VM-DPoSW, the Consensus Algorithm of BFDChain: The Design Principle and Quantitative Analysis

Befund Foundation Ltd.

Abstract— In this paper, we discuss the design detail of VM-DPoSW, the consensus algorithm that supports a robust BFDChain under the DAOS ecosystem. The workflow, design principle, implementation and its controllability analysis are discussed in detail.

I. Introduction

As one of the most important aspects of any blockchain systems, the consensus algorithm design is crucial to construct a robust and health blockchain ecosystem. In BFDChain, we designed a new consensus algorithm named VM-DPoSW, which is a virtual machine based hybrid system with both Proof of Work (PoW) and Delegated Proof of Stake (DPoS) support.

The rest of this paper is organized as follows: In Section II, we present the state of art review of popular consensus algorithms. We then provide detailed technical discussion of VM-DPoSW and its controllability analysis in Section III and IV. Section IV includes concluding remarks of our design.

II. State of art of Consensus Algorithms

A safe, orderly and healthy blockchain requires us to solve two fundamental problems: double spending and byzantine generals problem [8]. Double spending problem means to reuse the currency in two transactions simultaneously. Byzantine generals problem means during the peer to peer communication of the distributed system, some maliciously users may tamper the communication contents, thus lead to security breach or communication inconsistency.

In order to make the whole blockchain safe and consistent, the generation of block needs to reach a certain consensus, thus the consensus algorithm is one of the keys for any blockchain technologies. The common consensus algorithms are PoW, PoS, DPoS, PBFT, and RAFT.

PoW (Proof of Work): The workload proof mechanism, through a large number of HASH operations, calculates a suitable random number and produces a new block. And this is the safest way of security, but at the same time, it is also very energy consuming. Bitcoin [1] is the most typical PoW implementation.

PoS (Proof of Stake): The ownership proof mechanism, through the holding amount and holding time of the token, reduces the difficulty of the block production. This method solves the problem of energy consumption comparing with PoW, but there are certain bottlenecks in security, and system bifurcation is easy to appear. PPCoin [6] is one typical PoW implementation.

DPoS (Delegated Proof of Stake): The agent's equity proof mechanism, by which a certain number of agents are elected by the ballot papers, and the blocks are produced in a certain order between the agents. DPoS greatly reduces the number of verification node and improves transaction confirmation speed under the premise of security protection. However, the corresponding centralization degree is reduced. BitShares is an example of DPoS [2].

PBFT (Practical Byzantine Fault Tolerance): It is a practical Byzantine fault tolerance, and this kind of consensus cannot require the issuance of tokens, which is more suitable for the operation of the alliance chain. In 1999, the PBFT system [7][8] was proposed and the algorithm complexity was reduced to a polynomial level, which greatly improved efficiency. PBFT have 5 steps in its workflow, namely, 1)request, 2)pre-prepare, 3)prepare, 4)commit and 5)reply.

RAFT: To solve the consistency problem in PBFT, Lamport etc. proposed a new algorithm named Paxos, which is the initial prototype of RAFT. It was not until 2013 for RAFT algorithm to be formally proposed by Ongaro etc. in [9]. RAFT achieves the same effect as Paxos and is more convenient in engineering implementation and understanding.

For a specific business scenario, the consensus algorithm has a great impact on the participants' decisions. For the alliance chain with certain trust basis, most of them take PBFT as the first choice, and the PBFT consensus mechanism performs better when nodes are fixed and the number of nodes is less. In the low dependence environment, the robustness of the blockchain system is generally guaranteed by PoW, PoS, and DPoS.

III. BFDT PROXY VIRTUAL MACHINE (BPVM) CONSENSUS ALGORITHM

BFDChain serves the main chain for Befund's decentralized fund service platform for operating activities that are far more complex than that of Bitcoin and Ethereum. Thus, our goal is to design an efficient and robust consensus algorithm to support a sustainable and healthy eco-system.

We use virtual machine based hybrid DPoS and PoS (VM-DPoSW) consensus algorithm to achieve our design goal. Here is the implementation detail of VM-DPoSW:

3.1 Virtual Machine

Virtual Machines (VMs) are the abstract entities that perform mining work on BFDChain. VMs serve two purposes: first, they are the mining worker to solve the hash computing for the proof of work (PoW). Secondly, they are the delegates that represent the share stake of the shareholders of BFDT in the BFDChain eco-system (DPoS). To achieve this, VMs are created by smart contacts to have different computing powers, and the total number of the VMs are upper bounded in a given period of time based on supply and demands. BFDT Shareholders such as side chain owner, decentralized application developer, investors acquire the VMs with different computing power via bidding with BDFT. As VMs are the only eligible miners on the BFDChain ecosystem, and higher computing power represent higher voting right, the BFDT shareholders are incentive to invest on VMs and have BFDT locked in the BFDChain eco-system to achieve stable and healthy long-term growth.

3.2 Why VM-DPoSW?

In the original design of PoW, it is the hope of the designer that all mining workers can use the CPU to perform the mining work such that each node, even with different computing power (thus different hashing power), still has the equal opportunity to participate in the decision-making of the blockchain. However, with the development of the hardware such as GPUs and ASICs, and the aggregation of individual computing power into mining pools, the ordinary miners rarely have the opportunity to create a block. Furthermore, there are more and more criticize of PoW not being environment friendly and slowing down transaction speed on blockchain.

On the other side, the DPoS mechanism such as BitShares tries to tackle the problem of PoW by allowing each node to select the delegates based on its share stake. The top N delegates that have got the most votes have the accounting right. The sufficient decentralization is achieved as long as 50% of the voting shareholders believe their delegates are part of the delegates group that can perform the block creation and validation work [3]. Generally, the blockchain using DPoS is more efficient and power-saving than PoW because all of the blocks creation and validation occurred only on a group of delegates. Yet, there are more and more criticize from the community that pure DPoS only represents the interest of the large shareholders, and the small and medium shareholders rarely have the rights in the block chain decision making process.

The drawbacks of PoW and DPoS motivate us to come up VM-DPoSW, to balance the pros and cons of PoW and DPoS, for a stable, robust, and efficient consensus design.

3.3 How Does VM-DPoSW work?

We will use the Fig.1 to illustrate how VM-DPoSW works.

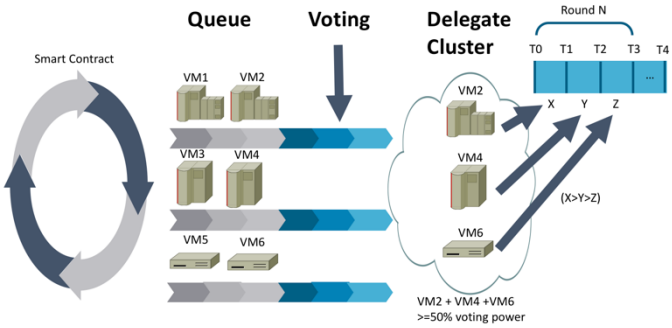

Fig.1 The workflow of VM-DPoSW

a. Create Virtual Machines

First, after a successful biding, the smart contracts on BFDChain are triggered to create virtual machines (VMs) that fall into different categories of computing power. For better illustration, we simplify the model to assume there are only three types of VMs: gold (large computing power), silver (medium computing power), and bronze (small computing power). The computing power of each type is designed such that gold > silver > bronze, i.e.,

$$VM_2^{gold} > VM_4^{silver} > VM_6^{bronze} \quad (1)$$

b. Queue Pool

Let's further assume the newly created virtual machines VM1/2, VM3/4, and VM5/6 belong to gold, silver, and bronze respectively. Right after VMs are created, their role is initially set as witness role, and are put into the queue pool as the delegate candidate.

c. Voting and Delegate Cluster

Sequentially, when the new transaction requests come, new smart contract is triggered to evaluate whether the delegate cluster pool has sufficient delegates to complete the transaction requests. If not, voting process is triggered to select additional witness from the queue pool to the delegate cluster pool. In our scenario, let's assume VM2 of gold type, VM4 of silver type, and VM6 of bronze type are selected as delegates and put into the delegate cluster pool, and they fulfill the requirements that

$$VM_2^{gold} + VM_4^{silver} + VM_6^{bronze} \geq 50\% VotingPower \quad (2)$$

d. Transaction Process

In VM-DPoSW, the transaction requests are processed in different "rounds" in the time spectrum, and in each "round", the hash difficulty is the same for all delegates. In our case, as illustrated in Fig.1., the round N starts at T0, and is expected to end at T3. Our algorithm is designed in the way that the total time in round N, $\Delta T = T3 - T0$, is equally divided into K slides, where K is the total number of delegates in the delegates cluster, i.e.,

$$T1 - T0 = T2 - T1 = T3 - T2 = \Delta T / 3 \quad (3)$$

Let's assume VM2 in gold type starts to serve the transaction request at T0 and stopped at T1 (the order may be different, and we will address the ordering in section 3.4). Within $\Delta T/3$ time frame, VM2 processed X number of transaction requests. Same scenario applies to VM4 and VM6 at T1 and T2, and each processed Y and Z number of transaction requests within $\Delta T/3$ time frame. Recall in terms of computing power, we have (1), and the hash difficulty is the same for all delegates in round N, thus we will have

$$X > Y > Z \quad (4)$$

We can see from eq.(3) that VM-DPoSW gives each delegate an equal opportunity to participate the mining process regardless the computing power of the delegates. On the other hand, eq. (4) shows that the delegate with higher computing power will process more transaction requests (thus more blocks) and thus generate more rewards for the shareholder with higher share stake, even it was only given same process time comparing with other delegates with lower computing power. In reality, we will put more constraints to ensure a sophisticated delegates system. For example, we may set the upper bound for the percentage of VMs in each category.

3.4 The signature and ordering of VM-DPoSW

As pointed out in [4], in PoW, the expected time to calculate a correct "nonce" is proportional the hash difficulty. i.e., the nonce H_n must satisfy the relations:

$$n \leq \frac{2^{256}}{H_d} \wedge m = H_m \quad (5)$$

With $(n,m) = PoW(H_{\backslash n}, H_n, d)$.

Where n is the mix-hash and m is the pseudo-random number cryptographically depend on H and d. $H_{\backslash n}$ is the new block's header H without the nonce and mix-hash components, and d is the current data set. PoW is the proof of work function. Eq.5 is the foundation of the security of the blockchain and is the fundamental reason why a malicious node cannot

propagate newly create blocks that would otherwise overwrite history.

In VM-DPoSW, however, we may choose to set the hash difficulty lower so that even the VM with lowest computing power can finish the hash computing quickly and can generate new block and process transactions in its given time window. While this design significantly increases the efficiency of the eco-system, it may increase the security vulnerability as malicious users may take advantage of the lower hash difficulty. This requires us to add additional security mechanism, namely, signature and random ordering, to safeguard the BFDChain ecosystem.

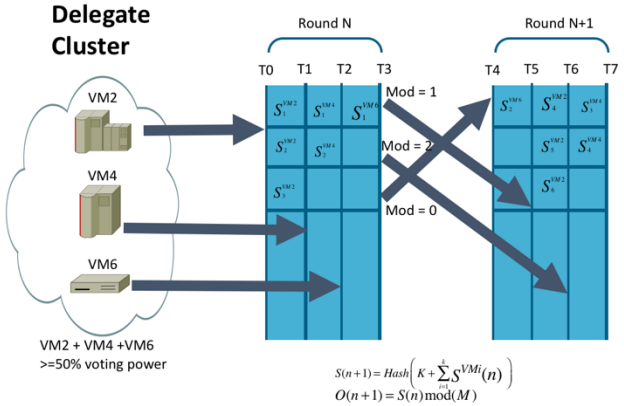

Fig.2 The signature and ordering of VM-DPoS

The design goal of the signature and random ordering is to ensure a given delegate VM in the delegate cluster can only process transaction request in the assigned "round" as well as the assigned time window. As illustrated in Fig.2., assume in round *N*, VM2 starts to perform the mining and create a new block at T0, we add a private key into the optional filed of the block and got a signature S_1^{VM2} by performing

$$S_1^{VM2} = Hash(K_1^{VM2}) \quad (6)$$

Where K_1^{VM2} is the private key value for the 1st block created by VM2 in the round *N*. Similarly, when VM2 creates the 2nd and 3rd block, the signature S_2^{VM3} and S_3^{VM3} are calculated by

$$S_2^{VM3} = Hash(K_2^{VM2} + S_1^{VM3}) \quad (7)$$

$$S_3^{VM3} = Hash(K_3^{VM2} + S_2^{VM3}) \quad (8)$$

respectively. Once VM2 creates its 3rd block, VM2 determines this is the last block it can process, it then broadcast the signature S_3^{VM3} to all other VM delegates, before T2.

Sequentially, VM4 and VM6 will be the second and third VMs to follow similar procedure to create their blocks and perform the signature broadcast at the end of their last block mining. In our case, S_1^{VM6} between T2 and T3, is the last signature in round *N*, and is the proof needed by each VM delegate to participate mining in the next round *N+1*. Image a malicious delegate tries to cheat the system by performing mining before round *N* finish and try to work on round *N+1*. Its mining of new block(s) will be rejected because it will not have the final signature S_1^{VM6} to sign the newly created block.

In round N+1, we can generalize (6)-(8) as below:

$$S(n+1) = Hash\left(K(n) + \sum_{i=1}^{k} S^{VMi}(n) \right) \quad (9)$$

Where *K(n)* is the key value at round N+1, and $\sum_{i=1}^{k} S^{VMi}(n)$ is the sum of the hash value of K number of VM delegate in the previous round (for example, we have K=3 in round N).

In addition, we use the following mechanism to determine the order of VM delegate in round N+1:

$$O(n+1) = S(n) \bmod (M) \quad (10)$$

Where *S(n)* is the signature of the last block in round *N* (i.e., S_1^{VM6} in our example) and M is the number of VM delegates. The mod operation will determine the serving order of each VM delegate in round *N+1*. In our case, in *N+1*, the mod result for VM2, VM4, and VM6 are 1, 0, and 2. Thus VM6 is scheduled to start to create block first at time stamp T4, followed by VM2 (3 blocks starting at T5), and VM4 (2 blocks starting at T6).

If there is conflict during the mod operation, we point the VM delegate to the next available slot. In case a particular VM delegate is not able to generate block in its given time windows, we will use the signature in the previous broadcast.

IV. The Controllability Analysis of VM-DPoSW

In any system design, the controllability is the most import aspect to inspect. The consensus algorithm design is not an exception.

By "controllable", we mean to evaluate whether

VM-DPoSW consensus algorithm can be properly managed even with heavy transaction requests from the main chain and side chain, and whether our algorithm can steer the resource efficiency over BFDChain eco-system from any initial value to the optimum state within a limited time window. This kind of controllability property is a crucial to achieve queue stabilization, delay bounds, and optimal resource control.

Assume

$$f(K,S(n)) = Hash\left(K + \sum_{i=1}^{k} S^{VMi}(n) \right) \quad (11)$$

$$g(S(n),M) = S(n) \bmod (M) \quad (12)$$

Eq.(9) and (10) yields

$$S(n+1) = f(K,S(n)) \quad (13)$$

$$O(n+1) = g(S(n),M) \quad (14)$$

Eq. (13) and (14) describe a non-linear discrete system, where the state vector $x(n) = [s(n), o(n)]^T$ represent the array of signature hash value and the ordering of the VM delegate. The input vector $u(n) = [K, M]^T$ represent the array of the key value and the number of VM delegates. The linearization is necessary to analyze the controllability [5]. Assume the equilibrium point is $(s(n)_0, o(n)_0, K_0, M_0)$, all of which are positive real numbers; linearizing Eqs. (13), (14) the equilibrium point, we obtain the following linearized system in state space:

$$\begin{pmatrix} \delta\dot{S}(n+1) \\ \delta\dot{O}(n+1) \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial S} & \frac{\partial f}{\partial O} \\ \frac{\partial g}{\partial R} & \frac{\partial g}{\partial O} \end{pmatrix} \begin{pmatrix} \delta S(n) \\ \delta O(n) \end{pmatrix} + \begin{pmatrix} \frac{\partial f}{\partial K} & \frac{\partial f}{\partial M} \\ \frac{\partial g}{\partial K} & \frac{\partial g}{\partial M} \end{pmatrix} \begin{pmatrix} \delta K \\ \delta M \end{pmatrix} \quad (15)$$

Let $A = \begin{pmatrix} \frac{\partial f}{\partial S} & \frac{\partial f}{\partial O} \\ \frac{\partial g}{\partial R} & \frac{\partial g}{\partial O} \end{pmatrix}, B = \begin{pmatrix} \frac{\partial f}{\partial K} & \frac{\partial f}{\partial M} \\ \frac{\partial g}{\partial K} & \frac{\partial g}{\partial M} \end{pmatrix}$, we have

$$\delta\dot{x}(n+1) = A\delta x(n) + B\delta u(n) \quad (16)$$

By modern control theory [5], the system is controllable iff $U = [B \; AB]$ is full row rank. As $(s(n)_0, o(n)_0, K_0, M_0)$ are all positive real numbers, we can conclude the $U = [B \; AB]$ is full row rank, and thus the VM-DPoSW is controllable.

V. Conclusions

In this paper, we have discussed the design detail of VM-DPoSW, the consensus algorithm that support a robust BFDChain under the DAOS ecosystem. We discussed our motivation, the work flow and the additional security mechanism such as signature and random ordering. At last, we use the modern control theory to prove the VM-DPoSW is controllable under the state space analysis.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", 2009.

[2] "https://bitshares.org/,".

[3] "https://bitshares.org/technology/delegated-proof-of-stake-consensus/,".

[4] Ethereum: A secure decentralized generaliaed transaction ledger EIP-150 revision", Gavin Wood

[5] Z. Bubnicki, Modern control theory, Spring Berlin Heidelberg New York 2005.

[6] S. King and S. Nadal, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake,", 2012.

[7] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in Symposium on Operating Systems Design and Implementation, 1999, pp. 173--186.

[8] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem," Acm Transactions on Programming Languages & Systems, vol. 4, pp. 382-401, 1982.

[9] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," Draft of October, 2013.