Performance Evaluation of Hyperledger Fabric with Malicious Behavior

Shuo Wang(✉)

Tsinghua University, Beijing, China
wangs16@mails.tsinghua.edu.cn

Abstract. Hyperledger Fabric is a widely-used permissioned blockchain platform for enterprise consortium applications. It adopts Practical Byzantine Fault Tolerance (PBFT) algorithm as the consensus protocol in its version 0.6. Faulty replicas could intentionally delay messages, be not responsive and send inconsistent messages to different replicas. Faulty clients and replicas could also launch denial-of-service attack to make resources unavailable. The malicious behavior significantly undermines the system. However, the existing performance evaluation for Fabric is accomplished in a fault-free environment without malicious behaviors. In this paper, we analyze the impact of malicious behavior, design malicious behavior patterns and test the blockchain performance on Hyperledger Fabric.

Keywords: Hyperledger Fabric · PBFT · Consensus · Permissioned blockchain

1 Introduction

Blockchain is an open, distributed ledger that can record transactions between parties in a trustable and immutable way. The key contribution of blockchain is achieving consensus in a decentralized environment among replicas from different parties who cannot fully trust each other. With this advanced property, blockchain has been widely used as the underlying data structure and consensus mechanism of bitcoin [1], alternative cryptocurrencies (altcoins) and decentralized application platforms such as Ethereum [2] and Hyperledger [3].

Hyperledger Fabric is an enterprise-grade permissioned blockchain platform for distributed applications among company consortiums. As its latest version has not adopted a fault-tolerant consensus protocol, we now focus on its version 0.6 [4], which adopts PBFT [5] as the consensus protocol. There are some works on performance evaluation benchmark for Hyperledger Fabric. However, they evaluate in a non-faulty environment. For permissionless blockchains such as bitcoin and Ethereum, malicious behaviors and attacks happen frequently because their crytocurrencies and applications has great value in the real world. Attacks will also be targeted when more and more business applications are run in permissioned blockchain platforms.

J. Joshi et al. (Eds.): ICBC 2019, LNCS 11521, pp. 211–219, 2019.
https://doi.org/10.1007/978-3-030-23404-1_15

The attacks could come from malicious clients, which send faulty transactions and try to undermine the blockchain system. Peers could also conduct internal malicious behavior during the consensus process to diverge or break down the blockchain system. The complete analysis and experiments of the system performance with malicious behaviors are needed. In summary, we mainly have three contributions:

- We thoroughly analyze the impact of malicious behavior on the PBFT consensus mechanism.
- We design the malicious behavior patterns of the faulty replica and clients and test the performance on Hyperledger Fabric.
- Based on the analysis and experiments, we provide some sights on how to improve the system performance under attacks.

2 Related Work

2.1 Practical Byzantine Fault Tolerance

PBFT is a classic form of BFT state machine replication algorithm. The algorithm is designed to tolerate no more than f faulty replicas in an asynchronous network where $N = 3f + 1$ is the total number of replicas. It guarantees its safety by a three-phase consensus protocol: (1) **pre-prepare phase:** the primary receives request messages from clients and broadcasts pre-prepare messages. (2) **prepare phase:** when a backup replica receives and accepts the pre-prepare message, it then broadcasts the prepare message to other backups and also receives prepare messages from them. (3) **commit phase:** Once a replica collects $2f + 1$ prepare messages corresponding to its pre-prepare messages, it marks as prepared and multicasts commit messages. When it receives $2f + 1$ commit messages, it can conclude that all non-faulty replicas agree on the order of the requests across views. During each phase, if a replica does not receive enough messages to proceed to the next step before the timeout, it multicasts a view-change message. When the potential primary in the new view received $2f + 1$ view-change messages, it will multicast a new-view message to start a new view.

2.2 Hyperledger Fabric Architecture and Performance Evaluation

Hyperledger Fabric could run chaincode and support many programming languages. Fabric version 0.6 follows the traditional state-machine replication design and adopts the classic and native Practical Byzantine Fault Tolerance as consensus. Fabric then introduces a novel execute-order-validate paradigm in its version 1 [6]. The architecture and workflow of Fabric have some fundamental changes and Fabric version 1 introduces some new roles, such as endorsers, orderers and committers. Endorsers simulate transaction execution and provide writeset and readset. Orderers uses consensus to establish a total order of transactions and deliver blocks in sequence to committers, which update the world state.

There are some previous work on Hyperledger Fabric blockchain performance evaluation and benchmark frameworks. Blockbench [7] design different types of workloads to measure and understand performance at different layers on Fabric v0.6. Hyperledger Project also has an official benchmark tool Caliper [8], which provides metrics such as latency, throughput, and resource consumption on multiple Hyperledger blockchain frameworks including Fabric version 1, Sawtooth, Iroha and Burrow. Harish Sukhwani uses a Stochastic Petri Nets modeling formalism to model the workflow of Fabric and find critical steps which could be potential bottleneck for performance [9]. However, they did not consider the malicious behaviors and the consequences on blockchain performance. There are some other blockchain platforms with performance evaluation with malicious behavior. For example, in the Algorand blockchain [10], a particular attack strategy is proposed that the block proposer with the highest priority sends one kind of block to half of the replicas and another different block to the rest of replicas and evaluate the performance. Blockchains are designed to tolerate faults and there will be attacks targeted to them when they are practically used and involve real-world business.

3 Analysis of PBFT with Malicious Behaviors

Faulty replicas may intentionally delay messages, be not responsive, or multicast inconsistent messages to different groups of replicas. These behaviors undermine the system, especially when the faulty replica function as the primary, though view-change mechanism can restrict the malicious behaviors to some extent. Here, we analyze how the faulty replicas could undermine the network at their best efforts during each phase of PBFT consensus process and how their malicious behaviors could affect the performance.

When the primary is a non-faulty replica, since it only sends a unique type of request messages m, all the pre-prepare, prepare, and commit messages will correspond to this request. In this case, if faulty replicas send messages inconsistent with m, non-faulty replicas will simply ignore them. The faulty replicas can also delay messages or simply not reply them. The first pattern of malicious behavior is not to cooperate with the non-faulty primary and backups during the consensus process.

When the primary is a faulty replica, it can create different messages in each phase and send them arbitrarily. Each replica will receive particular messages specially designated by the primary. Other faulty replicas also know how these different messages are distributed and collude with the primary to send different messages to different groups of replicas.

As the faulty replicas want to keep the faulty primary's position to have a long-term impact on the system efficiency, the faulty replicas have to guarantee that no more than f replicas will send view-change message. Therefore, in the pre-prepare phase, the primary still has to send consistent pre-prepare messages to at least $f + 1$ non-faulty replicas. Likewise, at least $f + 1$ non-faulty replicas should be prepared and committed. In summary, in all phases, faulty replicas

can decide which non-faulty replicas receive consistent messages (the number of those replicas is denoted as k).

Based on the analysis above, k cannot be less than $f + 1$ in any phase. In other word, $f + 1 \leq k \leq 2f + 1$. We denote the set of non-faulty replicas who receive enough consistent messages as set $\mathcal{N}_{pp}$, $\mathcal{N}_p$, $\mathcal{N}_c$ in pre-prepare, prepare, and commit phase respectively.

$$\mathcal{N}_c \subseteq \mathcal{N}_p \subseteq \mathcal{N}_{pp} \subset \mathcal{N} \tag{1}$$

The second pattern of malicious behavior is to keep some non-faulty replicas out of the normal consensus process.

When the primary is faulty, faulty replicas can delay the messages and make non-faulty replicas receive messages just before the timeout and it is called `delay attack`. Non-faulty backups rely on pre-prepare messages to trigger consensus process and have to wait for faulty replicas' messages to achieve a quorum. The third pattern of malicious behavior is to delay pre-prepare messages to trigger the consensus process late or delay prepare and commit messages to achieve a quorum late.

In the Fabric v0.6 implementation, the primary works in batch mode which means the primary creates a block with requests of batch size (500 by default) and multicasts pre-prepare message with the batch. Here, delaying pre-prepare messages means creating batches and sending pre-prepare messages at a low rate.

4 Denial-of-Service Attack

4.1 High-Rate Spam Transactions

In Fabric, malicious clients could flood the network with high-rate spam transactions by keep sending transactions to all peers. As malicious users may control many clients, they could also launch a distributed denial-of-service attack to exceed the capacity of blockchain.

When clients keep sending extremely high-rate transactions, message channels of peers are full of transaction messages and peers could not even receive critical consensus message. As a consequence, all peers are in frequent view-changes and could move forward.

4.2 Transactions with Infinite Loop

Clients could include an infinite loop in chaincode and send transactions to invoke the function with the infinite loop. When peers execute this kind of transactions, the execution will consume high usage of CPU. As a consequence, other normal transactions have limited CPU time to execute and the performance will severely degrade.

In Fabric version 1.0, the execute-order paradigm helps solve this problem. When the simulated execution time takes too much time, the endorsers stop

execution and reply errors to clients. In this case, malicious clients cannot collect enough valid replies from endorsers and hence the malicious transactions will not undermine committers.

5 Performance Evaluation

5.1 Overall Experimental Setting

The experiments were performed on a cluster and each server has two 2.3GHz CPU, 4GB RAM, 100GB SSD and 1Gb/s Ethernet connections with Ubuntu 16.04. We employed the modules of sending requests and checking the block status from blockbench benchmark framework. To evaluate the consensus layer, we uses `DoNothing` workload in blockbench. The chaincode does nothing and simply returns so it would reflect more about the influence of malicious behavior in the consensus layer. For transactions with an infinite loop, we design a specific chaincode to execute infinitely with high CPU consumption. For non-faulty peers, we used the original code of Hyperledger Fabric. For faulty primary and backup peers, we modified the code to implement each mode of malicious behaviors.

5.2 Malicious Behavior in Consensus Layer

Design of Malicious Behavior Patterns. Here, we discussed multiple malicious behavior patterns. The quick dictionary for all conditions is summarized in Table 1. Case **N** is used to represent baseline comparison when there are no malicious behaviors.

A. We set the primary as a non-faulty replica. **A1.** The faulty backups do not reply to any message in any phases. **A2.** The faulty backups send arbitrary messages in the three phases.
B. We set the primary as a faulty replica but the faulty primary do not delay messages. We set $\mathcal{N}_c = \mathcal{N}_p = \mathcal{N}_{pp}$ where $|\mathcal{N}_c| = f + 1$. It means a set of $f + 1$ non-faulty replicas can receive consistent messages in all phases, while the rest of replicas will receive inconsistent messages from at least one phase.
C. We set the primary as a faulty replica and the faulty replicas conduct delay attacks. The primary and other faulty replicas could delay pre-prepare, prepare and commit messages before timeout. As we discussed in Sect. 4, the impact of delay attack is different when the faulty primary sends to part of the replicas and all of them.
C1. Faulty primary sends enough consistent messages to all replicas. In other words, $\mathcal{N}_c = \mathcal{N}_p = \mathcal{N}_{pp}$ where $|\mathcal{N}_c| = 2f + 1$. Delay attack has two modes.
C1-a. Faulty replicas delay messages in all three phases. After receiving the first prepare message from non-faulty replicas, faulty replicas wait for some time and then multicast prepare messages. After receiving the first commit messages, faulty replicas wait for some time and then multicast commit messages. After multicasting commit messages, the primary will wait for some time and then

create a batch and send pre-prepare message with it. The waiting time at each phase should be no more than timeout minus round-trip time so that non-faulty replicas could receive required messages in time. In all the experiments, the waiting time is set as 1 second which is the half of the view-change timeout.
C1-b. Faulty replicas delay prepare and commit messages as they do in **C1-a**. But the faulty primary creates batches and sends prepare messages normally. In other words, whenever the number of outstanding requests reaches the batch size or the batch timer reaches 0, the primary creates a batch for it. If clients send requests fast enough, the replicas may work on the consensus process for multiple blocks concurrently.
C2. Faulty primary sends enough consistent messages to part of the replicas. We set $\mathcal{N}_c = \mathcal{N}_p = \mathcal{N}_{pp}$ where $|\mathcal{N}_c| = f + 1$. For simplicity, faulty replicas do not send messages to the rest of non-faulty replicas. Delay attack also has two modes like **C1**.
C2-a. Faulty replicas delay messages in all three phases like **C1-a**.
C2-b. Faulty replicas delay only prepare and commit messages like **C1-b**.

Table 1. Malicious behavior patterns.

No faulty behavior			N	
Non-faulty primary	Not reply		**A1**	
	Arbitrary messages		**A2**	
Faulty primary	No delay attack		$\mathcal{N}_c = 2f + 1$	$\mathcal{N}_c = f + 1$
			N/A	**B**
	Delay attack	All phases	**C1-a**	**C2-a**
		Prepare & commit phase	**C1-b**	**C2-b**

Performance. The comparisons of throughput and latency metrics are shown in Figs. 1 and 2. Comparing with the baseline condition when there is no malicious behavior, the system achieved similar results on A1, A2, and B, because in all the four cases, there exist $2f + 1$ replicas working at their best efforts to achieve consensus. For C1-b, there are $2f + 1$ non-faulty replicas working and there is no delay for pre-prepare message. When the $2f + 1$ non-faulty replicas receive pre-prepare messages, they could work well without depending on the help of faulty replicas. Thus, the system can work properly with no obvious change in throughput and latency. For C1-a, there are also $2f + 1$ non-faulty replicas working. But the pre-prepare messages are delayed and are sent at a low fixed rate. In this case, the throughput is bounded by the rate of blocks and latency increases by the delay period of pre-prepare phase. Similarly, for C2-a and C2-b, the system delayed for each phase as there are only $f + 1$ non-faulty replicas working and they have to wait for faulty replicas' messages to achieve each step of consensus process, which leads to increased latency by the delay periods of all

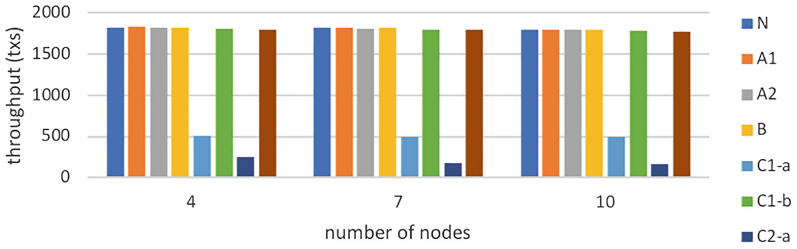

Fig. 1. Evaluation of throughput under multiple malicious behavior patterns.

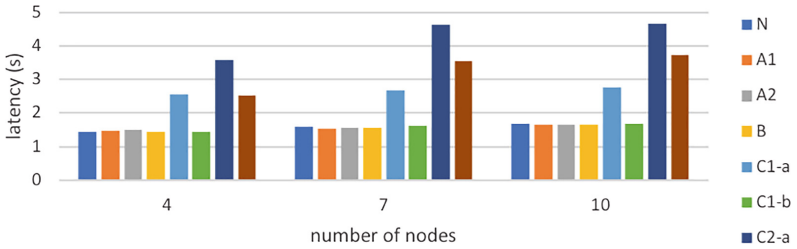

Fig. 2. Evaluation of latency under multiple malicious behavior patterns.

phases. For C2-b, as the system could work concurrently on consensus process of many blocks, the system efficiency can be guaranteed, although the time of committed the block will be delayed.

5.3 Attack of High-Rate Spam Transactions

For attacks of high-rate spam transactions in Fabric version 0.6, there are 10 peers and all of them are non-faulty. There are 5 clients and they try their best to utilize CPU and network bandwidth to send transactions to peers. The Fig. 3 shows the transactions committed per second with regard to time. During the first 58 s, the throughput is among 1200 transactions per second, which is much lower than the normal throughput, 1700 txs, shown in Fig. 1. At the 58th second, the throughput drops directly to zero. After checking the system log, the message channel of peers are full and peers reject any more message from each other, which causes all the peers to repeatedly send view-change.

5.4 Transactions with Infinite Loop

For attacks of Transactions with an infinite loop, specific chaincodes are written in go for both Fabric version 0.6 and version 1. The chaincodes for two versions are almost the same except some minor syntax. For both versions, there are 4 peers and all of them are non-faulty. The experiment result shows that in Fabric version 0.6 only two transactions could occupy all the CPU time. The normal transactions sent by clients are never executed or committed. In contrast, Fabric

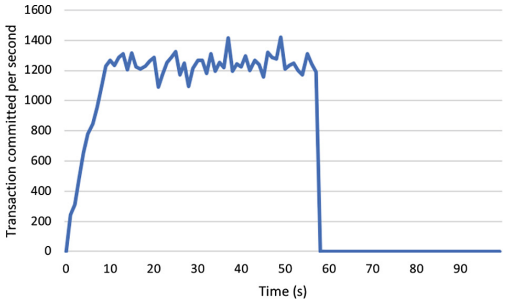

Fig. 3. Transaction committed per second under high-rate spam transactions.

version 1 performs well and are immune to these kinds of transactions because endorsers simply reject the transaction after seconds.

6 Conclusion

In this paper, we theoretically analyze the malicious behaviors in Hyperledger Fabric. We measure the blockchain performance under the designed faulty patterns. The results suggest that delay attack along with keeping some replicas out have a notable impact on the system performance. The malicious behaviors undermine the system most when the primary delay in all three phases and only sends enough consistent pre-prepare messages to $f + 1$ non-faulty replicas. For two kinds of denial-of-service attacks, Hyperledger Fabric version 0.6 fails in both cases and could not be available for normal clients. For Fabric version 1, the execute-order-validate paradigm helps resist attacks of transactions with an infinite loop. The result demonstrates the significance of performance analysis and evaluation under malicious behaviors. Future work would include other blockchain frameworks with different architectures and consensus protocols.

Acknowledgement. The work described in this paper was supported by the National Key Research and Development Program (2016YFB1000101).

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
2. Ethereum blockchain app platform. https://www.ethereum.org/
3. Hyperledger Fabric. https://www.hyperledger.org/projects/fabric/
4. Fabric version 0.6 source code. https://github.com/hyperledger/fabric/tree/v0.6
5. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, pp. 173–186. USENIX Association (1999)
6. Androulaki, E., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference (2018)

7. Dinh, T., et al.: BLOCKBENCH: a framework for analyzing private blockchains. In: Proceedings of ACM International Conference on Management of Data, pp. 1085–1100. ACM (2017)
8. Hyperledger Caliper. https://www.hyperledger.org/projects/caliper
9. Sukhwani, H.: Performance modeling and analysis of Hyperledger Fabric (permissioned blockchain network)(2018)
10. Gilad, Y., et al.: Algorand: scaling byzantine agreements for cryptocurrencies. In: 26th Proceedings on Operating Systems Principles, pp. 51–68. ACM (2017)