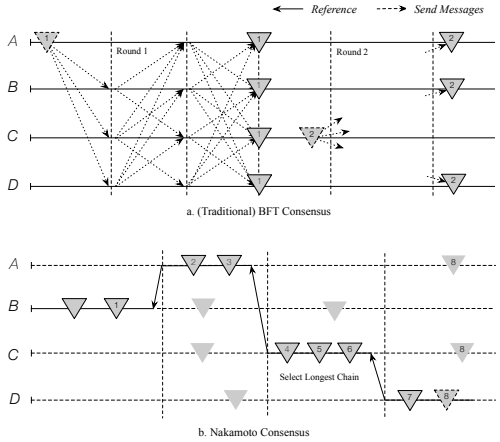
Poster: A Weak Consensus Algorithm and Its Application to High-Performance Blockchain

FastBox

Research Problem

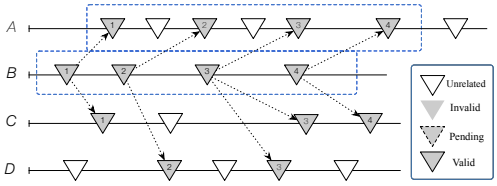
Blockchain systems adopting the BFT consensus algorithm and Nakamoto consensus (NC) algorithm suffer from low-performance issues due to massive communication or intensive computation.



Same principle. BFT-based and NC-based blockchain systems all require *strong consistency*, meaning that only one block is deemed as "valid" in different node in each round. This greatly constrains their overall performance since the procedures of conflict solving and total ordering are time-consuming.

Research Finding

We propose a new type of consensus mechanism, called *weak consensus*. Weak consensus guarantees that the relative sequences of blocks in one individual chain remain consistent with that in the other chains.



Example: The node *B* creates a serial of blocks $1, 2, 3, 4$. Our goal is to ensure that the relative order of $(B1 \rightarrow B2 \rightarrow B3 \rightarrow B4)$ is correctly maintained in nodes *A*, *C* and *D*. *(B1 represents the first block in node B.)*

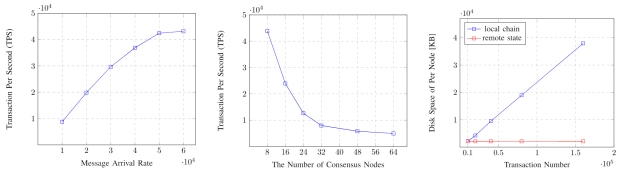
Detailed Algorithm

- ***Pre-prepare.*** The primary node receives the client requests and inserts the messages into local chain. Then, the node creates a *Pre-prepare* message to claim the relative position between two client messages.
- ***Prepare.*** A node receives the *Pre-prepare* message and checks the integrity, correctness, and validity. If the received *Pre-prepare* message passes the verification, the node updates his local-stored state and broadcasts the replied *Prepare* message to claim the correct relative position. Otherwise, the node aborts it.
- ***Commit.*** If any node receives a quorum $2f + 1$ of valid *Prepare* messages from peers within specified time interval, this node confirms the proposed decision by broadcasting a *Commit* message. When collecting more than $2f + 1$ *Commit* messages, the node transfers the state and replies to clients with updated state.

Complementary Mechanism. If a message (relative position) fails due to the lack of enough confirmation, the procedure of rebroadcast will be launched, and the counter increases each time of a retry. If the accumulated value is greater than the bound set in the counter, the node will accept the reversed relationship and rebroadcast it. If a node collects more than $2f+1$ *Commit* messages on the reversed position, the node replies to clients with updated state. Otherwise, the message will be aborted. On the other side, when the waiting time exceeds the predefined time-bound in the counter, the message will be aborted with sending a *timeout* message to the client.

System Evaluation

The average throughput of Sphinx reaches 43k TPS with 8 full nodes and drops to around 5000 TPS given 64 full nodes.



(a) TPS under Different Message Arrival Rates (b) TPS under Different Size of Nodes (c) Space Usage on Each Sphinx Node

- The throughput drops down as the number of participants increases.
- The throughput increases linearly (arrival rate <= 50k). The throughput flats out at around 43k TPS (arrival rate > 50k).
- The size of the local chain grows linearly with increased transactions.

Security Analysis

Relative persistence ensures that as soon as the relative position between two states has been confirmed by honest node, this relationship will ultimately be confirmed by every node in the network with a high probability.

Theorem 1. ***(Relative persistence)*** *If the relative position of two state y and x is accepted by the node N_i in iteration r and by the node N_j in $r+1$, respectively, their decisions on the relationship are the same.*

Liveness guarantees that all nodes eventually agree on a unique relationship *w.r.t* each chain.

Theorem 2. ***(Liveness)*** *If a correct relationship is committed in a honest node, then, every honest node will eventually accept such a relationship.*