Classification and Performance Comparative Analysis of Distributed Consensus Algorithms

Yilan Rao *

School of Dundee International College, Central South University, Changsha, 410083, China

* Corresponding Author Email: 2666085@dundee.ac.uk

Abstract. Distributed consensus algorithms are the core mechanism for ensuring data consistency and fault tolerance in distributed systems. Their evolution, from classical algorithms like Paxos and Raft to blockchain, reflects the diverse requirements for consistency, security, and performance in different application scenarios. This article analyzes the classification system of distributed consensus algorithms and divides them into two major categories based on fault models: Byzantine fault-tolerant and non-Byzantine fault-tolerant. On this basis, the system systematically organized the processes and characteristics of Paxos, Raft, Practical Byzantine Fault Tolerance (PBFT), and blockchain consensus algorithms, and performed a comparative analysis of two key performance indicators: throughput and consensus latency. Research has found that as the number of nodes increases, the performance of the traditional PBFT algorithm decreases significantly, while the Byzantine Fault Tolerant (TBFT) algorithm shows better scalability in both throughput and latency. Finally, this article combines the characteristics of various algorithms, summarizes their applicable scenarios, and provides theoretical basis and practical reference for the selection of consensus algorithms in distributed systems.

Keywords: Paxos, Raft, PBFT, Blockchain, Comparison of consensus algorithm performance.

1. Introduction

Because all applications and data were deployed on the same computer, a hardware failure would cause the entire system to collapse. With the growth of business demands, distributed computing emerged to solve this problem. It adopts the idea of 'divide and conquer,' where data is split and distributed across multiple computers, and different nodes handle different data requests in parallel. This approach solves the performance issues of large-scale data processing and has further spurred repeated innovations in the field of information technology. The core of distributed computing lies in distributed consensus algorithms, from Paxos to blockchain, demonstrating the gradual evolution of distributed computing. At present, distributed algorithms have been widely applied in various fields and play a significant role in the development of these fields. Existing research mainly focuses on parallel architecture design, performance optimization, parameter self-adaptive adjustment, and multi-objective optimization. For example, in the field of medicine, distributed parallel genetic algorithms [1], and in the field of artificial intelligence, distributed relative positioning algorithms [2]. This paper systematically classifies distributed consensus algorithms, outlines the core mechanisms of each algorithm, and compares and analyzes the performance of different algorithms under changes in node scale from the two dimensions of throughput and consensus latency, evaluating their scalability. At the same time, on this basis, analyze the typical applicable scenarios of each algorithm to provide a theoretical basis for the consensus selection of actual systems.

2. Applications of Distributed Consensus Algorithms

Distributed consensus algorithms have played a significant role in various fields. Taking oil exploration data processing as an example, traditional computing architectures have high communication overhead. As the data scale expands, frequent communication between nodes easily causes network congestion. Additionally, uneven task allocation leads to a decrease in overall resource utilization, and the limited scalability makes it difficult to support horizontal expansion, resulting in insufficient capability to process large-scale data. In contrast, Apache Spark, as an

emerging big data processing framework, relies on a distributed computing engine to schedule multi-node clusters in parallel, significantly improving data processing throughput and computational efficiency, and demonstrating high reliability [3].

3. The Development of Distributed Algorithms

3.1. Paxos

The Paxos algorithm is an important foundational algorithm for achieving multi-node consistency in distributed systems, with the function of maintaining the safety of log replication. It not only needs to ensure that all servers execute commands in the same order, but also needs to guarantee that the stopping of a minority of servers does not interfere with the smooth operation of the system. Its design goal is to ensure that multiple nodes can still reach consensus on a certain value even when some nodes fail, communication experiences delays, or messages are lost in an asynchronous network environment. The Paxos algorithm uses an asynchronous message-passing system, and its basic process includes three core stages: the Prepare stage, the Propose stage, and the Learn stage [4,5].

3.2. Raft

The Raft algorithm is a consensus algorithm built on a non-Byzantine fault model. It ensures that the logs reach consensus across the entire server cluster by using a log replication method, where more than half of the servers in a distributed environment maintain an identical log list. The Raft algorithm primarily employs the leader mechanism and state reduction methods, and defines three server states—leader, follower, and candidate—that can transition into one another under certain conditions. However, Raft cannot tolerate malicious nodes, and all read and write requests must go through the leader. When the cluster scale expands, there is a performance bottleneck, so it is not suitable for Byzantine environments like blockchain or high-latency deployments on a global scale [5-7].

3.3. PBFT

PBFT is a classic Byzantine Fault Tolerance (BFT) state machine replication algorithm, and it is also a voting-based algorithm. It has two important properties: safety and liveness. PBFT adopts a weak synchrony assumption to ensure node liveness, thereby avoiding the FLP (Fischer–Lynch–Paterson) impossibility problem. Under normal circumstances, the primary node orders all requests from clients, after which all correct nodes execute a protocol consisting of three phases: pre-prepare, prepare, and commit, in order to reach consensus [8].

3.4. Blockchain

Blockchain is essentially a distributed database or ledger that packages data into blocks, connects them in a chain structure, and ensures immutability through cryptography. Blockchain technology has core characteristics of decentralization, transparency, and immutability, and its nodes can be divided into full nodes, light nodes, miner nodes, and so on. The development of blockchain can be roughly divided into three stages. The programmable currency stage, mainly applied in the field of digital currencies. The programmable finance stage, which adds smart contract functionality. And the currently exploring programmable society stage, aimed at utilizing blockchain technology to address broader social issues [9].

4. Comparison of Distributed Algorithms

4.1. Throughput

Throughput refers to the ability of a system to process transactions in a unit of time. A commonly used metric is transactions per second (TPS), which also indicates the number of transactions the system processes per second. The expression of TPS is

$$TPS = \frac{Transaction}{\Delta t}. \quad (1)$$

Δt represents the time interval from the generation of all transactions to their writing into the block, and Transaction represents the total number of transactions written into the block during this time interval. Factors such as block size, block generation speed, and transaction injection speed affect TPS. Therefore, a fixed setting of a maximum of 2,000 transactions per block, generating a block every 5,000 ms, with a speed of injecting 2,000 transactions per second, and a total of 100,000 transactions. The number of nodes ranges from 10 to 60, with a step of 10, and the average of 100 experimental results is taken. The experimental results are shown in Figure 1 [10].

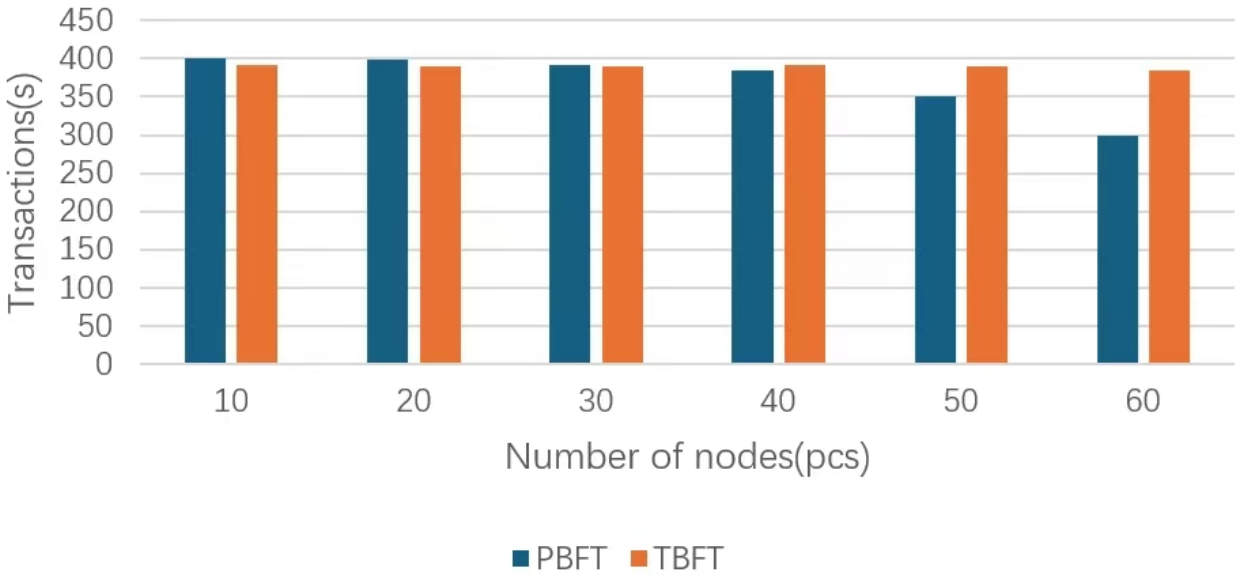

Fig. 1 TPS Test (Picture credit: Original)

According to Figure 1, it can be seen that as the number of nodes increases, the system throughput decreases. Compared to TBFT, PBFT's throughput decreases faster and its TPS is lower, and this trend becomes more apparent as the number of nodes increases. When the number of nodes is 60, TBFT's TPS is 385, while PBFT's TPS is 294, an increase in throughput of approximately 37% [10].

4.2. Consensus Delay

Consensus latency can be expressed as the time it takes for a client to receive f + 1 confirmation messages after sending a request message, without considering additional overhead such as propagation delay, transmission delay, and processing delay [10].

As shown in Figure 2, the number of nodes in the experimental setup is from 10 to 60, with a step size of 10, and the experiment is conducted 100 times [10].

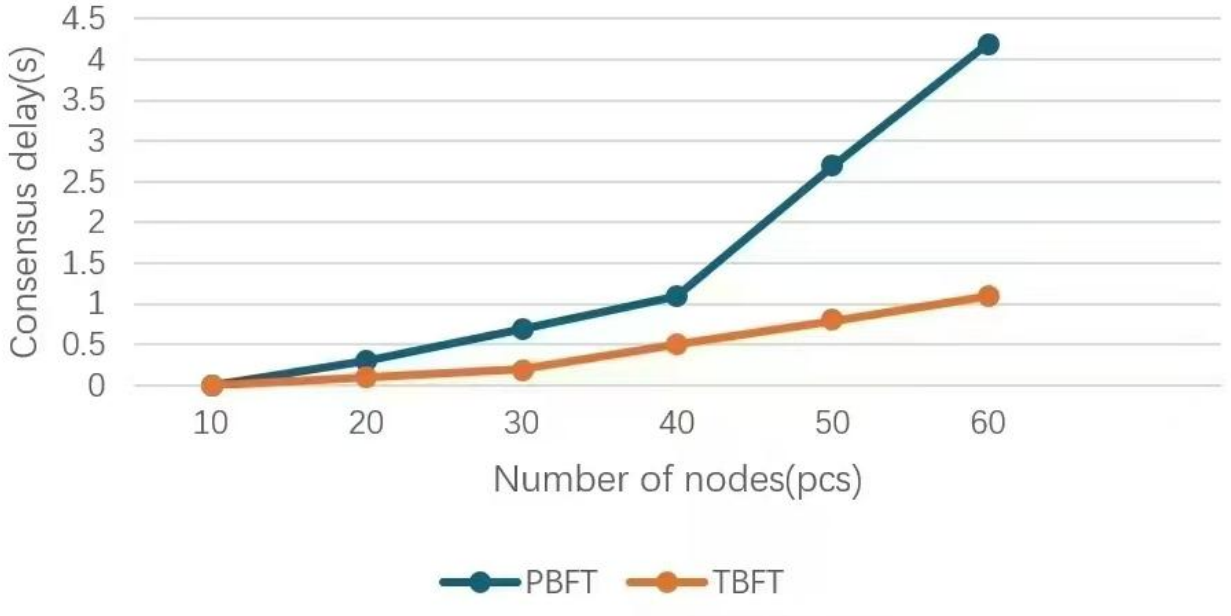

Fig. 2 Comparison of Consensus Delays (Picture credit: Original)

The experimental results show that as the number of nodes increases, the consensus latency of both consensus algorithms also rises. The more nodes there are, the faster the upward trend, but the consensus latency of the TBFT algorithm increases at a smaller rate compared to PBFT. This is because in the TBFT algorithm, Byzantine nodes and faulty nodes in the cluster can be identified and removed, reducing the number of nodes actually participating in the consensus. When the number of nodes is 60, the consensus latency of PBFT is 4.2s, while that of TBFT is 1.14s, a decrease of about 72% [10].

5. Applicable Scenarios of Different Distributed Algorithms

By combining the characteristics of various algorithms and the comparison of PBFT and TBFT in terms of throughput and consensus latency, the applicable scenarios for each algorithm can be determined. Due to its strong consistency guarantees, the Paxos algorithm is suitable for traditional distributed systems that require extremely high reliability, such as distributed databases and coordination services. However, because of its complex implementation, it is more suitable for enterprise internal environments with a limited number of nodes and relatively controllable network conditions. The Raft algorithm, due to its ease of understanding and engineering friendliness, is widely used in modern industrial-grade systems. It is particularly suitable for small to medium-sized clusters that require log replication and state machine synchronization, but it does not have Byzantine fault tolerance. The PBFT algorithm can tolerate Byzantine faults and is suitable for consortium or private blockchain environments with higher security requirements. However, its communication overhead increases significantly with the number of nodes, making it more suitable for scenarios with a moderate number of nodes. Blockchain consensus algorithms represented by PoW and DPoS are aimed at fully open network environments. PoW emphasizes decentralization and security but has lower performance, while DPoS has advantages in throughput and efficiency but has certain tendencies toward centralization. In summary, the choice of consensus algorithm needs to balance factors such as system fault tolerance, node scale, performance requirements, and degree of decentralization. In the future, attention can be paid to new consensus algorithms that combine DAG structures with BFT mechanisms to meet the demands of high concurrency and low latency in more complex scenarios.

6. Conclusion

This article systematically reviews the development trajectory and classification system of distributed consensus algorithms, from the classic Paxos and Raft to the Byzantine fault-tolerant PBFT, and to widely used consensus mechanisms in blockchain such as PoW and DPoS. By comparing the performance of PBFT and TBFT in terms of throughput and consensus latency, it analyzes the suitable scenarios for each algorithm. In summary, distributed consensus algorithms are already relatively mature, and the choice should take into account multiple factors such as fault tolerance, node scale, performance requirements, and system architecture. Although distributed algorithms still show limitations in areas such as consistency trade-offs, Byzantine fault overhead, and performance scalability, with breakthroughs in research on hybrid consensus, DAG BFT integration, and learning-based algorithms, the theory of distributed systems will be further developed, providing more reliable underlying support for distributed algorithms.

References

[1] Challa S J, Goyal N, Sharma A, et al. A Survey and Experimental Review on Data Distribution Strategies for Parallel Spatial Clustering Algorithms. Journal of Computer Science and Technology, 2024, 39(3): 610-636.

[2] Shuo W, Yongcai W, Deying L, et al. Distributed Relative Localization Algorithms for Multi-Robot Networks: A Survey. Sensors, 2023, 23(5): 2399-2399.

[3] Dezhi M, Wei W, Leiming S, et al. Application and Performance Advantage Verification of Distributed Computing in Petroleum Exploration Data Processing, 2025, 52(12): 22-24.

[4] Chunzhang L, Lele M, Da O, et al. Research on Optimization and Application of the Paxos Algorithm. Internet of Things technology, 2025, 15(17): 116-118.

[5] Jingxiong G. Optimization Migration Method Based on TLA Specification Optimization Migration Method Based on TLA Specification. Jiangxi: Nanchang University, 2023.

[6] Miao T. Research and Application of the Raft Consensus Algorithm. Henan: Zhengzhou University, 2022.

[7] Fanyao M, Xinxin L, Nan F, et al. Pipeline Log Replication Method Optimized Based on the Raft Consensus Algorithm. Telecommunication Engineering Technology and Standardization, 2025, 38(02): 41-46.

[8] Sheping Z, Chaoyue K, Rui Y, et al. RC-PBFT: An Improved PBFT Algorithm Based on Reputation Grouping. Computer Engineering, 2026, 1-12.

[9] Yong L. Research on Blockchain Consensus Algorithms Based on Evolutionary Game Theory. Jiangxi: Jiangxi University of Science and Technology, 2025.

[10] Yong X, Chuanheng S, Na L, et al. Improved Byzantine Fault Tolerant Consensus Algorithm for the Internet of Things. Computer Engineering and Design, 2025, 46(02): 360-367.