International Journal of Internet, Broadcasting and Communication Vol.16 No.4 100-109 (2024)
http://dx.doi.org/10.7236/IJIBC.2024.16.4.100

IJIBC 24-4-11

Enhancing Fault Tolerance in Private Blockchain Networks through Modified Raft Leader Election Algorithm

Youn-A Min

Professor, Applied Software Engineering, Hanyang Cyber University, Korea
yah0612@hycu.ac.kr

Abstract

In this paper, I study the Raft leader election process to enhance fault tolerance in a network composed of minimal nodes by considering various failure situations that may occur during consensus in a private blockchain network. In the process of processing network partition situations, node failure situations, and leader node failure situations, an Activity Score variable is set, so that the platform is configured with the minimum number of nodes in a network partition situation or node failure situation, and when successful leader election is required, it can be modified. Leader election is conducted according to the Raft algorithm, and leader node election and network failures are minimized based on trust to enhance fault tolerance even in a platform environment where the minimum number of nodes is operated. Excellent performance of over 12% on average was confirmed.

Keywords : *Block-chain, consensus algorithm , Smart Contract, Raft*

1. Introduction

With As technology advances and the amount of data produced through various devices increases, interest in various industrial applications using data is increasing [1].

Markets and markets, a global market research firm, announced that the global data fabric market size is expected to grow at an average annual rate of about 26.3% from $1 billion (KRW 1.399 trillion) in 2020 to $4.2 billion (KRW 5.8758 trillion) in 2026. Accurate and transparent data management is necessary to increase data efficiency and ensure data quality [2].

Blockchain technology stores data that can be distributed and shared. Blockchain has the characteristic of increasing the accuracy and transparency of data management and processing between multiple nodes on a distributed network [3-4]. Blockchain can be classified into public blockchain and private blockchain depending on the characteristics of participating nodes. In the case of public blockchain, there are no restrictions on participating nodes and the disadvantage is that block verification takes a lot of time, so blocks between nodes can be trusted and agreed upon. If chain connection is required, a private blockchain platform is used [3]. Raft is a consensus algorithm used in distributed systems such as Hyperledger Fabric that enable connections between multiple consensual nodes and It consists of a leader and followers and is a concise, easy-to-understand, highly available and consistent consensus algorithm [4-5].

Manuscript Received: September. 11, 2024 / Revised: September. 17, 2024 / Accepted: September. 22, 2024
Corresponding Author: yah0612@hycu.ac.kr
Tel: +82-2-2290-0872, Fax: +82-2-2290-0872
Professor, Applied Software Engineering, Hanyang Cyber University, Korea

Recently, blockchain technology has been applied and utilized in various places, including industries and governments, to efficiently utilize numerous data. Among them, private blockchain enables distributed sharing of data between nodes based on trust, making it possible to use it universally to utilize data between organizations [5].

In this paper, we present a modified processing process by modifying the leader election process of the Raft consensus algorithm used in the private blockchain platform, and show that the performance of leader election time, replacement frequency, transaction throughput, and network stability is improved through the modified processing process and prove it

2. Related work

2.1 Private Blockchain

Private blockchain is a blockchain platform where the access and permissions of nodes that can access the network are limited. It is mainly used internally within agreed-upon companies or organizations, and only authorized participants can access the network and verify transactions[4-6].

The characteristics of Private Blockchain include restricted access to nodes and high performance. Unlike Public Blockchain, the number of participants is limited, so processing speed is fast and the consensus algorithm is relatively simple. In addition, data privacy can be protected because transaction details and data are not made public, and unlike public blockchains, central management is possible because it is operated by a specific manager or consortium[5-6].

Hyperledger Fabric is a representative platform for private blockchain. Hyperledger Fabric is an enterprise blockchain platform and has a modular architecture that allows customization when sharing data with multiple layers to suit various industries and application fields. The characteristics of Hyperledger Fabric include the use of modular architecture, permission for transactions between specific groups of nodes through channels, and use of chain codes[1, 3, 6].

Existing research and use cases of Private Blockchain include services such as healthcare, supply chain management, financial services, cybersecurity, and NFT. Recently, the concept of Block-GPT has also been studied to track anomaly detection of real-time data based on LLM, creating a tracking representation of block activity and effectively identifying abnormal transactions in Ethereum transactions. This allows us to significantly improve the security and transparency of financial transactions. Additionally, it is utilizing financial (DeFi) applications by converging artificial intelligence and blockchain technology [2, 6].

2.2 Hyper ledger Fabric

The consensus algorithms used in the Hyperledger Fabric platform can be classified in various ways depending on the modular consensus structure of Hyperledger Fabric. The most commonly used consensus algorithms in Hyperledger Fabric are Kafka, Raft, and Solo [3,5-6].

Kafka is a consensus algorithm mainly used in early Hyperledger Fabric networks. Kafka is used as a message broker system that guarantees the order of transactions and is used for ordering services. Raft is currently the most widely used distributed consensus algorithm in Hyperledger Fabric and is composed of leader and follower nodes, so it requires technology for the leader election process and leader absence avoidance. Solo is a single-node ordering service and is primarily used in development and test environments[3, 4].

The consensus process of Hyperledger Fabric is processed as shown in Figure 1[3, 6].

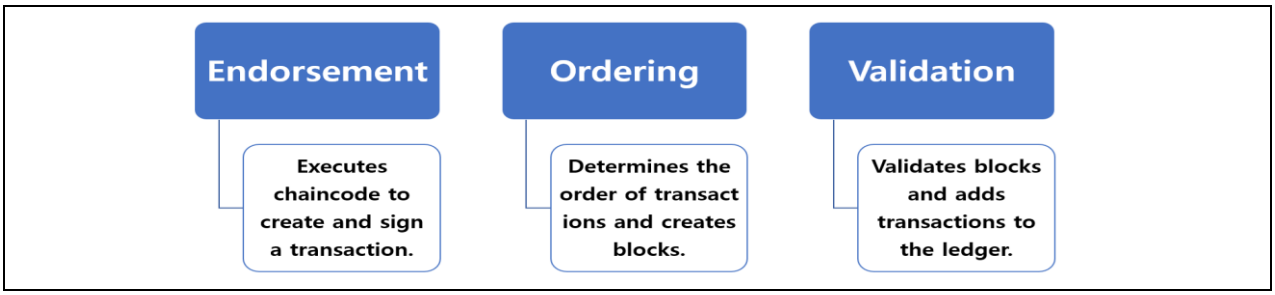


Figure 1. Consensus Process of Hyperledger Fabric

To change the consensus algorithm in Hyperledger Fabric, simply adjust the settings in the configuration file and run the ordering service [7, 8].

2.3 Raft

The Raft consensus algorithm is a widely used consensus algorithm in private blockchain. It is simple and highly usable, but has drawbacks in use. The biggest drawback is leader dependence. Raft is a leader-based consensus algorithm and is highly dependent on the leader node. If a leader node fails, time is needed to elect a new leader, during which time system availability may be temporarily reduced [9].

Limited scalability is also one of the disadvantages. Raft achieves consensus through communication between leaders and followers, so communication overhead increases as the number of nodes increases. This can be a limiting factor for scalability in large networks. Additionally, processing time is delayed due to leader election failure. If a leader node fails, it takes time to elect a new leader, during which the consensus process is halted. This may result in delays in transaction processing. In addition, when the network is divided, there is a network division problem that can occur when each partition attempts to independently elect a leader, and there is a problem with node failure handling where multiple nodes can simultaneously fail and the consensus process can be halted[9-10].

The Raft consensus algorithm is widely used due to its simple and efficient design, but it has disadvantages such as leader dependency, limited scalability, poor performance, leader election delay, network partitioning problem, and handling of multiple node failures. When using Raft in Hyperledger Fabric, the network must be designed and operated with these shortcomings in mind. The leader selected in the Raft consensus algorithm periodically transmits a 'heartbeat' message, and if each node does not receive the leader's 'heartbeat' message for a certain period of time, it times out, and the timed-out node becomes a candidate node and elects a new leader. Candidate nodes request votes from other nodes, and if (N+1)/2 nodes agree when voting, they are elected as leader. The elected leader maintains his/her leader status by sending 'heartbeat' messages regularly [11-14].

3. Modification of Raft Leader Election to Minimize Fault Tolerance

3.1 Modification of Raft leader selection process

The Raft consensus algorithm is designed to provide fault tolerance, meaning that the system can operate

normally even if some nodes fail. However, fault tolerance issues can arise under certain circumstances, and these issues can have significant implications for leader election.

In this paper, this paper proposes a modified Raft leader election algorithm to quickly elect a leader node while maintaining trust in various failure situations that may occur in a private blockchain platform.

Network split situations, node failure situations, and leader failure situations can be considered as various failure situations that can occur in a private blockchain platform. A network partition situation is when a cluster is divided into several separate parts, the part containing a large number of nodes can continue to elect a new leader and proceed with work, but a small number of nodes (nodes 3 to 3) have an error when electing the leader If this occurs, a leader cannot be elected and work cannot proceed, making it difficult to maintain data consistency. Node Failure is a situation in which some nodes in a cluster fail and no longer respond. Several nodes in the network may fail, making normal leader election impossible. In this case, as long as a majority of nodes (the majority) are still operating normally, the cluster can elect a leader and process tasks normally. However, if more than a majority of nodes fail, the cluster will not be able to elect a leader, which may lead to service interruption. there is. A leader failure situation means that the current leader node fails. When the leader fails, one of several follower nodes goes through the process of being elected as a new leader, but in this case, there is a brief service delay and a new leader is elected and errors may occur during the process.

This paper proposes a modified leader election algorithm to enhance fault tolerance in a blockchain platform with minimal nodes. During the processing process, an Activity Score variable is placed, and when the platform is configured with the minimum number of nodes in a network split situation or node failure situation and successful leader election is required, the leader node election is conducted according to the modified the Raft algorithm, and the leader node is elected and trusted based on trust. By minimizing network failures, fault tolerance is minimized even in a platform environment where the minimum number of nodes is operated.

The processing process proposed in this paper is as follows the Raft leader election process can be written in pseudocode as shown in Table1.

Table 1. Pseudo Code for Proposed Raft Leader Selection Process

Program start
Initialize the Raft-Node class Use class Raft-Node:, def__init__(self, node_id, nodes) and define Variable,. Calculate activity variables Use function calculate_activity_score(node):
Use factors such as transactions processed, blocks created, successful leader tenures, network availability, response time, and data consistency
Election begins, define function start_election(nodes):..request_vote(nodes)
Request to vote, define function request_vote(nodes):..
Receiving votes, define function receive_vote(candidate, node):
Heartbeat transmission define function send_heartbeats(leader, nodes):
Reset election timer define function reset_election_timer(), and Set election timer to a specific period
Update activity variables define function perform_transaction(node)
Simulate activity variables and restart elections (elect new leader), define function simulate_activity_and_restart(nodes):
Main execution define.

The contents of Table 1 are as follows. First, the Raft-Node class is initialized, each node has a unique node_id, and the list of nodes that are the leader candidate list is initialized. During the initialization process, activity variables are initialized. Afterwards, activity variables are calculated through calculate_activity_score( ), taking into account factors such as the number of transactions processed, number of blocks created, number of successful leader terms, network availability, response time, and maintenance of data consistency. The election begins, start_election ( ) is executed, nodes are converted to candidate state, and the election begins. At this time, a voting request is sent to each node including its Activity Score. Afterwards, the node requested to vote through a voting request through request_vote() compares its score with the candidate node's score and votes. When the vote is received, receive_vote() is executed, and if the vote is received and more than a majority of the votes are received, the leader becomes the leader and elected and transitions to leader status. After the leader is elected, heartbeats are sent through send_heartbeats(), and the leader maintains leader status by periodically sending heartbeat messages. The election timer is reset when the leader's term ends, and the election timer can be set to allow the election to start again after a certain period of time. To update activity variables, perform_transaction(), create_block, check_availability(), respond_to_request(), maintain_consistency(), etc. are presented in pseudocode, and activity variables are updated through these. Finally, simulation of activity variables and election begins. After simulating activity variables in all nodes, the election timer starts and the election is restarted.

Components of activity variables applied when electing a leader include measuring the number of transactions processed by the node, the number of blocks created by the node, the number of successful leader terms, and the node's availability time and data consistency maintenance time in the network.

Each time a transaction is processed, it indicates how many transactions the node has processed, and a high number of transactions processed can lead to a high Activity Score. Each time a block is created, the reliability of the leader can be measured by measuring the number of blocks created by the node and counting the number of times a block has been successfully created. Additionally, the number of successful completions of a leader's term can be considered by measuring whether the implementation was successful each time the leader's term ends. Additionally, by measuring the node's network availability time and data consistency maintenance time, whether there was no conflict or whether it responded quickly can be used as a performance evaluation indicator.

For this performance evaluation, a fixed weight is assigned to each activity through smart contract to calculate the total score.

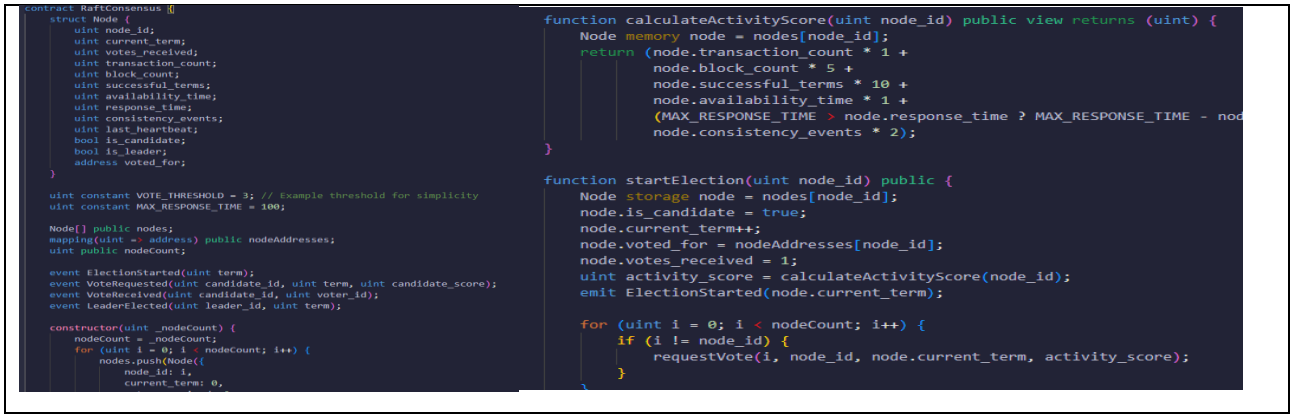
Table 2 shows the weight Variables for each activity.

Table 2. weight Variables for each activity

Number of transactions processed: 1 point/transaction
Number of blocks created: 5 points/block
Number of successful leader terms: 10 points/term
Network Availability: 1 point/hour
Response Time: -1 point/second (shorter response time, higher score)
Maintain data consistency: 2 points/consistency event

Figure 2 is a part of the smart contract creation screen that calculates the activity score by considering

the activity weights in Table 2 and reflects them in the leader election process.



```
contract RaftConsensus {
    struct Node {
        uint node_id;
        uint current_term;
        uint votes_received;
        uint transaction_count;
        uint block_count;
        uint successful_terms;
        uint availability_time;
        uint response_time;
        uint consistency_events;
        uint last_heartbeat;
        bool is_candidate;
        bool is_leader;
        address voted_for;
    }

    uint constant VOTE_THRESHOLD = 3; // Example threshold for simplicity
    uint constant MAX_RESPONSE_TIME = 100;

    Node[] public nodes;
    mapping(uint => address) public nodeAddresses;
    uint public nodeCount;

    event ElectionStarted(uint term);
    event VoteRequested(uint candidate_id, uint term, uint candidate_score);
    event VoteReceived(uint candidate_id, uint voter_id);
    event LeaderElected(uint leader_id, uint term);

    constructor(uint _nodeCount) {
        nodeCount = _nodeCount;
        for (uint i = 0; i < nodeCount; i++) {
            nodes.push(Node({
                node_id: i,
                current_term: 0,
```

```
function calculateActivityScore(uint node_id) public view returns (uint) {
    Node memory node = nodes[node_id];
    return (node.transaction_count * 1 +
            node.block_count * 5 +
            node.successful_terms * 10 +
            node.availability_time * 1 +
            (MAX_RESPONSE_TIME > node.response_time ? MAX_RESPONSE_TIME - nod
            node.consistency_events * 2);
}

function startElection(uint node_id) public {
    Node storage node = nodes[node_id];
    node.is_candidate = true;
    node.current_term++;
    node.voted_for = nodeAddresses[node_id];
    node.votes_received = 1;
    uint activity_score = calculateActivityScore(node_id);
    emit ElectionStarted(node.current_term);

    for (uint i = 0; i < nodeCount; i++) {
        if (i != node_id) {
            requestVote(i, node_id, node.current_term, activity_score);
        }
```

Figure 2 Part of Smart Contract

3.2 Performance evaluation

Set up an experimental environment so that the existing leader election algorithm and the modified algorithm can each be run in the same network environment. The same number of nodes, network configuration, and transaction throughput are used in each experiment. For this performance evaluation, we created virtual data of 100 customers used by Organization A, and based on that data, we would like to compare and evaluate the existing leader election method of the Raft consensus algorithm and the modified leader election method. The modified method evaluates performance by electing a leader by considering the node's Activity Score.

Table 3. Experiment environment and environmental variable settings

• Hardware: CPU: 8 cores, RAM: 8GB, SSD: 1TB
• Software: Hyperledger Fabric v2.2, JMeter v5.4.1
• Network configuration: 10 organizations, up to 35 nodes in each organization.
• VM environment: VMware, memory 8GB, dual mode
• Number of nodes: Consists of 3-5 nodes depending on the experimental scenario. / Each node is set to simulate the Raft consensus algorithm
• Default data amount: data of 100 customers.
• Experiment time: Each experiment lasts 10 minutes.
• Election Timeout: The leader election timeout range is set to a random value between 1 and 2 seconds.
• Calculation based on number of transactions processed, number of blocks created, number of successful leader tenures, network availability, response time, and data consistency maintenance events.
• Response time: Set to a random value between 0 and 2 seconds.

Performance evaluation indicators can measure leader election time, leader replacement frequency,

transaction throughput (TPS: Transactions Per Second), network stability (number of consistency maintenance events), response time, and system availability.

Regarding the above configuration, performance evaluation can be measured through jmetter and leader election time, leader replacement frequency, network stability, and system availability can be measured through the evaluation results. First, the leader election time detects the leader election event and records the leader election start time and leader election completion time to check performance. Leader election time can be measured using the formula leader election completion time−leader election start time.

Leader replacement frequency is a measure of the number of times a leader is replaced over a certain period of time. To measure this, the leader replacement event is detected, the number of leader replacements is recorded, and the leader replacement frequency is calculated to check the replacement frequency. Leader replacement frequency can be measured using the formula Number of leader replacements / Measurement period. Network stability is expressed as a percentage of the time the system operates normally, and can be evaluated through network error rate and transaction failure rate. Transaction success rate can be measured using the formula (number of successful transactions/total number of transactions) * 100%, and transaction error rate uses the formula (number of failed transactions/total number of transactions) * 100%. Additionally, network downtime can be calculated to measure the percentage of time the system operates abnormally, which can be measured using the formula (downtime/total operating time) * 100%.

System availability refers to the percentage of time that system requests can be responded to without error, and is evaluated through system uptime and downtime. System operation time is the total time the system operates normally and uses the formula of total operation time - downtime. System availability can be measured using the formula (uptime / total operating time) * 100%

As a result of measuring through J-meter and calculating the measurement results using formulas, the following results can be measured as a result of modifying and applying the Raft consensus algorithm in a private blockchain with 100 data per node and a platform composed of 3 to 5 nodes. Apply and measure the contents of A to F for performance indicators.

Table 4. Performance Indicators and Performance Evaluation Results

-A= Leader election time (micro second) - B= Leader replacement frequency (times/hour)
-C= TPS -D= Network stability (%) -E= Average response time (micro second)
-F= System Availability (%)

Raft	Node	A	B	C	D	E	F
Nomal Raft	3	198	3	269	99.6 ~ 99.7%	103	99.6 ~ 99.7%
	4	207	3	258		116	
	5	218	4	243		124	
Profosed Raft	3	249	2	263	99.8~ 99.9%	111	99.8~ 99.9%
	4	262	2	251		123	
	5	273	2	242		135	

For details on results in Table 4, Regarding leader election time, the leader election time of the modified algorithm is longer than before modification. This is because the modified algorithm considers more metrics when electing the leader. The frequency of leader replacement was reduced in the modified algorithm. This is because leader replacement becomes less frequent by taking into account the activity of existing leader nodes and selecting a trustworthy leader. TPS showed slightly higher performance than the algorithm before the modification. It appears that the additional metric calculations of the modified algorithm had some effect on performance. The modified algorithm showed better performance in network stability. This is a result of improved network stability by selecting a trustworthy leader. The average response time is slightly longer than that of the modified algorithm. This is due to the delay caused by additional metric calculations during the leader election process. The modified algorithm performed better in system availability. This results in improved availability of the entire system by choosing a trusted leader.

4. Discussion

Various failure situations that can occur in Raft include network partition situations, node failure situations, and leader failure. Factors that affect Raft leader election include leader re-election time, the influence of network conditions, node failure, and cluster size. When a leader failure occurs, it takes time to elect a new leader, and during this process, the cluster may be unable to process transactions for a while. The election process operates on a timeout basis, and the candidate who receives the majority of votes within a certain timeout period is elected as the new leader, so timeout settings can have a significant impact on the performance and reliability of the cluster. Additionally, in a network division situation, if each

divided part independently attempts to elect a leader, the cluster with the fewest nodes may have difficulty electing the leader and may not be able to maintain cluster consistency. Additionally, if a node failure occurs, it may have a significant impact on leader election depending on the size of the cluster, resulting in a situation where the leader must be elected with the minimum node configuration.

In this study, a Raft leader was designed to enhance fault tolerance in a network composed of the minimum number of nodes by considering various failure situations that may occur during consensus in a private blockchain network, such as network partition, node failure, and leader node. During the processing process, an Activity Score variable is placed, and when the platform is configured with the minimum number of nodes in a network split situation or node failure situation and successful leader election is required, the leader node election is conducted according to the modified the Raft algorithm, and the leader node is elected and trusted based on trust. Network failures are minimized to enhance fault tolerance even in a platform environment where the minimum number of nodes is operated.

Through this study, by applying the modified algorithm, the performance in terms of TPS and leader election time deteriorated, but performance improvement of more than 15% was confirmed in terms of leader replacement frequency, network stability, and system availability that can confirm trust between nodes. In the future, we plan to continue research to improve general performance in various environments considering the number of nodes and the number of data per node.

Reference

[1] R. Kaafarani, L. Ismail, and O. Zahwe, “An Adaptive DecisionMaking Approach for Better Selection of Blockchain Platform for Health Insurance Frauds Detection with Smart Contracts:Development and Performance Evaluation,” Procedia Comput Sci,vol. 220, pp. 470–477, Jan. 2023, DOI : 10.1016/J.PROCS.2023.03.060

[2] Data Fabric Market Overview : https://www.marketsandmarkets.com/Market-Reports/data-fabric-market-237314899.html

[3] Pajooh et al., " Hyperledger Fabric Blockchain for Securing the Edge Internet of Things”, Special Issue Recent Advances of Blockchain Technologies in Sensor Networks, pp.358-359, Jan, 2021. DOI : https://doi.org/10.3390/s21020359

[4] Jain et al.," Performance evaluation of hyper-ledger fabric-based consensus mechanism on multi-robot path planning”, Multimedia Tools and Applications: An International Journal. pp.15769-15783,July,2023,DOI: 10.1007/s11042-023-16341-6

[5] Kaushal, Rajesh Kumar | Kumar, Naveen,"Exploring Hyperledger Caliper Benchmarking Tool to Measure the Performance of Blockchain Based Solutions ", 2024 11th International Conference on, pp.1-6, Mar, 2024, DOI :10.1109/ICRITO61523.2024.10522188

[6] Chowdhury, Shovon Das et al., "S-DrivingRecords: Blockchain Based Enhancing Trust and Transparency in Driving Records Using Hyperledger Fabric", 2024 International Conference on,pp.1-6, Mar, 2024, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10499543, DOI:10.1109/iCACCESS61735.2024.10499543

[7] G. Al-Sumaidaee, R. Alkhudary, Z. Zilic, and A. Swidan, “Performance analysis of a private blockchain network built on Hyperledger Fabric for healthcare,” Inf Process Manag, vol. 60, no. 2, pp. 103160-103163, Mar. 2023, DOI: 10.1016/J.IPM.2022.103160.

[8] A. Lohachab, S. Garg, B. H. Kang, and M. B. Amin, “Performance evaluation of Hyperledger Fabric-enabled framework for pervasive peer-to-peer energy trading in smart Cyber–Physical Systems,” Future Generation Computer Systems, vol. 118, pp. 392–416, May 2021, DOI: 10.1016/J.FUTURE.2021.01.023.

[9] Choumas, Kostas,Korakis, Thanasis,"When Machine Learning Meets Raft: How to Elect a Leader over a

Network", 2023 IEEE. pp.3705-3710 Dec, 2023, DOI: 10.1109/GLOBECOM54140.2023.10437805
[10] Bao et al., "Model Checking the Safety of Raft Leader Election Algorithm",2022 IEEE 22nd International Conference on, pp.400-409 Dec, 2022, DOI:10.1109/QRS57517.2022.00048
[11] Battisti, Joao H. F et al., "Performance analysis of the Raft consensus algorithm on Hyperledger Fabric and Ethereum on cloud", 2023 IEEE International Conference on, pp.155-160, Dec, 2023, DOI:10.1109/CloudCom59040.2023.00035
[12] Dautov, Rustem, Husom, Erik Johannes, "Raft Protocol for Fault Tolerance and Self-Recovery in Federated Learning", 2024 IEEE/ACM 19th Symposium on, pp.110-121, Apr, 2024, DOI: https://doi.org/10.1145/3643915.364409
[13] Xu, Jinjie et al., " Raft-PLUS: Improving Raft by Multi-Policy Based Leader Election with Unprejudiced Sorting", SYMMETRY-BASEL, Vol.14, No.6, pp.1122-1124, JUN 2022, DOI:10.3390/sym14061122
[14] Yang, Sijia, Tan, Pengliu, Fu, Haowei, "Improved Raft consensus algorithm based on NSGA-II and K-Means++",2024 10th International Symposium on, pp.383-390, Mar, 2024, DOI:10.1109/ISSSR61934.2024.00055
[15] Zuo, Nan, Chen, Yubin, Zheng, Yuanjie, "Raft Consensus Grouping Mechanism Based on Affinity Propagation Clustering Algorithm",2024 4th International Conference on, pp.15-19 Jan, 2024, DOI:10.1109/ICCECE61317.2024.10504245