) × T(|t_i|+|t_j|+|t|) / (|t_i|+|t_j|);
9      subTrees.append(t*);
10 return subTrees.first;
```

7 Experiment and Evaluation

In this section, we first evaluate the effectiveness and efficiency of our HCT construction algorithms on both real and synthetic datasets. Then, we choose *hyperledger fabric v0.6* as a permissioned Blockchain example to implement PAS and measure the performance and scalability enhancement brought by PAS.

7.1 HCT Construction Evaluation

Real Dataset. To obtain the real interaction frequency of Blockchain users, we use the dataset extracted from Ethereum blocks during the period from Dec 17, 2017, to Feb 23, 2018. It contains 14,393,250 unique addresses and 64,719,559 transactions. Specifically, we treat the token transform from one user to another as one task modifying the states of two account balances. We uniformly sample and group the unique addresses to form different sizes of user groups and obtain the interaction frequency distribution matrix among the groups.

Table 1. Synthetic datasets

Number of users $\lvert U\rvert$	4, 8, 16, **32**, 64, 128
σ of normal distribution	0.05, 0.1, **0.15**, 0.2, 0.25, 0.3
a of power-law distribution	1, 2, **3**, 4, 5, 6
Minimum scope size k	**1**, 4, 7, 10, 13, 16

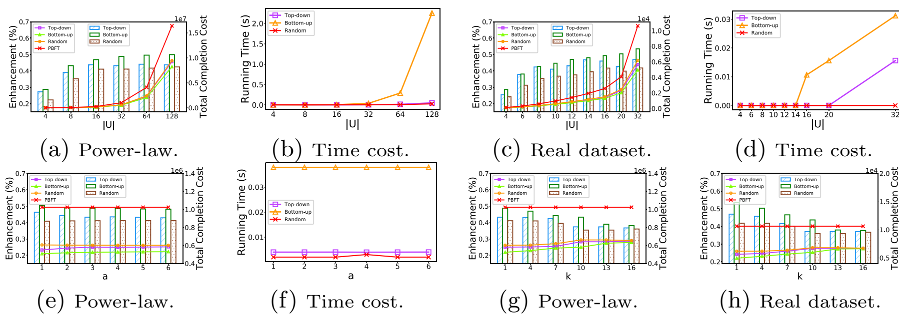
Synthetic Dataset. We generate synthetic interaction frequencies by following **uniform** (in the range of [0,1]), **normal** (with $\mu = 0.5$) and **power-law** (with $c = 1$) distributions. Table 1 shows the parameter settings we used in synthetic datasets and default values are in bold. Similarly, we construct transactions between users as the token transform. We first generate a $|U| \times |U|$ triangular contribution matrix (the sum of all elements is 1) by following the distributions mentioned above. This matrix denote the contribution of each pair of users to the transaction generation frequency (TGF mentioned in Sect. 6). Given the system

TGF, we can obtain the interaction frequency matrix by multiplying the TGF and the contribution matrix. Notice that, TGF is not selected as a parameter. Because the interaction frequency $f_{i,j}$ between two users p_i and p_j is computed by $TGF \times d_{i,j}$ where $d_{i,j}$ is the frequency contrition of user pair u_i and u_j. Thus, the overall message complexity is represented by $TGF \sum_{u_i,u_j \in U} d_{i,j} T(\mu_{i,j})$. For the same experiment settings, TGF is a constant which will not affect the result.

Implementation and Metrics. We implement our HCT construction algorithms in Python 3.7. The experiments are conducted on a server with Intel(R) Core(TM) i5 3.0 GHz CPU with 16 GB RAM. Each experiment is repeated 30 times and we report the average results. We choose PBFT as our baseline consensus protocol and set its $T(\mu) = \mu^2$. In each experiment case, we measure and compute the total message complexity result of each method and the **enhancement percentage** $(1 - \frac{algorithm}{baseline})$ compared with the baseline whose message complexity is $TGF|U|^2$. We compare our solution framework with the HCT construction algorithms of top-down, bottom-up and the random pair which randomly forms a valid HCT. The aim is to show the performance of the PAS even if in a random construction fashion.

Experimental Results. Since the real dataset tends to follow the normal distribution, due to the space, we only report the results on the real dataset and synthetic dataset following the power-law distribution.

Impact of Number of Users $|U|$. The first row of Fig. 4 shows the results of varying $|U|$ in both real and power-law synthetic dataset. The line chart shows the total consensus message complexity, while, the bar chart shows the enhancement percentage comparing with the baseline. For the top-down and bottom-up algorithms, the message complexity reduction is from 25% to 56%. The enhancement of the bottom-up is better than the top-down algorithm from 2% to 10% since the bottom-up always takes $T(\mu)$ into consideration, while, the time cost of bottom-up increases dramatically when $|U|$ increase. With increasing $|U|$ the enhancement of HCT also increases but the incremental speed becomes slower. Because HCT does not change the intrinsic complexity of the protocol itself. PBFT, as an example, with more validators, its $O(n^2)$ message complexity becomes obvious making the enhancement of HCT be a constant factor.



(a) Power-law. (b) Time cost. (c) Real dataset. (d) Time cost.

(e) Power-law. (f) Time cost. (g) Power-law. (h) Real dataset.

Fig. 4. Results of the comparison of HCT construction algorithms

Impact of a in Power-law Distribution. Figure 4e shows the impact of a in power-law distribution. With a larger a, the enhancement of HCTs built by three algorithms all decrease. Because a larger a means most of the interaction frequency is very small. Thus, the enhancement brought by the reduced validators in each consensus process becomes insignificant. Especially, the top-down algorithm is more sensitive to a, since it only considers a single node at each time which is more likely to reach local optimal.

Impact of Consensus Scope Cardinality k. Figure 4g and Fig. 4h show the results on varying k. With larger k, there are fewer consensus scopes in the HCT making the average number of validators need to be involved in each consensus process increase. Thus, the HCT enhancements all decrease. However, even if we only have 2 consensus scopes ($k = 16$), the performance enhancement can still reach 35% indicating to have a better balance between performance and security, it is not necessary to divide the consensus scope into extremely small ones.

Summary of the Results. From the above discussion, with HCT, the total consensus message complexity can always be reduced. Especially, bottom-up construction algorithm can achieve better performance than others with the reduction of PBFT message complexity by at least 30%. Moreover, the HCT enhancement is better when the interaction frequency distribution tends to follows a normal distribution. It means when all users frequently interact with each other, our mechanism can better improve the system. Besides, the cardinality constraint k will not influence the performance too much. The difference between $k = 1$ and $k = 16$ in a 32 nodes system is nearly 10%. Therefore, for better security, it is reasonable to set the k constraint higher.

7.2 PAS Evaluation

We evaluate the actual performance of PAS by implementing it on Hyperledger Fabric v0.6 and evaluate the systems with and without PAS.

Implementation and Metrics. The main aim of PAS is to make transactions be ordered in different consensus scopes based on involved users. Thus, the PAS system should support validators from the same Blockchain network in achieving consensus within different sub-networks. Therefore, we implement an external HCT module to make validators participate in the partial consensus of multiple subnets simultaneously. Specifically, the HCT module mainly does two things: **1. Compute and record the structure of HCT:** Since the HCT is constructed based on the estimated interaction frequency among users, after obtaining the interaction frequency from others (consensus may be required), nodes can build the HCT by running the construction algorithm by themselves. This process is only conducted when forming a new network, or the interaction frequency changes dramatically. Users (even only in a branch of the HCT) can decide to reconstruct the entire (or a sub) tree. **2. Routing transactions:** When a transaction is received by the validator, it will use the HCT module to determine in which scope to reach partial consensus to execute and order the transaction.

Simple key-value storage is implemented to record the valid modifiers of each system state obtained by the transaction history on the ledgers they maintain.

The experiments are conducted on the Azure cloud service cluster. We create validators with 16 GB RAM, 500 GB hard drive, running Ubuntu 18.04 LTS on each of them. They are connected to each other via a 1GB bandwidth network. The aim of our experiment is to measure the peak throughput of each system on varying the number of users/validators (from 4 to 16). We use the official chaincode transferring token between users as the task in the workflow. Then we use client nodes to simulate the transaction generation by following the interaction frequency distribution obtained from the real dataset. In each experiment, we steadily increase the overall system TGF to obtain the peak throughput which is measured by the completion rate from the time a transaction is generated until receiving the commit message of the block which contains the transaction.

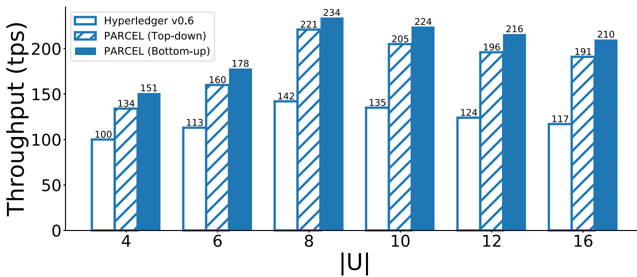


Fig. 5. Result of throughput on varying $|U|$

Experimental Results. Figure 5 shows the throughput comparison between PAS-based and original Hyperledger. The throughput of Hyperledger is between 100 to 142 tps, while, PAS can reach 134 to 234 tps. With the bottom-up constructed HCT, the performance enhancement is at least 50%. Meanwhile, despite the slightly lower enhancement of the top-down algorithm, as compared in Fig. 4 the lower time complexity of the top-down algorithm makes it more suitable for large scale systems. With more users, the enhancement ratio also increases which is similar to what we observed in the HCT construction experiments. Besides, with the nodes increasing, the throughput tends to decrease. In fact, in Hyperledger, to confirm connectivity between nodes, messages such as PeersMessage are sent periodically to check the connection status. When the number of validators in the network increases, such requests also increase, which affects the overall performance of the network to a certain extent. For example, the throughput of the two systems drops linearly from 8 to 16 nodes. Meanwhile, the dropping speed of HCT-based is slower than the original Hyperledger, which can also prove the better scalability of PAS.

8 Conclusions

In this paper, we introduce PAS, a consensus mechanism for permissioned Blockchain to satisfy the requirements in the general distributed collaboration scenario. Specifically, PAS enables a user to join or leave different tasks flexibly by using the transaction to specify the valid modifiers of each system state. We introduce the partial consensus to order transactions in different tasks in parallel. Moreover, to ensure the order of transactions determined by a set of nodes can be eventually agreed by all nodes with the BFT guarantee, we propose the hierarchical consensus tree (HCT) to coordinate the consensus process. When a transaction is ordered, the acceptance of its conflict transaction is strictly prevented. We also propose the OHCT problem to obtain an optimal HCT with the maximum system enhancement. We proved the NP-hardness and approximation hardness of the OHCT problem and propose a framework with efficient algorithms to solve it. Finally, we implement PAS on Hyperledger and conduct extensive experiments to evaluate it. The result shows that PAS can significantly improve system performance and scalability.

Acknowledgment. This work is partially supported by the Hong Kong RGC GRF Project 16213620, CRF Project C6030-18G, C1031-18G, C5026-18G, AOE Project AoE/E-603/18, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, Didi-HKUST joint research lab project, and Wechat and Webank Research Grants.

References

1. Amiri, M.J., Agrawal, D., Abbadi, A.E.: Caper: a cross-application permissioned blockchain. In: VLDB (2019)
2. Androulaki, E., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: EuroSys (2018)
3. Androulaki, E., Cachin, C., De Caro, A., Kokoris-Kogias, E.: Channels: Horizontal scaling and confidentiality on permissioned blockchains. In: ESORICS (2018)
4. Arora, S., Karger, D., Karpinski, M.: Polynomial time approximation schemes for dense instances of np-hard problems. JCSS (1999)
5. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. arXiv preprint arXiv:1807.04938 (2018)
6. Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Inf. Process. Lett. (1992)
7. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI (1999)
8. Churyumov, A.: Byteball: A decentralized system for storage and transfer of value (2016)
9. Dang, H., Dinh, T.T.A., Loghin, D., Chang, E.C., Lin, Q., Ooi, B.C.: Towards scaling blockchain systems via sharding. In: SIGMOD (2019)
10. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: Blockbench: A framework for analyzing private blockchains. In: SIGMOD (2017)
11. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling byzantine agreements for cryptocurrencies. In: SOSP (2017)

12. Han, S., Xu, Z., Chen, L.: Jupiter: a blockchain platform for mobile devices. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 1649–1652. IEEE (2018)
13. Han, S., Xu, Z., Zeng, Y., Chen, L.: Fluid: a blockchain based framework for crowdsourcing. In: Proceedings of the 2019 International Conference on Management of Data, pp. 1921–1924 (2019)
14. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Ford, B.: Omniledger: a secure, scale-out, decentralized ledger. In: IEEE SP (2018)
15. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: a secure, scale-out, decentralized ledger via sharding. In: SP (2018)
16. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative byzantine fault tolerance. In: SIGOPS (2007)
17. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: SIGSAC (2016)
18. Mckeen, F., et al.: Innovative instructions and software model for isolated execution. In: Hasp@isca (2013)
19. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
20. Popov, S.: The tangle. cit. on (2016)
21. Stathakopoulou, C., David, T., Vukolić, M.: Mir-bft: High-throughput bft for blockchains. arXiv preprint arXiv:1906.05552 (2019)
22. Syta, E., et al.: Scalable bias-resistant distributed randomness. In: SP (2017)
23. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper (2014)
24. Wüst, K., Gervais, A.: Do you need a blockchain? In: CVCBT (2018)
25. Xu, Z., Han, S., Chen, L.: Cub, a consensus unit-based storage scheme for blockchain system. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 173–184. IEEE (2018)
26. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: PODC (2019)
27. Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: Scaling blockchain via full sharding. In: SIGSAC (2018)