A Survey on State Machine Replication in Blockchain Systems: Protocols, Failure Models, and Implementation Strategies

Amjad Alqahtani
Department of Computer Science-Cybersecurity
University of Texas at San Antonio
San Antonio, Texas
amjad.alqahtani@my.utsa.edu

***Abstract*—State machine replication (SMR) is a fundamental technique for achieving fault-tolerance and ensuring consensus in cloud computing systems or any distributed application systems. This survey paper provides an in-depth review of SMR. The paper is highlighting SMR's critical role in building robust and reliable blockchain distributed systems. The paper also explores the state-of-the-art protocols and understands their strengths and weaknesses. Additionally, the paper delves into the failure model addressed by SMR based protocols and examines the design space that shapes how SMR is implemented in blockchain systems. By reviewing existing research, this review aims to provide a comprehensive understanding of the current landscape in the SMR based blockchain systems.**

***Keywords*—fault-tolerance, consensus, state machine, blockchain.**

I. Introduction

In general, one of the critical design requirements in distributed systems are ensuring reliability and consistency across large-scale system infrastructures, for instance AWS, Google Cloud, Microsoft Cloud, and recently P2P distributed cryptocurrency systems named blockchain. These cloud services and crypto systems demand robust mechanisms to handle failures while maintaining the state and the performance. To achieve the robustness required for reliability and consistency in large-scale distributed, State Machine Replication (SMR) is essential. State Machine Replication (SMR) has emerged as a fundamental technique for implementing fault-tolerant services. SMR offers the resilience needed to sustain crypto applications under failure scenarios [1].

In SMR technique, system reliability is implemented by replicating state across multiple nodes and enforcing deterministic execution of operations on each node of the network. The use cases span distributed databases, blockchain technology, and real-time systems, making it a cornerstone of any fault-tolerant computing.

This survey aims to provide a comprehensive overview of SMR in blockchain systems, covering main principles, key protocols, optimization techniques, and recent advancements. By categorizing existing techniques and highlighting trade-offs, the aim to offer insights into its evolution and guide for future research direction.

The rest of the paper is organized as follows. We begin with the fundamentals of SMR, including failure models and core requirements. Next, we delve into key protocols for achieving agreement and order, followed by optimizations to improve efficiency. We then discuss applications of SMR and recent advancements in blockchain.

II. Background: State machine replication (SMR) in action with blockchain example

To understand State Machine Replication (SMR) in action, let's consider a distributed ledger system based on Tendermint System [2] used by cryptocurrency like Luna Crypto. Imagine the system is designed to maintain crypto account balances for millions of customers. The accounts geographically distributed across the users or data centers. Ensuring fault tolerance is critical. If some servers fail or behave maliciously, the system must remain reliable.

In SMR, every server in the system stores a replica of the same state, such as account balances, and applies the same set of transactions (e.g., deposits, withdrawals, or transfers) in the exact order. For instance, if a customer withdraws $50 and then deposits $100. Then, replicas must process these transactions in this order to maintain a consistent state across the system, as illustrated in Figure 1. This guarantees that even if a subset of servers is faulty, the remaining servers can provide the correct state.

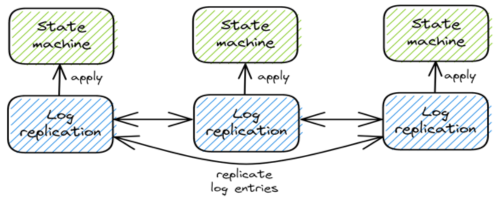

Figure 1: State machine replication

To achieve this consistency among all replicas, SMR relies on consensus protocols, such as Fast-HotStuff [3]. These protocols ensure all servers agree on the sequence of transactions to apply and make sure none of the servers is malicious. For example, in crypto system, servers must agree on the order of deposit and withdrawal requests before executing/committing them. Consensus protocols enforce three critical properties:

- **Non-triviality**: The agreed-upon value must be proposed by a server.
- **Safety**: All non-faulty servers must agree on the same sequence of values.
- **Liveness**: Eventually, all functioning servers must process some valid values.

However, achieving consensus in distributed ledger systems is challenging due to the **FLP impossibility theorem**. This means no deterministic states protocol can guarantee liveness in an asynchronous system if even one server can crash or behave maliciously. To overcome impossibility, consensus protocols such as Fast-HotStuff protocol [3], use additional techniques like failure detectors, randomness, or timing assumptions.

For instance, imagine that up to one-third of the servers of Fast-HotStuff in the crypto system could act maliciously (Byzantine faults). Fast-HotStuff ensures that all correct servers can still agree on the order of transactions by using a partial synchronous network model. Fast-HotStuff protocol guarantees **safety** (all correct servers agree on the same transaction sequence) and **liveness** (progress is eventually made) under any conditions, if the number of faulty servers does not exceed one-third of the total.

SMR has a failure model that are demonstrated in protocols such as Fast-HotStuff [3] and BigBFT [4]. These two protocols have a Byzantine failure model, which tolerate any malicious nodes up to one-third of the network nodes.

In summary, SMR provides a robust framework for fault-tolerant systems like our distributed ledger applications by replicating states across servers and leveraging consensus protocols to ensure consistency, reliability, and fault tolerance despite failures.

III. Fast-HotStuff System based on SMR

Fast-HotStuff [3] is a system based on State Machine Replication (SMR) in Byzantine systems. It was introduced by the University of British Columbia in Canada. Fast-HotStuff is a Byzantine Fault Tolerance (BFT) consensus algorithm designed to address performance bottlenecks in Byzantine Fault Tolerance/blockchain systems.

Fast-HotStuff emphasizes reducing the number of communication phases from four to three to reach consensus. This is achieved using a cryptographic technique called a quorum certificate. Fast-HotStuff provides a significant improvement compared to the state-of-the-art in BFT systems and ensures distributed consensus in fault-tolerant systems.

Fast-HotStuff prioritizes clarity by structuring the process into well-defined components such as leader election, log replication, and safety. The authors of Fast-HotStuff explained each component in detail, making the algorithm easier to grasp and implement. The algorithm has been demonstrated to be robust and practical for real-world distributed ledger systems.

A. Strengths

Fast-HotStuff reduces the number of communication phases from four to three, which is a key strength. The authors provide a detailed explanation of the algorithm's components, such as leader election, log replication, and safety. This clarity makes the protocol easier to understand and implement in practical systems. The algorithm has been demonstrated to be correct through rigorous proofs. The authors conducted extensive experimental evaluations and compared it with the HotStuff protocol, which was originally developed by Facebook.

B. Weaknesses

While the system is designed to tolerate up to one-third of malicious nodes, like other BFT algorithms, its scalability may be a problematic as the number of nodes increases significantly or if malicious behavior surpasses the tolerated threshold. The system's dependency on leader election could lead to performance degradation or vulnerabilities, especially if the leader node becomes a target of attacks or encounters failures. Although the quorum certificate enhances safety, its implementation and management introduce overhead, as the certificate must be carried forward across phases.

IV. BigBFT System based on SMR

The BigBFT protocol [4] was developed at SUNY Buffalo in collaboration with Microsoft Research. BigBFT is a multi-leader state machine replication protocol. Each leader is assigned a block number before the communication phases. The leaders then exchange protocol messages and reach consensus on all blocks. If one of the leaders fails or acts maliciously, the block position will remain unfilled because no agreement is reached and no state changes occur for that particular block.

BigBFT is divided into two phases to reach consensus on the proposed values in SMR. The first phase involves preparing values for the rest of the network, as demonstrated in Figure 2. Upon successful completion, all nodes will have the proposed messages from all other leaders.

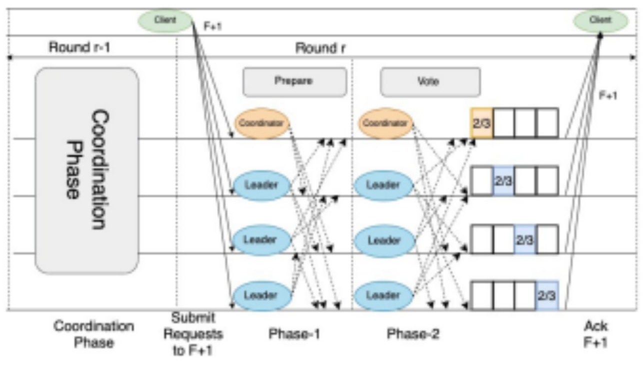

Figure 2: BigBFT based SMR [4]

The second phase, known as the vote phase, involves voting on the received proposals. If each leader collects a majority of votes for its proposal, it commits the proposal locally and waits for the next block to begin in order to commit the block globally. To optimize this process, the protocol implements a technique called piggybacking, where messages from previous blocks are carried forward to the next block. This approach eliminates one round of communication required for global commitment.

BigBFT ensures that no two nodes can decide on different values, even in the presence of failures. The protocol leverages mathematical invariants to guarantee safety. Progress is assured as long as a majority of nodes are responsive and the network remains reliable.

V. Revisiting Tendermint and New Insights

Tendermint [2] builds on the PBFT protocol, which was not originally designed to support blockchain systems. It incorporates the proof-of-stake concept to incentivize nodes to behave honestly, as malicious behavior results in the loss of their staked values within the blockchain. Tendermint employs a timeout mechanism in each phase to ensure the liveness of the protocol. Additionally, the work emphasizes practical considerations, such as reducing communication overhead and balancing the trade-offs between liveness and safety in partially synchronous environments. Tendermint's modular design, which separates the consensus and application layers through the Application Blockchain Interface (ABCI), provides developers with greater flexibility for building blockchain systems. These features enhance the applicability of Tendermint and solidify its role in advancing blockchain protocols for robust and practical use cases. However, one drawback of Tendermint is its use of waiting times to address the hidden leader problem in blockchains. This approach can lead to significant latency, as highlighted in [1].

VI. Optimizations used in SMR

The Fast-HotStuff protocol [3] is an optimized version of the earlier HotStuff protocol. Its primary optimization lies in leveraging a quorum certificate from the cryptographic domain to enhance performance. On the other hand, the BigBFT protocol [4] is more focused on engineering advancements than on research innovations. BigBFT employs piggybacking techniques to reduce the number of communication phases. It also utilizes a multi-leader system instead of a single-leader approach, mitigating the risks associated with single-leader failures and distributing the network load across multiple leaders. Additionally, it pre-assigns block identifiers or numbers before the protocol begins each round to avoid conflicts between leaders.

VII. Conclusion

I have studied the State Machine Replication (SMR) technique in blockchain systems, which serves as a fundamental method for achieving reliability and fault tolerance in distributed ledger systems. Additionally, I have explored the protocols that implement SMR. In particular, I focused on Fast-Hotstuff and BigBFT. I delved deeply into their mechanisms and design principles and built a good understanding of SMR in blockchain.

Acknowledgment

I would like to thank Professor Glenn for giving us the opportunity to work on many excellent assignments, projects, and papers in the Cloud Computing course, including this one. I gained a great deal of background knowledge and deep insights from the course tasks. Professor Glenn was always available and helpful. Many thanks to him and the course staff for making this journey both educational and enjoyable.

References

[1] S. Alqahtani and M. Demirbas, Bottlenecks in Blockchain Consensus Protocols, 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), Barcelona, Spain, 2021, pp. 1-8.

[2] E. Buchman, R. Guerraoui, J. Komatovic, Z. Milosevic, D.-A. Seredinschi, and J. Widder, Revisiting Tendermint: Design Tradeoffs, Accountability, and Practical Use, 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S), Baltimore, MD, USA, 2022, pp. 11-14.

[3] M. M. Jalalzai, J. Niu, C. Feng, and F. Gai, Fast-HotStuff: A Fast and Robust BFT Protocol for Blockchains, IEEE Transactions on Dependable and Secure Computing, 2023.

[4] S. Alqahtani and M. Demirbas, BigBFT: A Multileader Byzantine Fault Tolerance Protocol for High Throughput, 2021 IEEE International Performance, Computing, and Communications Conference (IPCCC), Austin, TX, USA, 2021, pp. 1-10.