Empowering Fault-Tolerant Consensus Algorithm by Economic Leverages

Igor Mazurok, Valeriy Pienko and Yevhen Leonchyk

Odessa I.I.Mechnikov National University, 2, Dvorjanskaja st., Odessa, 65082 Ukraine
igor@mazurok.com, vpenko@onu.edu.ua, leonchyk@ukr.net

Abstract. This paper describes an integrated parallel fault-tolerant consensus algorithm for systems of distributed processing and storage of information with low latency. An essential characteristic of this algorithm is the integration with an economic model, ensuring its sustainable development in accordance with the goals of functioning. The proposed algorithm is called WWH (What, Where, How much), because it allows for one pass of the protocol to obtain consistent solutions on the following issues: what information will be stored; to which place of the synchronized storage it will be recorded; determination of nodes reward for fair functioning. The algorithm is based on the ideas of the SBFT algorithms, Raft and the basic principles of the Computable general equilibrium to construct the internal economy of the system functioning. The algorithm assumes resistance to two types of errors - Byzantine errors and equipment failures.

Keywords: consensus, fault-tolerant algorithm, tokenomics, blockchain.

1 Introduction

Characteristic trend of modern information systems is the transition to a decentralized architecture. In this case it is important to provide the following requirements:

- performance;
- scalability;
- tolerance to various types of attack;
- immutable and tampering safety;
- logic consistency.

Recently to support such a set of characteristics blockchain technology has been used. Due to the decentralized nature the key item of these systems is the consensus mechanism - the ability to make agreed decisions and record the corresponding results in a distributed database, which is implemented as a set of identical linear information structures (ledger). In traditional systems based on blockchain, a consensus of the Proof-of-Work (PoW) type is used (Bitcoin, Ethereum etc.). However, for many applications this type of consensus is not acceptable due to poor system performance (and/or bandwidth) and allows the system to be temporarily in an undeterministic state (branching of the blockchain - fork). There are also other types of consensus.

Some of them provide greater performance, but are vulnerable and poorly scalable (Ripple, EOS, Stellar, NEM).

The evolution of modern blockchain systems began with Bitcoin. His mythical author Satoshi Nakamoto [1] offered a mechanism that takes into account almost all aspects that determine the dynamics of this system. This ensured the stability of the system's behavior throughout the entire lifecycle. However, the Bitcoin functionality does not satisfy modern challenges. An example of a system that provides a higher degree of flexibility and variation is Ethereum. A key feature of Ethereum is the support of smart contracts, which allowed its authors to call their system World Computer. The emergence of Ethereum was the stimulus for the huge number of blockchain startups initiated by ICO. But the experience of implementing such startups has shown that the PoW mechanism used in the Etherium does consider the specific requirements of their functioning and development.

Other types of consensus suffer with essentially the same drawback - by providing individual key requirements, they do not allow to consider the whole set of requirements.

Solution to this problem seems to us as follows. Instead of searching for a certain consensus mechanism that would provide the full range of required system properties, we suggest using a consensus mechanism, whose architecture contains managed components that allow the main mechanism to be adapted to specific conditions. These components are integrated into the consensus protocol, providing one or another model of the economic behavior of the nodes and the system as a whole.

This methodology was tested in the implementation of the blockchain system to support the public key architecture (PKI). The consensus mechanism implemented within this system has to meet the following requirements:

- support for enterprise-scale systems, i.e. the system sets a high entry threshold for new nodes and, as a result, the number of network nodes is limited to several hundred;
- the system must provide a high response rate to the client request;
- processing nodes of the system are functionally equal, ensuring its decentralization;
- reliability of operation, consisting in resistance to Byzantine threats and the fork occurrences;
- transactions in the system are processed separately, not combined into blocks that reduces the guaranteed transaction processing time due to high overall system performance (there is no common transaction pool).

The relevance of the presented work is justified by the necessity to overcome the limitations associated with the popular types of consensus PoW and PoS during modern decentralized systems development.

2 Related Works

Listed above features make us seek a suitable consensus algorithm in a more general context of distributed computing systems. In this sense, the Paxos [2] algorithm is the

most discussed in academic publications. However, attempts to use it in real systems cause difficulties. To avoid them, various modifications of this algorithm have been proposed. In particular, Raft [3] was designed to support modularity, which in turn led to its clarity and ease of implementation. It contains sufficient tools to support a ledger and is focused on the use of the leader node and provides only CFT. However, considering the above requirements, this is not enough, since Byzantine threats may arise in the system. There is known modifications of the Raft algorithm that overcome this limitation [4, 5]. Some features of these modifications were used in this paper. However, the known modifications lack incentivization for fair nodes. This incentivization is often implemented as a separated subsystem. There are some attempts to inject adequate economic mechanism into consensus algorithms (e.g. [6]) but they still have no enough adjustability to certain economical model. But we offer to integrate economic logic into the consensus mechanism.

3 Consensus with integrated economic

As mentioned above, the main mechanism to obtain the required properties of decentralized system is to integrate into the consensus logic a model of economic behavior which provides necessary adjustment of the system dynamics. This model is usually called tokenomics, since it is based on the use of tokens - specialized cryptocurrency.

The system functions by transmitting and processing asynchronous messages between peer-to-peer network nodes. In effective tokenomics it is important to distinguish between the roles of nodes during their intercommunications:

- a regular node that signs and stores certificates;
- a receiving node that receives a certificate signing request (CSR) from a client, it also receives payment from the client and rewards the nodes;
- a leader node that coordinates messaging between other nodes.

To find the optimal parameters for the system functioning we define a model containing information about history of actions performed by the participants of the system.

Such a model can naturally be defined in form of an oriented weighted graph, whose nodes are the nodes of the system (or rather their participation in a certain stage of network communication), communications — messages or requests marked with information accompanying such communications.

The route in this graph describes the sequence of steps in the communication protocol, which leads to the achievement of the result. At this point, you can determine the share of the participant's rewards based on his participation in the protocol. All information necessary for such calculations is recorded in the blockchain of the system together with a certificate.

The tokenomics of the entire system functions as a cyclic process, each stage of which consists of the following steps:

1) When a CSR appears, each node i makes a bet, ensuring its honest and reliable functioning. The sum of these rates s and client certificate payment S_{cert} form the total budget of the certificate:

$$B_{CSR} = S_{cert} + \sum_i s_i$$

2) Nodes perform consensus protocol;
3) The budget of each certificate is distributed in accordance with the share of each node participation and its rate. As a result, the node reward is lost (reduced) if the node is an attacker or has failed (crashed).

At the moment we offer the following models of the participant's bid value:

1) Free Fixed Stake Model: All nodes can either play at a fixed rate σ or not play at all;
2) Free Gambling Stake Model: A model that allows you to choose bets in a certain range [α; β], or not to play at all;
3) Force Fixed Stake Model: Participants play only at a fixed rate σ;
4) Force Gambling Stake Model: A model that obliges to play, however, allows you to independently determine the value of the bet in the range [α; β].

The system must accept one of these models for all participants . Given the above notation the differences between models can be represented by the following Table 1.

Table 2. Stake differences between models.

	Free Stake	**Force Stake**
Fixed Model	$s_i \in \{0, \sigma\}$	$s_i = \sigma$
Gambling Model	$s_i \in \{0\} \cup [\alpha; \beta]$	$s_i \in [\alpha; \beta]$

A comparison of these models allows us to make some preliminary conclusions about the results of their use. Forced rates make it economically unprofitable node downtime. This is important for the consensus using the majority principle. Gambling Model allows you to enter the concept of the node reputation into functioning of the system, which means the degree of its reliable operation. Trusted nodes can afford to bet larger. Thus, the option Force Stake Gambling Model is more adaptive to use in environmental conditions with possible equipment failures and communication protocol algorithms.

The optimal model of tokenomics should provide an economic motivation for the correct execution of all processes implemented by the system: extract the certificate, write to the ledger and maintain its integrity.

For a detailed description of the tokenomics parameters, we describe the communication protocols used in the system.

3.1 The Protocol for Issuing a Certificate Including the Node's Contribution

Next, the certificate issuance protocol derived from the SBFT algorithm [5] will be described. All node's messages transmit information in a cryptographically signed form in order to avoid its distortion during the transmission by the communication channel.

In order to provide sustainable functionality, the system has to satisfy the next requirement: the total node number *n* must be greater than *2f+c* where

- *f* - the maximum number of faulty (byzantine) nodes;
- *c* - the maximum number of crashed nodes (at least within established period of time).

At each step, one of the nodes plays the role of a leader. Its main task is to relay messages received from ordinary nodes. At every communication stage the current *leader* gets not less than *2f+1* uncorrupted messages with no more than *f* faulty messages among them. Thus, there are always at least *f+1* identical messages which is sufficient for the majority of considerations.

The protocol has several steps and can be presented schematically (see Fig. 1).

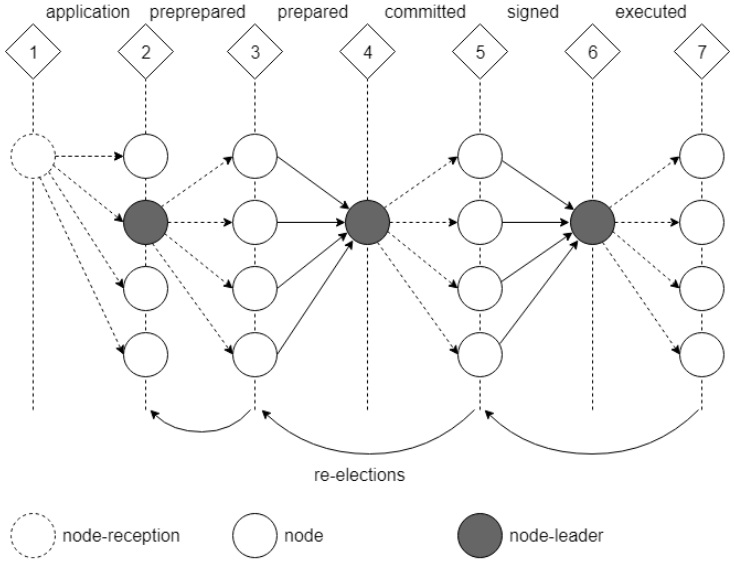

Fig. 1. The certificate issuance protocol.

At each step, the nodes send messages to each other with the following unified structure: From-Message-To. Below is a description of the actions at each step. Step is a transition from one state to another, for example, the transition from the first state to the second one is indicated by 1-2.

1-2) *node*-reception sends to all nodes the *applicationRequest* message (a request to receive a certificate with proposed payment Scert) with following content:
From: *node-reception* **To:** *all nodes* **Message:** *PK_applicant, Payment*
where *PK_applicant* is a client public key, *Payment* is amount of tokens. This message may have some additional data. All nodes record source and time of receiving the request.

Possible attack: Availability of payment information prevents spam-sending unpaid requests.

2-3) After receiving the *applicationRequest* message, current *leader* checks *Payment* and sends the following *prepreparedRequest* message to all *nodes*:
From: *node-leader* **To:** *all nodes*

Message: *node-reception,PK_applicant, Payment, FFLN, t_leader, LId, VD*
where *FFLN* is the first free number in the ledger of the current node-leader that was not previously reserved. Thus, a new certificate will be added to the end of the ledger. The node leader also transmits *t_leader* — the term number (used to select a leader for re-election) and *LId* — leader's identifier.

At this step, the node-leader may initiate a selective check of the the ledger block relevance. To do this, a *VD* field (verification data) is added to the *prepreparedRequest* message, containing a request for some data from the ledger blocks.

If the node-leader did not perform its functions - did not send a message within a predetermined time (time-out) at this step or sent an incorrect message - the node initiates re-election. Re-election starts when a sufficient number of nodes initiate re-election (the re-election procedure is described below). An example of an invalid message is the *prepreparedRequest* message, in which the public key does not match the public key in the initial node-reception request.

Possible attack: To compromise a node-leader, a node-reception can send a different message to it. To eliminate this, the node-leader additionally sends a message which it received from node-reception to the other nodes. All the nodes in this step verify this message with one they received directly from the node-reception and, in case of any discrepancies, ignore this request, terminating the certificate issuance protocol.

3-4) Each node sends the preparedRequest message to the current node-leader:
From: *node* **To:** *node-leader* **Message:** *preparedRequest, LId, NId*
NId is the node number that generated this message.

As soon as the node-leader received the first *2f+1 preparedRequest* messages, the transition to the next step takes place.

4-5) Node-leader passing *2f+1* message in the *commitedRequest* message as an array.
From: *node-leader* **To:** *all nodes* **Message:** *preparedRequest*[*2f+1*]

Each node performs a validation of the node-leader term, i.e. *t_leader* is not less than its node term. Also, the *FFLN* relevance check is performed, providing the entry into the longest ledger.

Any violation leads to the initiation of a re-election of the leader by the node, and if a sufficiently large number of such failures are accumulated, this will lead to the re-election of the node-leader.

Further according to the majority rule node prepares the following information for inclusion in the message: *node-reception, PK_applicant, Payment, FFLN, LId*. The majority rule says that the value of the field is included in the message if it is encountered no less than in the *f+1* elements of the *commitedRequest* message array (the message of the node itself is also taken into account).

Accumulated to this point in the communication messages information allows you to determine which nodes acted efficiently and correctly in step 3-4. This information is analyzed and recorded as a binary vector, *Participation*_1, in which the *i*-th element

is 1 if the node was among the first *2f+1* nodes that sent the *preparedRequest* message, and its value coincided with most similar messages from other nodes.

5-6) Next, each node sends the following message to the signedRequest:

From: *node* **To:** *node-leader*
Message: *node-reception*, *PK_applicant*, *Payment*, *FFLN*, *NId*, *LId*, *CId*, *Participation*_1, *VDR*

NId is the identifier of the node that formed this message, *CId* is the identifier of the node from which the *commitedRequest* is received (*CId* may differ from *LId* if re-election occurred at the previous stage), *VDR* is the information sent in response to the *VD* request.

6-7) Node-leader accumulates *2f + 1 signedRequest* message and sends them in the message *executedRequest* in the form of an array.

From: *node-leader* **To:** *all nodes* **Message:** *signedRequest*[*2f+1*]

Information accumulated in the *signedRequest* array makes it possible to evaluate the participation of nodes in step 5-6 (in addition to the vector *Participation*_1) and honest storage behavior of the ladder (comparison of *VD* and *VDR*). This information is fixed in the form of two binary vectors *Participation*_2 and *DataSaving*, similarly to the vector *Participation*_1.

Information provided in these three vectors, as well as data on the participation of nodes as leaders, is sufficient to calculate the remuneration of all participants.

If there are *f+1* identical records in the message, their content is considered authentic and is written to the ledger by all nodes, and the reception-node sends the signed certificate to the Client.

Here, as well as at the end of steps 2-3 and 4-5, node-leader re-election is possible.

Since unfair storage of ledger is not economically justified, the nodes are interested in updating its status. For this, any node based on *FFLN* or *VD* (*VDR*) can request the node-leader for all previous blocks, starting with a certain number, or for missing certificates to update its ledger.

Possible attack: Malicious nodes can intensively send a ledger repair requests to the leader. To prevent collapse in this case, you can pay an additional fee for servicing such requests.

3.2 Fault Tolerant Election Protocol

As mentioned above, a specially chosen node-leader plays an important role. The need for re-election arises if a node for a sufficiently long time does not receive messages from the current node-leader or if it receives a message with violated cryptographic integrity. The election of a new node-leader is a multi-step messaging process with the robustness feature.

3.3 Rewards calculation

Upon successful completion of the protocol, all nodes have information about the number of messages from each other with confirmed accuracy. Node messages are considered reliable if they fall into the final majority vector. Current node-leader mes-

sages are considered confirmed if the node's response message is valid. To define rewards participant's contributions have to be counted:

1. The contribution of term node-leader for reaching a consensus -from *f* to *n* and for replicating the ledger from 0 to *n-f*;

2. The contribution of the ordinary node for reaching a consensus from 0 to 2 and for replication and storage – 0 or 1.

At different terms node i can play different roles and its total contribution C_i is summed over all terms. The weighted share of P_i's reward, taking into account the s_i rate for the Gambling Model, is:

$$P_i = \frac{C_i s_i}{\sum_i C_i s_i}$$

Thus, the node reward is determined by the formula: $R_i=[B_{CSR},P_i]$, where square brackets denote banking rounding.

4 Conclusions

Presented algorithm allows the participants (nodes) of the decentralized system to come to a common opinion (consensus) on the contribution of each other directly during consensus execution and does not require additional nodes that control the process. In particular, the proposed approach was implemented within a decentralized system supporting PKI and allowed to consider requirements of both consistency and needed economic features.

The method of rewarding nodes described above is one of the possible options and depends on the specific type of tasks and goals of the functioning of the system. The search for optimal parameters of tokenomics for effective work is an open question. Due to the fact that these parameters in the proposed scheme are expressed in the protocol messages, it is possible to build a multi-factor simulation model to achieve the optimal configuration.

References

1. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2009). https://bitcoin.org/bitcoin.pdf.
2. Lamport, L.: The Part-Time Parliament. ACM Transactions on Computer Systems. 16 (2): 133–169 (1998).
3. Ongaro, D., Ousterhout, J.: In Search of an Understandable Consensus Algorithm (2014). USENIX Annual Technical Conference. https://ramcloud.stanford.edu/wiki/download/attachments/11370504/raft.pdf
4. Wang D., Tai, N., An, Y.: Byzantine Fault Tolerant Raft. Stanford Computer Science Department (2018). http://www.scs.stanford.edu/17au-cs244b/labs/projects/wang_tai_an.pdf
5. Gueta G.G., Abraham I., Grossman S., Malkhi D. et al.: SBFT: a scalable decentralized trust infrastructure for blockchains (2018). https://arxiv.org/abs/1804.01626
6. Amoussou-Guenou, Y., Del Pozzo, A. et al.: Correctness and Fairness of Tendermint-core Blockchains. IACR Cryptology ePrint Archive (2018). https://arxiv.org/abs/1805.08429