Analysis of Byzantine Fault Tolerant Consensus Algorithms

*Note: Sub-titles are not captured in Xplore and should not be used

Mingyu Jo
School of Computer Science and Engineering
Chung-Ang University
Korea
mgjo@cslab.cau.ac.kr

Donghyeon Kim
School of Computer Science and Engineering
Chung-Ang University
Korea
dhkim@cslab.cau.ac.kr

Sangoh Park
School of Computer Science and Engineering
Chung-Ang University
Korea
sopark@cau.ac.kr

***Abstract*—Maintaining data consistency in distributed computing is essential to maintain system reliability. To this end, many consensus techniques that can maintain data consistency even if a failure occurs are being studied, and research is being conducted on consensus techniques for Byzantine failures that can occur in unreliable public networks. In this paper, we introduce Byzantine fault tolerance consensus techniques: Practical Byzantine Fault Tolerance, High performance and Scalable Byzantine Fault Tolerance, and Zyzzyva. And we evaluate the performance of them. Afterward, directions for future research are presented.**

Keywords—byzantine fault tolerance, consensus algorithm, distributed system

I. Introduction

In distributed computing, fault tolerance is essential to maintain data consistency. Ceph[1], a distributed storage platform, uses a fault-tolerant consensus algorithm called Paxos[2] to agree on management data between monitors. However, the Paxos algorithm can only be used in places where reliability is guaranteed, such as networks within data centers. Data reliability and consistency cannot be guaranteed if used in a public network environment where reliability cannot be guaranteed.

To maintain consistency even in unreliable networks, Byzantine fault-tolerant consensus algorithms were designed[3]. Data consistency between anonymous nodes in a wide area network can be maintained through these algorithms, and distributed ledger systems such as Bitcoin and Ethereum can be built[4].

In this paper, we describe the Byzantine fault-tolerant consensus algorithms in Chapter 2 and analyze its performance evaluation results in Chapter 3. Chapter 4 presents research directions and concludes through the performance evaluation results.

II. Byzantine Fault Tolerant Techniques

A. Practical Byzantine Fault Tolerance

Practical Byzantine Fault Tolerance(PBFT)[5] is a consensus technique that performs consensus through exchanging messages with consensus nodes. It has 3 stages for processing a request of a client: PrePrepare, Prepare, and Commit stages.

PBFT is divided into one primary node and the remaining consensus node. A client sends a signed request to the primary to request a transaction on the network. Upon receiving a request message from the client, the primary node generates Pre-Prepare message with its unique sequence number and broadcasts the message with a sign by the primary node. Backup nodes that have received the Pre-Prepare message verify the message. Backup nodes that determine the message are correct to proceed to the next stage, the Prepare stage. A node in the Prepare stage broadcasts a Prepare message. Nodes that have received 2f+1 Prepare messages enter the Commit stage. The nodes in the Commit stage broadcast a Commit message and the nodes that have received 2f+1 correct Commit messages to execute the client's request and deliver the execution result. When the client receives f+1 correct messages, it determines that the request is normally reflected in the network.

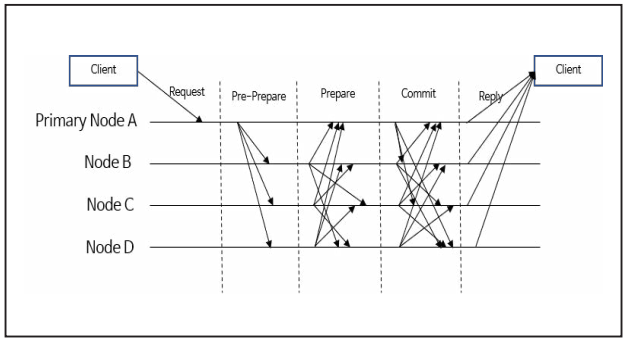

Figure 1 Practical Byzantine Fault Tolerance

B. High performance and Scalable Byzantine Fault Tolerance

High performance and Scalable Byzantine Fault Tolerance (HSBFT)[6] uses a method in which the leader node collects and broadcasts Prepare messages or Commit messages through the Prepare-Collect and Commit-Collect stages to reduce the message complexity of PBFT. Therefore, the complexity of HSBFT is O(n).

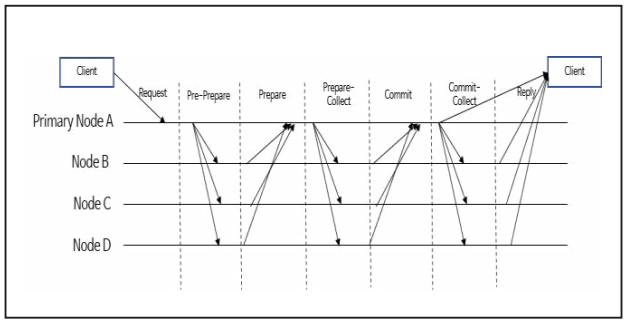

Figure 2 High Scalable Byzantine Fault Tolerance

For this, Primary is selected from among candidate nodes with large network bandwidth. In addition, the View Change protocol was improved to dynamically add/delete network nodes that were limited and static in PBFT.

However, since a centralized node called NST Center was introduced for access to new nodes and requests from clients, it takes a contradictory structure of centralization for decentralization and causes a lot of message load on the NST Center. In addition, since there are two more steps than the existing PBFT, the consensus speed may be slower than that of the PBFT when the number of nodes is small.

C. *Zyzzyva*

Most of the Byzantine fault-tolerant consensus algorithms assume that the client can be trusted, and Zyzzyva[7] uses this assumption by checking the consistency of the consensus cluster for the client. This is the technique that allows the fastest agreement in an environment without obstacles.

When a client receives a request, the primary node assigns a sequence number and broadcasts the OrderReq message to other nodes. The node that received the OrderReq message executes the request and delivers a SpecResponse message containing the execution results to the client. If the client receives 3f+1 SpecResponse messages, it is determined that the cluster is maintaining consistency. If 2f+1 to 3f SpecResponses are received, it is determined that a failure has occurred and steps are taken to restore the consistency of the cluster. A Commit message created with the received information is delivered to all nodes. The nodes that receive this forward the LocalCommit message to the client so that the agreement can be completed.

In Zyzzyva, the client must check that all nodes in the cluster have executed requests in a consistent order, but this is not possible in existing consensus algorithms such as PBFT. Therefore, Zyzzyva adds history to the message to enable verification of the order.

D. *SAZyzz*

Due to Zyzzyva's safety violations, SAZyzz[8] was proposed. SAZyzz not only solved Zyzzyva's safety violation, but also introduced a tree-based communication technique to reduce the number of messages needed to reach consensus and to implement consensus steps such as Fast Path Simple Mode, Fast Path Scalable Mode, Slow Path Simple Mode, and Slow Path Scalable Mode which are not present in Zyzzyva. So scalability, safety, and liveness were guaranteed. In addition, the cost of encryption was reduced by applying the latest encryption technique, a multi-party signature scheme.

III. Performance Evaluation

In this paper, PBFT, HSBFT, and Zyzzyva were implemented and a performance evaluation was performed. The simulation environment consists of an Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz CPU and 128GB of DDR4 RAM. We measured the consensus processing times of Zyzzyva, HSBFT, and PBFT with one client node and four consensus nodes in a LAN environment.

For additional comparison, a comparison is also performed with the Paxos algorithm. Paxos is a Crash Fault Tolerant consensus algorithm that can tolerate failures due to node suspension, and the leader node transmits the content to be agreed upon to general nodes and collects it, repeating the process twice to achieve consensus.

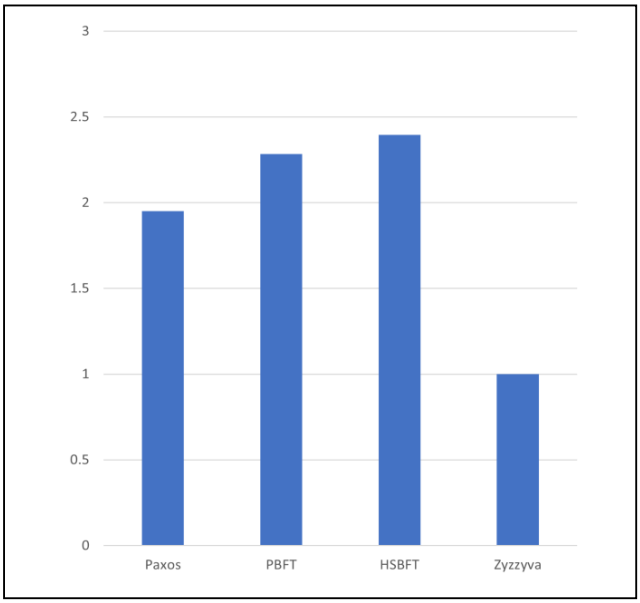

Figure 3 Result of performance evaluation.

[Figure3] shows the performance ratio of Paxos, HSBFT and PBFT times based on Zyzzyva's processing time. Zyzzyva showed the fastest results because it required the fewest steps from client request to response. In the case of HSBFT, it is a technique to improve node scalability by reducing message complexity according to the number of nodes in PBFT, but when the number of nodes is small, the Collect steps of Prepare and Commit are added, so it showed lower performance than PBFT.

Except for Zyzzyva, PBFT and HSBFT showed lower performance compared to Paxos, a Crash Fault Tolerant consensus algorithm. This appears to be due to the difference in the number of messages and the signature work to tolerate the Byzantine Fault..

IV. Conclusion

In this paper, we introduced the Byzantine fault tolerance algorithms PBFT, HSBFT, and Zyzzyva among the consensus algorithms for maintaining data consistency in a distributed computing environment. PBFT maintains data consistency through broadcasting between nodes. Because broadcast costs increase, in order to reduce the number of broadcasts, HSBFT reduced the message cost by adding a step to synthesize broadcast messages. Zyzzyva reduces the cost of consensus and maintains consistency by having trusted clients check the consistency of consensus clusters.

We performed performance evaluations on them. Zyzzyva, which uses the assumption that it is a trustworthy client, performed the best, but since it is difficult to guarantee the reliability of the client virtually, research on ways to complement this is necessary.

Acknowledgment *(Heading 5)*

This research was supported by Korea Institute for Advancement of Technology(KIAT) grant funded by the Korea Government(MOTIE)

(P0020632, HRD Program for Industrial Innovation)

References

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2023-2018-0-01799) supervised by the IITP(Institute for Information & communications Technology Planning & Evaluation)

[1] "Ceph." https://ceph.io/en/

[2] L. Lamport, "Paxos made simple," ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001), pp. 51-58, 2001.

[3] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," in Concurrency: the works of leslie lamport, 2019, pp. 203-226.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Decentralized business review, 2008.

[5] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in OsDI, 1999, vol. 99, no. 1999, pp. 173-186.

[6] Y. Jiang and Z. Lian, "High performance and scalable byzantine fault tolerance," in 2019 IEEE 3rd information technology, networking, electronic and automation control conference (ITNEC), 2019: IEEE, pp. 1195-1202.

[7] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," ACM Transactions on Computer Systems (TOCS), vol. 27, no. 4, pp. 1-39, 2010.

[8] N. Sohrabi, Z. Tari, G. Voron, V. Gramoli, and Q. Fu, "SAZyzz: Scaling AZyzzyva to Meet Blockchain Requirements," IEEE Transactions on Services Computing, vol. 16, no. 3, pp. 2139-2152, 2023, doi: 10.1109/TSC.2022.3214976.