Actor-based Model for Concurrent Byzantine Fault-tolerant Algorithm

Chi Zhang[1, a], Rong Wang[1, b], Wei-Tek Tsai[1,2, c], Juan He[1, d], Can Liu[1, e], Qi Li[1, f]

[1]School of Beihang University, Beijing 100191, China.

[2]Beijing Tiande Technologies, Beijing 100080, China.

[a]zy1606603@buaa.edu.cn, [b]wangrong@buaa.edu.cn, [c]tsai7@yahoo.com, [d]xiongbao_hj@buaa.edu.cn, [e]liucan@buaa.edu.cn, [f]liqi7@buaa.edu.cn

Abstract. Consensus mechanism is the core element of blockchain (BC) technology. A good consensus mechanism can improve the performance of BC system and promote the application of BC technology. The most widely used consensus algorithms are Proof of Work (PoW) algorithm and the Proof of Stake (PoS) algorithm. However, only 3 to 7 transactions can be processed per second, which means the BC system does not meet the performance requirements of commercial application. The Practical Byzantine Fault Tolerance (PBFT) algorithm provides (n-1)/3 fault-tolerance under the premise of ensuring liveness and safety, which can process hundreds of transactions per second. However, large commercial BC applications often require higher performance and the throughput of BC system need to reach thousands of transactions per second (TPS). To further improve system performance, we propose actor-based model for Concurrent Byzantine Fault Tolerant (CBFT) algorithm, which uses actor model to achieve higher concurrency and improve the transaction processing speed of BC system. We design, development and test a BC system, which is based on CBFT algorithm. The experimental results show that the algorithm can maximize the bandwidth of the network at the optimal block interval time of 3 seconds, the TPS can reach 1500-2000, and the delay can be controlled between 100ms and 1000ms, which can meet the performance requirements of most commercial systems.

Keywords: Blockchain, consensus mechanism, actor model, concurrent byzantine fault-tolerant algorithm.

1. Introduction

The concept of blockchain was first proposed by Satoshi Nakamoto in the bitcoin forum in late 2008. The paper "Bitcoin: A Peer-to- Peer Electronic Cash System" known as bitcoin whitepaper elaborated an electronic cash system architecture combining with P2P network technology, encryption technology, the time stamp technology, blockchain technology [1]. BC system is a distributed database system, and it can also be understood as a distributed ledger technology (DLT) maintained by multiple nodes, which is not easy to tamper with, difficult to counterfeit, and can be traced back. In the blockchain system, each participant is a node. All nodes have a complete set of ledgers, where all historical account information is recorded. Any node that needs to initiate a transaction can send the transaction to each node in the blockchain network. Each node will verify and execute the transaction, and ensure that the ledger on this node can be accurately updated. The blockchain records all the information contained in the transaction. Once the data entered the blockchain, internal staff could not make any changes to the blockchain. This unchangeable feature does not come from the use of some kind of operation, but because of the blockchain system and the mechanism itself, which makes blockchain technology easier and more efficient than other security technologies.

The blockchain provides a trust mechanism for completing the transaction without an intermediary in an untrusted environment, and greatly reduces the cost of trust. It will become the cornerstone of building a future value Internet [2]. Blockchain technology will be widely used in the future, such as digital copyright protection, supply chain logistics, anti-counterfeiting traceability, digital lottery and other very common applications, and will become the infrastructure of digital society [3].

The bottleneck of the blockchain is the performance of processing transaction. The shortcomings of POW series algorithm are the low throughput, high latency, and inefficiency [4]. Bitcoin which is the representative of the public blockchain. The transaction processing frequency of Bitcoin is about 6-7 TPS. Bitcoin requires six new blocks to confirm a transaction. On average, it takes ten minutes to generate a new block. The entire network to confirm a transaction takes 1 hour. These shortcomings make blockchain difficult to apply to most business scenarios.

The consensus algorithms of public blockchain like POW and POS cannot avoid problems such as concentration of rights, long block confirmation period, and low transaction frequency etc. The anonymous public blockchain system enables any node on the whole network to access data and the attack to blockchain cannot be traceable. In many application scenarios, user privacy cannot be effectively protected, which is fatal in commercial applications [5].

The PBFT algorithm was proposed by Miguel Castro and Barbara Liskov of MIT in 1999 [6]. Their intention was to design a low-latency storage system, reducing the complexity of the BFT algorithm from exponential to polynomial and making the BFT algorithm feasible in practical system applications. However, the computational efficiency depends on the number of nodes participating in the protocol, and is not applicable to the blockchain system with an excessive number of nodes [7]. The blockchain system with PBFT algorithm usually has a transaction processing performance of hundreds of TPS, but this is far from meeting the needs of commercial applications. The scalability is poor and it is difficult to apply it commercially.

The main goal of this thesis is to design and implement blockchains with high performance, high availability and high scalability. We optimize and improve the PBFT algorithm, and propose a CBFT algorithm based on the Actor model. Further, we implemented CBFT using the Akka toolset and did a series of experiments. The content of this paper is organized as follows: Section 2 introduces the relevant technical background; Section 3 proposes the CBFT algorithm based on the Actor model; Section 4 conducts experiments and analysis; Section 5 summarizes the work of this paper.

2. Background

2.1 CBFT.

CBFT algorithm is a new consensus algorithm developed from the PBFT. CBFT algorithm is mainly to make each node of the BC reach a consensus on the creation, validation and storage of blocks, to ensure the consistency of the replicas of each node in BC system. In a period of time, the transaction requests initiated by the client will be stored in the transaction buffer pool of each node. It consists of four phases: block determination, pre-prepare, prepare and commit. The last three stages are similar to the three stages of PBFT algorithm. As shown in Fig. 1. An important advantage of CBFT is concurrency that each block can be voted and built in parallel with other blocks, thus greatly improving the speed of consensus [8]. Another important feature of CBFT is that it can detect damaged nodes during the commit phase and broadcast messages in the final phase to identify traitor nodes. The steps include confirmation of the transaction level and voting, building block, block verification, block confirmation. As shown in Fig. 1.

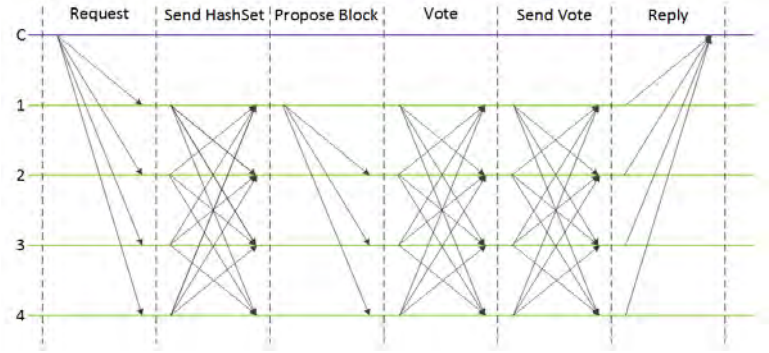

Fig. 1 CBFT consensus algorithm

(1) Each node maps all transactions in the local transaction pool to the HashSet through the hash algorithm, and sends the HashSet to other nodes. Each node has the HashSets of all nodes.

(2) Each node takes the common transactions of all nodes from the local transaction pool according to the received transaction map set HashSets and the master node needs to verify the obtained common transaction, build the block and send it to other nodes.

(3) After each node receives the new block created by the master node, each node completes the verification by comparing the common transaction obtained from the local transaction pool with the transaction in the received block. After the verification, the verification results and digital signatures are sent to other nodes.

(4) Each node receives the voting result and signature of all nodes to the new block and each node re-signs and forwards the collected voting result set so that each node receives the votes of all nodes and counts the voting result set to get the final result, so as to decide whether to accept the block or not.

2.2 Actor Model.

Actor model is a concurrent programming model, proposed by Hewitt et al. In 1973 [9]. It uses actors as a basic element of concurrent programming. Actors can make local decisions based on received messages, create more actors, send more messages and decide how to respond to the next message. The actor model has now become the theoretical basis for many computational theories and concurrent systems. The actor model has many features, such as no shared state, simple high-level abstraction and asynchronous non-blocking event-driven programming model which makes it ideal for modeling concurrent programs [10]. In addition, the most implementation of actor model is very lightweight so that users can quickly create and destroy a lot of actors. Hundreds of thousands of actors running in parallel at the same time is very common, but it only takes up very little memory.

Benefits of using the actor model:

(1) Event model driver: Communication between actors is asynchronous, and actor can handle other things without blocking or waiting after sending the message.

(2) Strong isolation: The methods in the actor cannot be directly called from the outside and everything is done through messaging, thus avoiding data sharing between actors. Actor who wants to observe a change in the state of another actor can only ask through messaging.

(3) Position transparency: The code is the same whether the actor is local or remote.

(4) Lightweight: Actor is a very lightweight computing stand-alone. A single actor only accounts for more than 400 bytes and even a small amount of memory can achieve high concurrency.

(5) Resilience: Batch creation and destruction of lightweight actors can achieve batch migration of processing elements, thus dynamically adjusting the distribution of processing elements in the cluster.

3. Actor-based Model for Concurrent Byzantine Fault-tolerant Algorithm

3.1 Model.

The actors are decoupled from each other and communicate asynchronously through messaging. The BC system based on the actor model can achieve high concurrency and improve the overall processing performance of the system. Just like in every stage of the factory pipeline where workers are responsible for a processing flow, in the four stages of CBFT algorithm, actor are responsible for storaging state and transmitting messages between nodes in CBFT algorithm. Actors are independent of each other, but they cooperate with each other, asynchronously process the messages and jointly complete the whole process of the algorithm. The schematic diagram of the flow of pipeline is shown in Fig. 2.

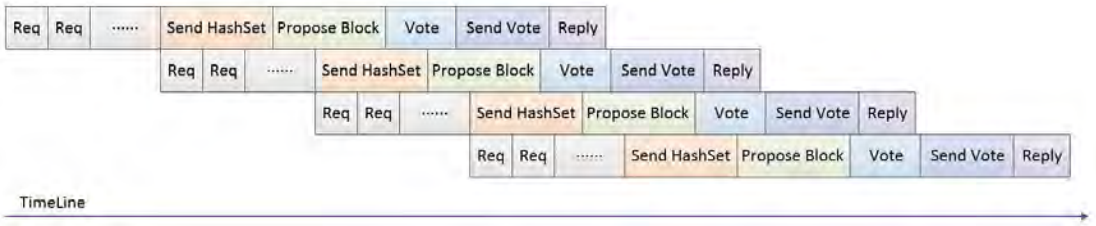

Fig. 2 CBFT parallel pipelining diagram

In the design and implementation of Actor-based model, we abstract each consensus node into a Node ActorSystem, and each ActorSystem contains many different types of Actors to deal with different messages. Each ActorSystem is composed of these actors, such as RequestActor, HashSetActor, BuildBlockActor, VerifyBlockActor, VoteActor, VoteSetActor, BlockChainActor. The entire process from the client sending the request to the request being processed by the ActorSystem is shown in Fig. 3.The circle represents the Actor,and the arrow line represents the type of message sent, and the symbol # indicates that the message is a broadcast message which needs to be broadcast to other Actors of the same type in the ActorSystem.

Each component function :

(1)The RequestActor is responsible for receiving the client 's request and broadcasting it to other nodes.

(2) The HashSetActor is responsible for collecting transaction HashSet and computes the common request intersection of four nodes.

(3)The BuildBlockActor is responsible for taking the transaction from the redis buffer for building blocks.

(4) The VerifyBlockActor is responsible for verifying the Block , including whether the transaction has been tampered with and whether the transaction is in the transaction buffer pool.

(5) VoteActor is responsible for collecting voting results of all nodes for new blocks, and broadcasting the voting result set to the VoteSetActor of other nodes.

(6)The VoteSetActor is responsible for counting the voting results of all nodes for the new batch number batchum.

(7)The BlockChainActor is responsible for the final voting statistics.

3.2 Algorithm.

In a distributed consensus network, in order to reach a consensus, each representative must start from the same state, this means that the original blocks of each representative node must be consistent. In order to achieve this step, each node must independently listen to the transaction data in the network and write it to the memory in real time. The master node initiates a consensus process, and the transaction data and block information of all honest consensus accounting nodes in the whole network are consistent. The so-called consistency means that in the configuration of the consensus, the configuration number, the block height , the previous block hash index, and the version number are all the same. The algorithm flow is as follows in Figure 3:

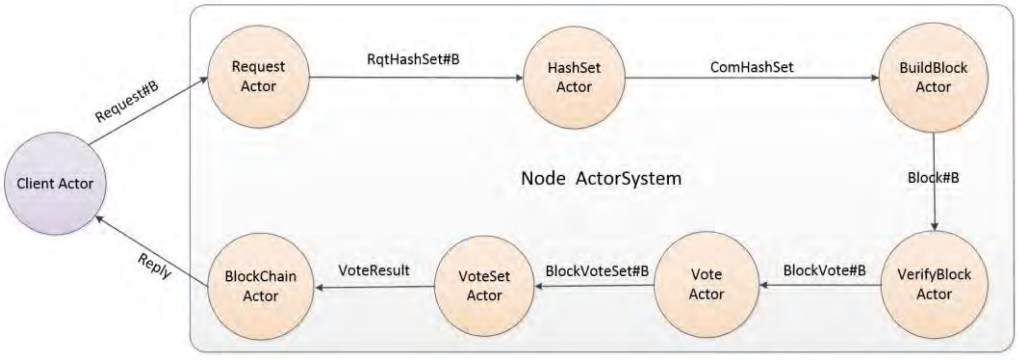

Fig. 3 Algorithm Flow

(1) When the initiator of the transaction in the system initiates a transaction, the transaction is signed by the private key and sent to the BC network through the client. The RequestActor collects the

client request, assigns a batchnum to the batch of requests and broadcasts a ReqHashSet corresponding to the requests. HashSetActor collects HashSet and for ReqHashSet with the same batchnum, the common request intersection of four nodes is calculated and sent to BuildBlockActor with ComHashSet;

(2)The BuildBlockActor extracts the transaction from the redis buffer according to the common request intersection, calculates the merkle_tree and merkle_root , and builds the block. The master node broadcasts the new block with batchum to the VerifyBlockActor of all ActorSystems ;

(3)VerifyBlockActor verifies the Block , including whether the transaction has been tampered with, whether the transaction is in the transaction buffer pool, if all the transactions in the block are verified, the node voted in favor, otherwise it voted against, and the voting result is signed with the private key of the node. Then broadcast to other nodes' VoteActors ;

(4)The VoteActor collects the voting results of all the nodes for the new block, and then collects the voting result set with the private key of the node and broadcasts it to the VoteSetActor of other nodes. The VoteSetActor counts the voting results of all the nodes for the batch sequence batchnum of new block, and determines the ActorSystem whether to submit the new block to Blockchain. If the final voting statistics are true, the BlockChainActor submits the new block to the Blockchain in batchnum order. If false, it discards the new block directly and sends back the transaction in the block to RequestActor for the next round of consensus process.

4. Experimental and Analysis

The Akka toolset is used to write fault-tolerant and highly scalable Actor-based model applications [11] . We use Akka toolset to implement CBFT algorithm based on actor mode. We design and implement the algorithm on three principles: high performance, scalability and usability. In order to test and get experimental results, we introduce a transaction simulation module in the BC system and create 1,000 new simulation trading accounts. Through 1000 new simulated transaction accounts, each transaction account has an initial quota of 100,000, and the volume of each transaction is 1.

4.1 Lab Environment.

Due to the limitations of the experimental conditions, the experiment uses four computers configured as: Intel Core i7-7700HQ processor, 8G memory. To facilitate experimentation and eliminate other interference factors, the four computers use the same software configuration. The configuration is as follows:

Operating System: Ubuntu 16.04

Database: MySQL 5.1.73.

Transaction cache: Redis 3.2.

Java version: Java 1.8

In order to eliminate other interference factors on the experiment, four nodes are connected to the same switch and a simple dedicated BC network is constructed using the same IP subnet. We record the time spent on broadcasting in the communication phase, and the speed of transaction processing.

4.2 Throughput.

Throughput is used to measure the system's ability to handle transactions and requests per unit time. It is an important indicator of the system's concurrency ability. In this article, we use TPS to measure throughput. In BC system, transaction throughput refers to the total number of transactions confirmed and written into the block chain. We took eight different block time intervals of 1s, 3s, 10s, 20s, 30s, 60s, 120s and 240s respectively, as shown in Figure 4. Each time interval was tested 30 times, and the average of 30 times was taken as TPS of each block time interval.

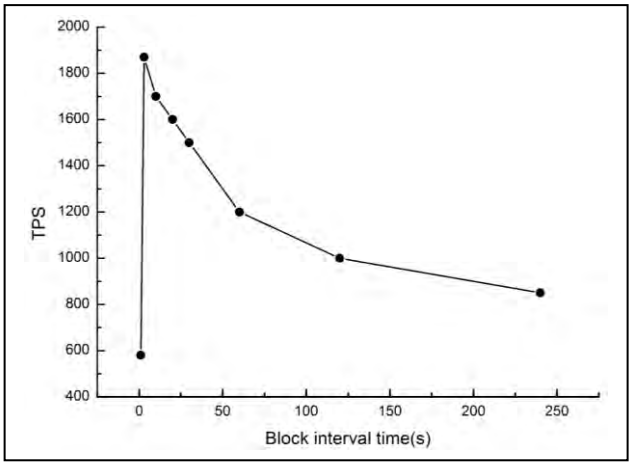

Fig. 4 Throughput in different block interval time

As we can see from Fig. 4, TPS of BC can reach 1500-2000 in a certain block time interval. Different block time intervals will affect TPS. The longer the block time interval, the more transactions a node receives during that time. The more transactions in the block, the larger the block, and the longer the time for network propagation and verification. TPS first increases and then decreases with the block time interval, and 3s is the best block time interval for BC.

4.3 Delay.

Transaction latency and throughput are closely related in BC system. The delay time of a transaction refers to the time interval during which a transaction is confirmed. The latency indicator measures the network communication performance of the entire system and the running time of the consensus algorithm. The low latency means that users wait less time and the experience is better. We take eight different block time intervals of 1s, 3s, 10s, 20s, 30s, 60s, 120s and 240s respectively, and count the time required for the ten blocks, then we calculate the average of all transaction delays in the block. Transaction processing delays at different block time interval is shown in Fig. 5.

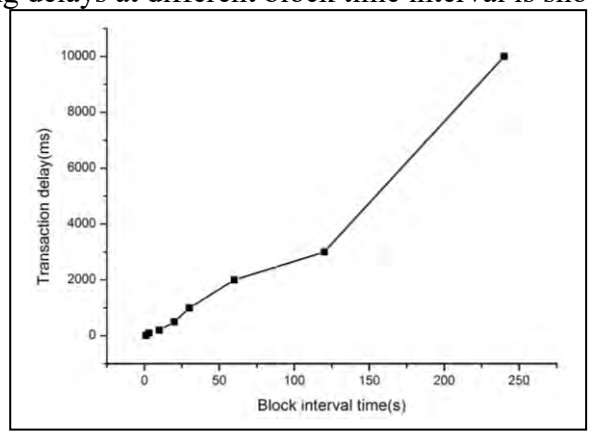

Fig. 5 Delay in different block interval time

As shown in Fig. 5, different block time intervals affect latency. The longer the block time interval, the more transactions a node receives during that time, the more transactions the block contains, and the larger the block, the longer the verification time. Due to the limited network bandwidth, the larger the block confirmed by the broadcast, the lager the propagation delay. With the increase of the block time, the more transactions can be processed per unit time, the higher the TPS; but if the transactions in the block exceed the processing power of one node, the transcations accumulate, the thread is blocked, then Reduce TPS.

5. Summary

In this paper, we propose actor-based model for CBFT algorithm, which uses actor model to achieve higher concurrency and improve the overall transaction processing speed of BC. It uses the

de-coupling between actors, asynchronous communication through messageing, which greatly improves the concurrency of the system and improves the overall processing performance of the system. We use the Akka toolset to implement the CBFT algorithm based on the actor model and conduct experimental tests. After experimental testing, the algorithm has the advantages of high throughput, low latency, and high concurrency, which can meet the performance requirements of most commercial systems.

Future work includes two aspects:

(1) Improve node fault tolerance of BC system

Consider communication problems that may occur between nodes, such as delays and loss of messages. The node down detection function is added, and the master node needs to perform view change, and the slave node does not need to start the view replacement. Consider the case where the traitor node forges a message, deliberately delays forwarding or not forwarding the message, or sends a different message to different nodes.

(2) Consider state synchronization after rejoining of a recovery node

If a node crashes due to an unexpected situation, it often needs to restart and rejoin the BC network, but other nodes may have been running for a long time, and the newly joined nodes need to resynchronize the state of other nodes. It is impossible for other nodes to wait for the new node to complete synchronization before starting to process the transaction. First restore the latest state through a stable checkpoint in a certain period of time, synchronize the block after the stable point, and then let the new node directly participate in the consensus, and finally gradually synchronize the block before the checkpoint. In this way, other nodes will not enter the stop state because of the long synchronization time of the new node. This also ensures the consistency and continuity of the state of each node in the BC system.

References

[1]. Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." (2008).

[2]. Mougayar, William. The business blockchain: promise, practice, and application of the next Internet technology. John Wiley & Sons, 2016.

[3]. Swan, Melanie. Blockchain: Blueprint for a new economy. “O’Reilly Media, Inc.", 2015.

[4]. Tschorsch, Florian, and Björn Scheuermann. "Bitcoin and beyond: A technical survey on decentralized digital currencies." IEEE Communications Surveys & Tutorials 18.3 (2016): 2084-2123.

[5]. Conti, Mauro, et al. "A survey on security and privacy issues of bitcoin." IEEE Communications Surveys & Tutorials (2018).

[6]. Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance." OSDI. Vol. 99. 1999.

[7]. Vukolić, Marko. "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication." International Workshop on Open Problems in Network Security. Springer, Cham, 2015.

[8]. Tsai, Wei-Tek, and Lian Yu. "Lessons Learned from Developing Permissioned Blockchains." 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2018.

[9]. Hewitt, Carl. "Actor model of computation: scalable robust information systems." arXiv preprint arXiv: 1008.1459 (2010).

[10]. Vernon, Vaughn. Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka. Addison-Wesley Professional, 2015.

[11]. Roestenburg, Raymond, Rob Bakker, and Rob Williams. Akka in action. Manning Publications Co., 2015.