State Machine Replication(SMR) under honest setting

Instructor: Nitin Awathare

August 31, 2024

In the previous lecture, we discussed various blockchain layers. While this classification isn't standardized, we used it to help clarify the concepts. We also outlined the focus of this lecture series and highlighted how it differs from other books and lecture series on the subject. In this lecture, we will begin with layer-1, where we will explore different consensus algorithms that operate under various assumptions.

1 The State Machine Replication (SMR) Problem

There are several notions of "consensus," and we'll cover at least three. We'll begin with the version most relevant to blockchain protocols: the state machine replication (SMR) problem, a concept dating back to the 1980s. It's remarkable—and a testament to the enduring value of theory and abstraction—that challenges from that era are still central to modern blockchain protocol design. So, what exactly is a "state machine"? If you've studied automata, like deterministic finite automata (DFAs), you're already familiar with the concept of states and state transitions. If not, some examples will clarify this.

A classic application of the SMR problem is managing a replicated database. Imagine a large company like IBM, which has a valuable database that customers access through queries and updates. To guarantee close to 100% of uptime and hence availability, they can't rely on a single machine due to the risk of hardware and software failures. The obvious solution is to replicate the database across multiple machines in different locations, reducing the likelihood of simultaneous failures. However, with multiple copies, a new challenge arises: keeping them in sync. For example, if a customer updates one copy, the change must be reflected across all copies so that queries return consistent results. Here, "state" refers to the current contents of the database, and each update triggers a "state transition," moving the database from one state to another that reflects the change.

On the other hands, in the context of blockchain, the state consists of two main parts: 1. Smart Contract State: This includes the state of all smart contracts on the blockchain, represented as key-value pairs where the keys are contract variables. Smart contracts also contain methods that modify these variables. 2. User State: This typically includes user accounts and their balances, also represented as key-value pairs. Executing a transaction, such as a payment between accounts, triggers a state transition, updating the state to reflect the new account balances. Unlike databases, where replication mainly aims to improve uptime, blockchain replication primarily seeks decentralization—distributing responsibility across many machines to prevent any single entity from controlling the state or execution of the protocol.

In both database and blockchain scenarios, the objective is to keep multiple nodes in sync so they all follow the same sequence of state transitions (database operations or transaction executions) and agree on the current state (database contents or blockchain state). This is known as the SMR problem. To summarize: 1. A set of nodes runs a consensus protocol, while clients can submit transactions to these nodes. 2. Each node maintains a local append-only data structure—an ordered list of transactions, which we'll call its history. While some refer to this as a ledger, the term often implies a focus on payments, which underestimates the broader potential of blockchain protocols as a new computing platform.

Informally, the goal of the SMR problem is to deploy code that keeps all nodes synchronized with the same local histories (i.e., identical ordered sequences of transactions). In this lecture, we aim to design and study protocols that solve the SMR problem. Once we have a protocol, the key question is: what makes it a "correct" solution? This involves defining the guarantees we expect, such as safety guarantees (ensuring certain bad events never occur) and liveness guarantees (ensuring certain good events eventually happen). We'll discuss this individually below.

1.1 Consistency

A protocol satisfies consistency if all nodes agree on the transaction history, with each node's local history being a prefix of others' histories (allowing for some delay). The key is that no two nodes should disagree on the

order of transactions. Consistency ensures this by preventing such disagreements. However, consistency alone isn't enough; we also need a guarantee that work will eventually be completed, or else the protocol would be ineffective, as even an empty protocol would meet the consistency criterion.

1.2 Liveness

Every transaction submitted to at least one node will eventually be added to every node's local history. Whether protocols can solve the SMR problem by satisfying both consistency and liveness depends on factors like network reliability and the number of compromised nodes. Upcoming lectures will cover key results on SMR consensus, showing how these results help compare different layer-1 protocols, some prioritizing liveness and others consistency. We will start with a protocol that works under ideal network conditions and many compromised nodes, and later explore protocols for weaker network assumptions but with fewer compromised nodes.

Meeting either consistency or liveness alone is trivial, but achieving both simultaneously is challenging. Over the next few lectures, we'll explore how the existence of SMR protocols that satisfy both properties depends on factors like network reliability, the proportion of malicious participants, and any pre-protocol setup allowed. In this lecture series, we will explore possibility results that outline the conditions under which achieving both consistency and liveness is feasible. We will also examine impossibility results that detail the conditions under which it is not achievable.

2 Assumptions

We'll start this lecture with a set of assumptions, more than usual, making it easy to find protocols that satisfy both liveness and consistency. We'll then gradually relax these assumptions, developing more sophisticated and robust solutions to the SMR problem. Here are four assumptions that we are going to consider while solving the SMR problem.

2.1 Permissioned Network

The first assumption is that the set of nodes running the protocol is fixed and known in advance. Each node knows the identities and IP addresses of all other nodes. We'll denote the number of nodes as n, with distinct names 1, 2, 3, ..., n, allowing for direct communication among them.

To effectively discuss blockchains like Bitcoin and Ethereum, we need to transition from permissioned to permissionless settings. However, starting with the permissioned setting is beneficial because the impossibility results for this simpler case also apply to the more complex permissionless scenario. Additionally, developing protocols for permissionless systems often begins with the permissioned case as a special scenario, then extends to the general case. Many prominent blockchain protocols follow this approach, reducing permissionless consensus to permissioned consensus.

2.2 Public Key Infrastructure (PKI)

The PKI assumption extends the permissioned setting by requiring that all nodes not only know each other's IP addresses but also have distinct public-private key pairs, with all public keys being common knowledge at the start. Each node can verify signatures from other nodes using these public keys. This assumption is stronger than simply having cryptography, as it presumes that public key distribution is correctly handled before the protocol begins. While various methods for public key distribution exist, we assume it has been done correctly. This assumption is less likely to be problematic, and some blockchain protocols, like Bitcoin and early Ethereum, do not rely on PKI.

2.3 Synchronous Network

We will assume an optimistic communication model called the synchronous model, which includes:

Shared Global Clock: All nodes agree on the exact time and time steps, even without communication. While this is not realistic due to clock drift, it's a useful theoretical approximation.

Bounded Message Delays: Messages sent at the start of a time step arrive before the next time step begins. For instance, messages sent at 80 seconds arrive before 90 seconds. This idealized assumption is often unrealistic in practice, especially on the Internet, where network outages and attacks can occur.

This synchronous model helps establish a baseline for evaluating blockchain protocols, ensuring they perform well under ideal conditions. However, real-world protocols must also handle network unreliability. When assessing a protocol, consider its resilience to prolonged outages and attacks: does it compromise liveness, consistency, or both?

2.4 All Nodes are Honest

The final assumption is that all nodes are honest, meaning they follow the protocol correctly and without deviations or bugs. This assumption is overly strong, even for older applications like IBM's database replication, where servers may occasionally fail. We will first design a consistent and live SMR protocol under this assumption, and then develop a more robust protocol that maintains consistency and liveness even if some nodes deviate from the protocol.

3 Solution to SMR with a Round-Robin Leaders

Implementing the rotating leaders concept is straightforward under our previous assumptions. In the synchronous setting, a shared global clock ensures all nodes agree on the current time step. In the permissioned setting, the set of nodes and their names are known in advance, allowing everyone to follow the round-robin order for leadership. The leader for each time step coordinates by:

- Collecting all pending transactions it knows about and ordering them arbitrarily.
- Sending the ordered list of transactions to all other nodes.

By the start of time step t, every node will have received the list from the leader of time step $t-1$. Each node, including the previous leader, appends this list to its local history. Thus, nodes broadcast ordered lists when they are leaders and continuously update their histories with received lists. Given the above protocol let's discuss the formal proof of consistency and liveness.

Theorem 1. The round-robin leader protocol satisfies both consistency and liveness under a permissioned setup, a synchronous network, all nodes behaving honestly, and with a PKI in place.

Proof. Let's start with consistency, the safety property that ensures no two nodes ever disagree on the order of transactions. This protocol guarantees consistency because all nodes operate in lockstep: during each time step t, the leader sends the same transaction list to every node. In the synchronous model, these messages arrive before time $t+1$, and at the start of $t+1$, all nodes append these identical lists to their local histories. Since nodes begin with identical histories, they remain synchronized indefinitely.

For liveness, consider that if a transaction is submitted to at least one node, it will eventually be included. Since each node takes its turn as the leader (every n time steps, where n is the number of nodes), a node that knows about the transaction will eventually lead and include it in the broadcasted list. □

References:

1. Foundations of Blockchains `https://timroughgarden.github.io/fob21/`.
2. Blockchain gets better: moving beyond Bitcoin
`https://www.comp.nus.edu.sg/features/2018-blockchain-gets-better/`

Disclaimer :

Some of the content and/or images in these notes have been directly sourced from the books and/or links cited in the references. These notes are exclusively utilized for educational purposes and do not involve any commercial benefits.