Programmable Consensus

The author *BUPT*

***Abstract*—This paper conducts a holistic reevaluation of blockchain usability, examining it through the lens of a five-stage consensus lifecycle and introducing a three-dimensional framework for design principles. Building on these insights, we introduce the concept of Programmable Consensus. This approach moves away from the traditional order-execute framework and offers blockchain users programmable interfaces across the entire consensus lifecycle, fully unleashing the potential of blockchain.**

I. Introduction

Blockchain system has gone beyond its cryptocurrency origins, transitioning into broad decentralized platforms that build trust for diverse applications. This evolution has been marked by pivotal structure modifications on the consensus procedure, enabling the transactions to call the execution of Turing-complete code and support the intricate logics through smart contracts.

In this paradigm, transactions are not limited to the transfer of assets but serve as the record of actions. Regardless of the importance of the action, every action requires recording on the blockchain in the form of a transaction, ensuring transparency and immutability nature of blockchain. Ethereum stands as the most recognized implementation of this paradigm, demonstrating its extensive application. Consensus mechanisms, on the other hand, make sure the agreement on transaction history that calling the smart contracts to perform functional logics, meeting the functional usability demands of various applications.

To summarize it, most blockchain system adhere to the same design: a clear demarcation between the blockchain's low level procedure and its application layer. The smart contract and its execution are derived from the transactions that receive direct execution during the consensus, decoupling into an individual layer. This design offers a straightforward way for integrating blockchain into specific scenarios, maintaining structural integrity.

However, it is debatable whether blockchain should rigidly adhere to this paradigm. Smart contracts, which are born out of the consensus mechanism, exhibit design limitations. Firstly, for certain types of embedded blockchain systems, like communication, supply chain management, and the Internet of Things (IoT), a more integrated, or hybrid layer approach is favored. This involves embedding specific application functionalities or logic directly into the core protocol, a practice that can significantly boost efficiency and performance, particularly in handling certain transaction types or data.

Moreover, the dynamics between the application and foundational layers are often more intricate than anticipated. The execution of smart contracts can profoundly affect the performance and security of the blockchain infrastructure, suggesting that application layer design and operations cannot be entirely decoupled from the foundational protocol. A rigid separation of these layers may introduce technical and security challenges. For instance, if the application layer's design inadequately accounts for the foundational protocol's characteristics and constraints, it could lead to exploitable security flaws, and vice versa. Hyperledger Fabric represents a notable stride in addressing the demand for more adaptable blockchain systems. It proposes modular design in its consensus components, attempting to move beyond the 'one-size-fits-all' paradigm. Also, it introduce the concept of system chaincode for low level system define. While these approaches allow users to configure the core components like consensus during deployment, it falls short of providing full access to the inner workings of the consensus process. A more nuanced approach to leverage the core workflow of blockchain is required.

A. Motivation and Contribution

Consensus Lifecycle. To find possible nuanced approaches to leverage the core workflow of blockchain, we rethink the five-stages an action(transaction) undergoes, name as Consensus Lifecycle. Consensus Lifecycle could include most workflows of blockchain systems, where we discussed programmable operations at each stage.

Program Principles. Examine the programmable operations, we included them into different Program Principles from different dimensions, which could guide the design of consensus.

Programmable Consensus. We provide a new paradigm, named Programmable Consensus, that implemented the above principles. Programmable Consensus provides users with interfaces among the Consensus Lifecycle, making many key components programmable. Three use cases that leveraged the Programmable Consensus is provided for guidance.

II. Overview

A. Overall Evaluation: Consensus Lifecycle

Consensus Lifecycle aims at elucidating the possible customizable intervals among the journey of transactions. A transaction in blockchain represents action upon the on-chain data, covering the execution of instructions like storing, querying, and editing. When a network user aims at interacting with others, one or multiple transactions are created, propagated, validated, and eventually confirmed on the network. Blockchain systems may vary in the realization of these steps, but basically all of them undergoes these stages to agree on the validity of transactions and the modification of states. Therefore, we call the stages a transaction pass through as the Consensus Lifecycle. A Consensus Lifecycle start with the

Identify applicable funding agency here.

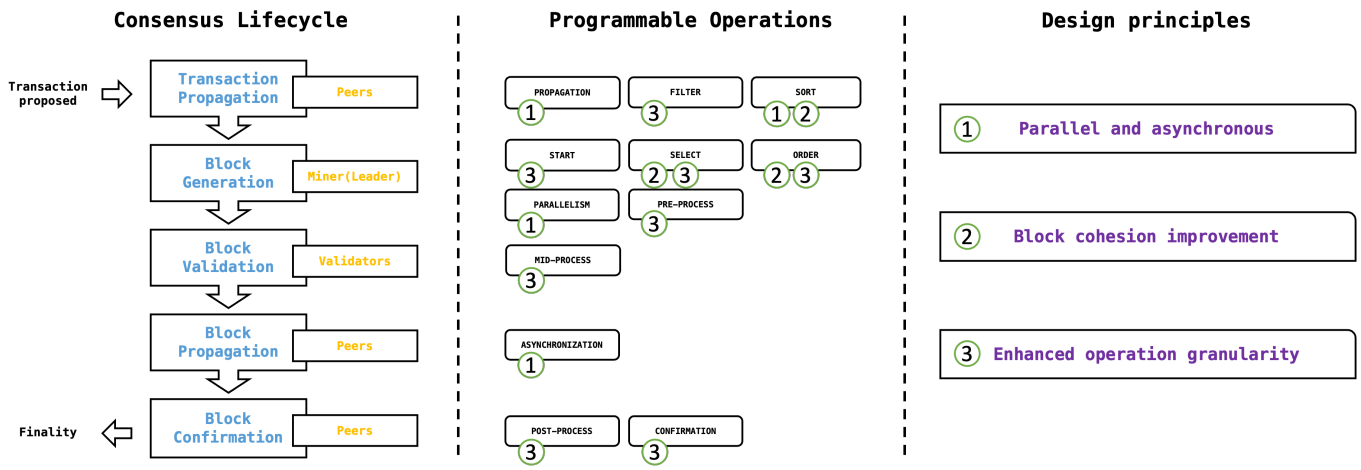

Fig. 1. Rethinking & Reassessing

proposal of a transaction and ended when the transaction and related state modification are recorded. Five common stages can be identified:

Stage 1: *Transaction Propagation*: The proposed transaction is propagated to the validating peers engaged in the Consensus Lifecycle. Peers filter the received transactions and may sort them into different schemes for possible tailored consensus.

Stage 2: *Block Generation*: When certain rules are met, the leader (also known as the miner) node will generate a block to start up this round of consensus. According to the starting strategy, the generation of a block may not require the end of last Consensus Lifecycle. Leader node will compliance a set of rules upon transaction selection and order in the block. Some execution procedure may include here as pre-process before assembling the block. All the operations adopted by the leader should be reproducible for followers, or the block would be abandoned during later stages because of the inconsistency.

Stage 3: *Block Validation*: This crucial step utilizes consensus algorithm, a defining characteristic of the blockchain system. All the validators would attest the block and reach an agreement on the block through steps of consensus. Afterwards, the block is regarded valid and executable. At this stage, specific blockchains activate their smart contract engines to execute transactions, producing a set of results that are attached to the block for subsequent confirmation.

Stage 4: *Block Propagation*: The valid block is propagated to peer nodes, ready for the update. Concerning the sequential execution to this stage and parallel strategy leveraged above, this stage may cause bottleneck and asynchronous strategy is adoptable.

Stage 5: *Block Confirmation*: After receiving the valid block, a peer will commit it to its own replica of ledger and renew its replica of state, which may include some execution. The block would be considered confirmed until its update meets some policies and is regarding as the alignment of the whole network. The Consensus Lifecycle comes to a finality at this stage, where the transactions in the block and related actions are recorded and cannot be reverted.

Transitions from one stage to another marked a shift of target, where the roles engaged with the stage will adopt the system-specified strategies. Discussions could be divided into stages.

B. Consensus Lifecycle for Existing blockchain systems

Ethereum: For Transaction Propagation, users initiate transactions, which are then propagated to the Ethereum network. These transactions can include simple transfers of Ether (the native cryptocurrency) or more complex interactions with smart contracts. Ethereum uses a peer-to-peer network to propagate transactions. Nodes broadcast transactions to their peers, which in turn relay these transactions further. Nodes in Ethereum may apply filters to transactions based on certain criteria like gas price. Transactions with higher gas prices are often prioritized. There isn't a distinct sorting strategy at this stage in Ethereum. However, miners may sort transactions in the block based on the transaction fee offered.

For Block Generation, Miners (or validators in the case of Ethereum 2.0's Proof of Stake) create new blocks by including a set of transactions. In Ethereum's original Proof of Work system, miners start creating a new block after finding a solution to a cryptographic challenge. In Ethereum 2.0's Proof of Stake, validators are randomly selected to propose blocks. Miners or validators choose transactions based on transaction fees (gas price). They generally prioritize transactions offering higher fees. There is a validation of transaction format and nonce before including it in a block.

For Block Validation, in Proof of Work, validation involves checking the miner's solution to the cryptographic puzzle. In Proof of Stake, it involves attesting to the proposed block by other validators. Smart contracts may be executed as part of transaction processing, and their results are included in the block.

For Block Propagation, once a block is validated, it is propagated across the network and each node updates its copy of the blockchain ledger.

For Block Confirmation, a block is considered confirmed after several subsequent blocks are added, reducing the likelihood of a fork. Ethereum typically considers a block final after 12 confirmations (or blocks) but this can vary depending on the network's state and the level of security required.

Hyperledger Fabric: For Transaction Propagation, the transaction propagation begins when a client submits a transaction proposal to the endorsing peers. Endorsing peers execute the transaction against their current state database (without updating it) and endorse the transaction by returning a proposal response to the client. Fabric utilizes "channels" to facilitate private transactions and data segregation. Each channel represents a separate blockchain ledger; transactions are only visible to the nodes that are participants of that channel.

For Block Generation, once the transaction is endorsed, the client sends it to the ordering service, which doesn't need to know transaction semantics, batches transactions into blocks. The strategy here is not about transaction selection by a single leader, as in traditional blockchains, but rather about ordering transactions consistently across the network.

For Block Validation, Blocks delivered by the ordering service are distributed to all peers on the channel. Each peer validates the transactions within the block, ensuring they meet endorsement policies and have not been tampered with since endorsement. Fabric employs a pluggable consensus mechanism. The default is Raft, a crash fault-tolerant (CFT) consensus protocol. However, it can be configured to use other protocols like Kafka or even Byzantine Fault Tolerant (BFT) mechanisms.

For Block Propagation, after validation, the block is committed to the ledger on each peer. The propagation in Hyperledger Fabric is controlled within the context of channels. Information about transactions is confined to the members of the channel, offering privacy and confidentiality.

For Block Confirmation, Once a block is committed to the ledger on a peer, the transactions within that block are considered confirmed. The finality in Fabric is straightforward as there is no possibility of blockchain forks, a common issue in many other blockchain systems.

C. Programmable Operations

Looking through the lens of the five-stage Consensus Lifecycle, Ethereum and Fabric could be concluded into the same pattern. At each stage, we extract some programmable operations for possible customizing upon the strategies.

1. Transaction Propagation:

PROPAGATION: For partitioned blockchain, nodes are grouped to go through the rest stages. Even though nodes in a network are physically interconnected, they possess a logical boundary for communication partitioning. Here exists a propagation strategy for nodes to designate peers that should receive the proposed transactions. Usually, propagation strategy is formed by service needs and business relationships. Considering business changes and ongoing maintenance, propagation can be adjusted according to the requirements of new transactions. For example, the stakeholders involved in a certain type of transaction changed.

FILTER: Filtering the received transactions including security and functional consideration. Security consideration involves validating transaction integrity, thwarting potential spam and DoS attacks. Also, according to the business logic, one node can also set rules upon the transaction semantics optimize the space of transaction pool.

SORT: The accepted transaction would be collected into the transaction pool. To support the parallel consensus, multi-pool is adoptable for each individual consensus. This strategy will allocate transactions into different transaction pools according to their read-write conflicts and business logic, reducing the mixing of transactions.

2. Block generation

START: Block packaging strategies vary depending on transaction time-sensitivity, including immediate packaging upon arrival, fixed interval waiting, or awaiting specific transaction combinations. Block formation and consensus initiation occur only upon fulfilling these predefined conditions.

SELECT: The new block is determined as a selection of transactions among the pool of pending transactions. Usually, leader owns the right of selection and fairness is enforced. The selection strategy could be further extended to support specific business needs. Thus, transactions involved in the block has a natural correlation and block is not just serve as a batch of transactions.

ORDER: To leverage the correlation brought by the combination of transactions in the block, giving them a certain order in the block to guarantee more complex execution engaged with more than one transaction. Since a business event is operated by multiple participants with multiple actions, "combination of transactions" offers higher granularity in the scheduling.

PARALLELISM: TODO

PRE-PROCESS: Process here refers to the read, handling, and modification of transaction itself. For permissionless blockchain, leader must execute the block first and generate a final state for later validation. For permissioned blockchain, execution of transactions could happen before block generation, leveraging the time pending in the transaction pool and reducing time delay in the subsequent stages. Thus, user should have the right to designate if transactions should be pre-processed according to the network condition and execution complexity of the transaction.

3. Block validation:

MID-PROCESS: When validators received the new block, they may process the transactions in it. It could be used for the validation, or as the output that update into the state after the block confirmation. The mid-process could be further extended to partially process the transactions. For example, according to the transaction combinations, aggregating and merging them into single propose for later record. It is worth to notice that, mid-process is happened during the consensus procedure, taking a large account of consensus time and may cause consensus delay. Thus, carefully choose mid-process is vital to the performance and stability of the consensus lifecycle.

4. Block Propagation:

ASYNCHRONIZATION: The phase of waiting for commit updates can be asynchronous with the previous stages. Blocks have been parallelly generate and validated in the previous stages. However, they will obey a sequential on-chain stage here. To avoid the bottleneck may exist at the stage, asynchronous propagation could be adopted here. The committee is required to coordinate the selection and chaining of blocks.

5. Block Confirmation:

POST-PROCESS: If transactions include time-intensive smart contract execution, it could be set here after the block is recorded in the blockchain. Nodes will locally execute the transactions in the recorded blocks and the output is used for the update of its own state.

CONFIRMATION: Confirmation thresholds is a trade-off between security and speed, which also varied with specific scenarios and could be configured by the user.

III. Design Principles

- Parallel and asynchronous
- Block cohesion improvement
- Enhanced operation granularity

IV. Programmable Consensus

V. Implementation & Experiment

VI. Conclusion