Practical Byzantine Fault Tolerance (pBFT)

Return to Consensus Mechanism

The following definition/explanation is provided by Curran [1)]

*The **Practical Byzantine Fault Tolerance (PBFT)** model primarily focuses on providing a practical Byzantine state machine replication that tolerates Byzantine faults (malicious nodes) through an assumption that there are independent node failures and manipulated messages propagated by specific, independent nodes.*

The algorithm is designed to work in asynchronous systems and is optimized to be high-performance with an impressive overhead runtime and only a slight increase in latency.

- *Essentially, all of the nodes in the pBFT model are ordered in a sequence with one node being the primary node (leader) and the others referred to as the backup nodes.*
- *All of the nodes within the system communicate with each other and the goal is for all of the honest nodes to come to an agreement of the state of the system through a majority.*
- *Nodes communicate with each other heavily, and not only have to prove that messages came from a specific peer node, but also need to verify that the message was not modified during transmission.*

For the pBFT model to work, the assumption is that the number of malicious nodes in the network cannot simultaneously equal or exceed ⅓ of the overall nodes in the system in a given window of vulnerability.

The more nodes in the system, then the more mathematically unlikely it is for a number approaching ⅓ of the overall nodes to be malicious. The algorithm effectively provides both liveness and safety as long as at most (n-1) / ⅓), where n represents total nodes, are malicious or faulty at the same time.

The subsequent result is that eventually, the replies received by clients from their requests are correct due to linearizability.

Each round of pBFT consensus (called views) comes down to 4 phases. This model follows more of a "Commander and Lieutenant" format than a pure Byzantine Generals' Problem, where all generals are equal, due to the presence of a leader node. The phases are below.

1. *A client sends a request to the leader node to invoke a service operation.*
2. *The leader node multicasts the request to the backup nodes.*
3. *The nodes execute the request and then send a reply to the client.*
4. *The client awaits f + 1 (f represents the maximum number of nodes that may be faulty) replies from different nodes with the same result. This result is the result of the operation.*

The requirements for the nodes are that they are deterministic and start in the same state. The final result is that all honest nodes come to an agreement on the order of the record and they either accept it or reject it.

The leader node is changed in a round-robin type format during every view and can even be replaced

with a protocol called view change if a specific amount of time has passed without the leader node multicasting the request.

A supermajority of honest nodes can also decide whether a leader is faulty and remove them with the next leader in line as the replacement.

1)

Brian Curran, Blockonomi, What is Practical Byzantine Fault Tolerance? Complete Beginner's Guide, 18 April 2020, Accessed: 18 July 2021, https://blockonomi.com/practical-byzantine-fault-tolerance/#An_Overview_of_Practical_Byzantine_Fault_Tolerance