State Machine Replication(SMR) in Partially Synchronous Network

Instructor: Nitin Awathare

September 16, 2024

Till now we have discussed the two extremoties of the network synchrony, i.e., Synchronous Network and Asynchronous Network. We rejected the synchronous model due to the inevitability of network outages and prolonged attacks. Instead, we consider the partially synchronous model, which distinguishes between "normal" periods (modeled synchronously) and "under attack" periods (modeled asynchronously). The goals are then to: 1) Achieve all desired properties (e.g., safety and liveness) during "normal operation conditions"; 2) Minimize damage during attacks (e.g., sacrifice only safety or liveness); 3) Recover quickly after an attack once conditions normalize.

We'll use a minimal model with one synchronous phase and one asynchronous phase. To study recovery, the model will start with the asynchronous phase followed by the synchronous phase. (Chaining multiple instances of this model can simulate arbitrary alternations.). Furthermore, we can model the partially synchronous network in two ways, which we will discuss along with their formal representation below:

- **Model with Global Stabilization time (GST):**
  This partially synchronous model is defined by Dwork, Lynch, and Stockmayer, that incorporates a shared global clock and two phases: synchronous and asynchronous. Here are the key elements:
  - Timing Assumptions: A global clock ensures nodes agree on time steps, even during attacks. This is a less restrictive assumption compared to the synchronous model.
  - Synchronous Phase: Defined by a parameter Δ, which specifies the maximum message delay. This parameter is known upfront and influences protocol design.
  - Global Stabilization Time (GST): The transition from synchronous to asynchronous mode occurs at an unknown GST. Protocols must handle this transition without prior knowledge.

  These key properties put following constraints on the message delivery:

  - During the asynchronous phase, messages sent by time $t \leq GST$ are received by time $GST + \Delta$.
  - During the synchronous phase, messages sent by time $t \geq GST$ are received by time $t + \Delta$.

  The partially synchronous model, where GST is unknown, is more general than the synchronous model and ensures that messages will eventually be delivered.

- This version connects to our earlier discussion on relaxing the synchronous model (with fixed delay $\Delta = 1$) to one with variable message delays up to a known Δ. While that version guarantees message delivery within Δ steps, the protocol speed adapts to network speed, unlike the GST version. In this model, the delay Δ is unknown and cannot be factored into the protocol.

Both these versions of the partially synchronous model are defined by Dwork, Lynch, and Stockmeyer. Throughout this lecture, we focus on the "GST" version we've discussed so far. However, the second version is worth mentioning. The two versions of the partially synchronous model—the "GST version" and the "unknown Δ" version—each have distinct goals: quick recovery after an outage for the GST model and adaptability to network speed for the unknown Δ model. Despite their differences, they are grouped under "partial synchrony" because results for one version often hold in the other. For instance, impossibility and possibility results tend to be consistent across both. While not formally equivalent, they're close enough for most purposes.

The FLP impossibility result shows that no consensus protocol can guarantee termination, validity, and agreement in the fully asynchronous model. However, the partially synchronous model, being less general due to message delivery constraints, is not directly affected by FLP. During the asynchronous phase of partial synchrony, FLP still applies, meaning protocols can't guarantee termination until after the global stabilization time (GST). Thus, liveness can't be assured pre-GST without risking validity or agreement violations, but post-GST, both safety and liveness can be guaranteed.

During the asynchronous phase, the natural and widely accepted choice is to give up liveness while preserving safety. Safety ensures "nothing bad ever happens," even during the asynchronous phase, while liveness, meaning "something good eventually happens," can be interpreted as applying only after the global stabilization time (GST).

Many blockchain consensus protocols focus on preserving safety while giving up liveness during asynchrony, which will be our focus in this lecture. However, on the other hands, the longest-chain consensus takes a different approach, prioritizing liveness over safety during periods of asynchrony.

Like we discussed in the previous chapters there are impossibility results with respect to the Partially Synchronous Network, which we will discuss next. Before we delve down in to the formal arguments, lets discuss the intuition behind this impossibility (stated in Theorem 1).

In the synchronous model, to ensure termination, an honest node must halt with an output even if it hears nothing from some nodes, as Byzantine nodes might stay silent. Since up to f nodes can be Byzantine, an honest node can only rely on receiving messages from $n - f$ nodes before making a decision. In the partially synchronous model, however, delayed messages from honest nodes (due to an unknown GST) may be mistaken for Byzantine silence. This uncertainty means that if a majority of $n - f$ nodes are Byzantine, an honest node might be misled, requiring f to be less than $1/2(n - f)$, which ultimately is reduced to $f < n/3$.

This argument underscores the challenge of Byzantine nodes colluding with adversarial message delivery. Both can enforce silence from f nodes for an extended time, leaving honest nodes unable to tell if the issue is Byzantine or network-related. Effectively, each Byzantine node neutralizes two honest nodes—one ignored due to delayed messages and another confused by conflicting messages. To ensure at least one honest node remains unaffected, we need $(n - f) - 2f > 0$, or equivalently, f must be less than $n/3$.

Theorem 1. In the partially synchronous model, with or without the PKI assumption, for every f and n with $f \geq n/3$, no Byzantine agreement protocol achieves the goals, i.e., agreement (always), validity (always), and termination (eventually, possibly only after the GST).

Proof. We focus on the simplest case of the impossibility result with $n = 3$ and $f = 1$ (three nodes, one of which may be Byzantine). Despite its simplicity, this case fully captures the complexity of the general result. Let's label the nodes Alice (A), Bob (B), and Carol (C). Suppose there's a protocol π that guarantees agreement, validity, and termination. In this scenario, Alice is honest with input 1, Carol is honest with input 0, and Bob is Byzantine (his input is irrelevant). This setup leads to a contradiction, illustrating the core of the impossibility result.

Consider Bob using a Byzantine strategy with the aid of an adversary controlling message delivery:

- The adversary delays messages between Alice and Carol indefinitely.
- Bob interacts with Alice as if his private input is 1 and he hasn't received messages from Carol.
- Bob interacts with Carol as if his private input is 0 and he hasn't received messages from Alice.

This strategy is feasible whether or not the PKI assumption holds. Since the GST can be arbitrarily large, the adversary can delay messages between Alice and Carol for an extended period.

From Alice's perspective:

- She receives conflicting information from Bob and silence from Carol.
- She may suspect Bob is Byzantine and Carol's messages are delayed. However, Bob might be honest, and Carol could be Byzantine.

To hedge against this, Alice must eventually output her result based on validity (same input from honest nodes). Since she receives no input from Carol, she outputs 1.

From Carol's perspective:

- She might suspect Bob is Byzantine and Alice's messages are delayed, or that Bob is honest and Alice is Byzantine.
- Carol must output her result by some time T2 based on validity (same input from honest nodes). Since she receives no input from Alice, she outputs 0.

With the adversary ensuring messages between Alice and Carol are delayed for $max\{T1, T2\} + 1$ time steps, Alice outputs 1 and Carol outputs 0, leading to a contradiction since both Alice and Carol are honest. This outcome violates the agreement property of the protocol π.

$\square$

Next we will discuss one more impossibility, called CAP theorem.

1 CAP Theorem

The CAP Principle is a key concept in distributed systems, where "CAP" stands for three informal properties:

- C: Consistency — The system should provide the same result regardless of which replica handles the request, akin to a centralized system.
- A: Availability — Every request should eventually be fulfilled, similar to liveness in SMR protocols.
- P: Partition Tolerance — The system should continue to function despite network partitions or communication failures.

The CAP Principle asserts that a distributed system cannot simultaneously guarantee all three properties. Specifically, if a system must handle network partitions, it must forgo either consistency or availability. The CAP Principle highlights a design trade-off in distributed systems. When faced with a network partition, a node must choose between consistency and availability. For instance, if a node receives a command to update a variable x and then faces a partition, it might either:

- Answer queries with the updated value but risk inconsistency with nodes that haven't received the update, or
- Stick with the old value to maintain consistency but sacrifice availability.

Designing a distributed system involves:

- Assessing Partition Concerns: Determine if your system needs to handle network partitions (e.g., operating over the Internet).
- If No Partition Concern: You can demand both consistency and availability.
- If Partition Concern: Decide whether to prioritize consistency or availability.

For example:

- A bank might prioritize consistency over availability to prevent financial discrepancies.
- A search engine might prioritize availability over consistency to avoid downtime.

In blockchain protocol design, this trade-off mirrors choices between protocols that sacrifice liveness (e.g., Tendermint) or consistency (e.g., longest-chain consensus) when under attack.

2 FLP Impossibility Theorem vs. CAP Principle

The CAP Principle and the FLP impossibility result both highlight the trade-off between safety/consistency and liveness/availability during network attacks. Despite their relevance, the FLP result is typically covered in distributed computing theory, while the CAP Principle is discussed in system design courses. Blockchain experts should understand both and their distinctions. The major similarities and/or distinctions are listed below:

- The FLP impossibility proof is complex, while the CAP Principle's argument is simpler, due to the CAP setup allowing for messages to be delayed indefinitely or dropped.
- The CAP Principle is easier to argue because it involves an adversary that can delay or drop messages indefinitely, unlike the FLP result where the adversary must eventually deliver all messages. This makes the CAP argument more straightforward.
- The FLP adversary can act more freely compared to the CAP adversary, who is limited to network partitions. Network partitions are a key attack vector that captures much of the power of more general adversarial strategies.
- The CAP Principle's adversary is so powerful that it makes the argument valid even with all honest nodes ($f = 0$). In contrast, the FLP result requires at least one faulty node to hold.

The FLP impossibility result can be adapted to apply even with a single crash fault, which is a benign type of fault where the node is honest until it crashes. This is analogous to infinite message delays—if a node crashes just before sending messages, it's like those messages are indefinitely delayed. Essentially, the FLP result shows that even the potential for a single crash fault is sufficient to demonstrate the trade-off between safety and liveness, similar to what is observed with infinite message delays.

In next lecture we will discuss a consensus protocol can ensure safety and eventual liveness (post-GST) if fewer than one-third of nodes are Byzantine. Specifically, we will discuss Tendermint, which supports several major blockchain protocols.

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 :