Module 1 Lecture 2: **Principles of Consensus Protocols**
Prepared by Gwyneth Ang for *NTU CE/CZ4153 Intro to Blockchain*

---

1. Faults and Fault Tolerant Systems

Faults are conditions that causes the software to fail to perform its required function. Faults in systems are everywhere and may sometimes be inevitable. However, we want our system to be dependable; only then can we trust the system enough to accept the services it delivers. A dependable system should be:

- **Available**
  Ready for use when needed. More formally, it refers to the probability that the system is operating correctly at any given moment and is available to perform functions on behalf of its users.
- **Reliable**
  Able to provide continuity of service while it is being used; system can run continuously without interruption or failure for long time periods.
- **Safe**
  No catastrophic events occur when the system temporarily fails to operate correctly.
- **Maintainable**
  Failed system can be repaired easily, automatically, if possible.

For a system to achieve dependability, we require fault tolerance techniques at execution time. Fault Tolerance refers to the ability of a system to continue operating properly despite the failure of one or more components. As much as we try to avoid faults in the design phase, there are bound to be faults in any system. Faults cause errors which lead to failures. However, a system with fault tolerance will allow the system to continue operating in an acceptable way despite the presence of faults and failures occurring.

2. Synchrony Assumptions (Communication Models)

Synchrony, Asynchrony and Partial Synchrony
Read Recommended Reading:
https://decentralizedthoughts.github.io/2019-06-01-2019-5-31-models/

Achieving Fault Tolerance
Fault tolerance can be achieved through multiple ways, such as replication or consensus. Replicating processes into groups increases process resilience, thus protecting against process failure. If one or more processes are faulty, other identical processes can mask the faulty ones. Consensus ensures that the whole group behaves as "a single, highly robust process". Each non-faulty process executes the same user-defined commands or incoming user requests in the same order to maintain homogeneity.

3. Consensus Protocols

Consensus achieves overall system reliability in the presence of faulty processes by getting different parties to come to an agreement. If we are trying to carry out a transaction to transfer funds from one account to another (think of a distributed account-based banking ledger for the time being), the processes involved must consistently agree to perform the debit and credit operations.

Consensus vs Agreement
The consensus problem is a generalization of mutual exclusion, election, and many other problems.

Read the Recommended Reading to understand the consensus problem and its variants:
https://decentralizedthoughts.github.io/2019-06-27-defining-consensus/

Safety and Liveness
Properties of interest in distributed systems fall into two categories – safety and liveness. A safety property maintains that nothing bad happens during execution; the program does not reach a bad state. A liveness property maintains that something good will eventually happen; the program eventually reaches a good state. Safety and liveness should be defined through the context of the specific protocol in consideration. In the context of the Agreement problem, safety asserts that agreement is upheld throughout execution, there will not be any two honest nodes that decide on a different value. Liveness asserts that the program will eventually terminate – all honest nodes will eventually decide on a value and terminate.

Crash Failure Tolerant Consensus
Paxos is the most traditional consensus protocol. Among all the proposed values, Paxos ensures that a single value is chosen. To understand how Paxos works, you may read (optional reading):
https://lamport.azurewebsites.net/pubs/paxos-simple.pdf

If there are to be multiple values to be agreed upon, Multi-Paxos or Raft can be employed. Multi-Paxos is an optimization on using Paxos for several consecutive rounds. It removes the overhead by skipping one of the phases as the leader is stable. To understand how Raft works, you may read (optional reading): https://raft.github.io/

However, in most permissionless blockchain systems, arbitrary failures occur too. Arbitrary failures, aka byzantine failure, refer to a type of failure where any error may occur, such as collusion with other participants, selective non-participation, fabrication of messages etc. Therefore, crash failure tolerant consensus is not enough as malicious actors can try to "bias the outcome" to their favour.

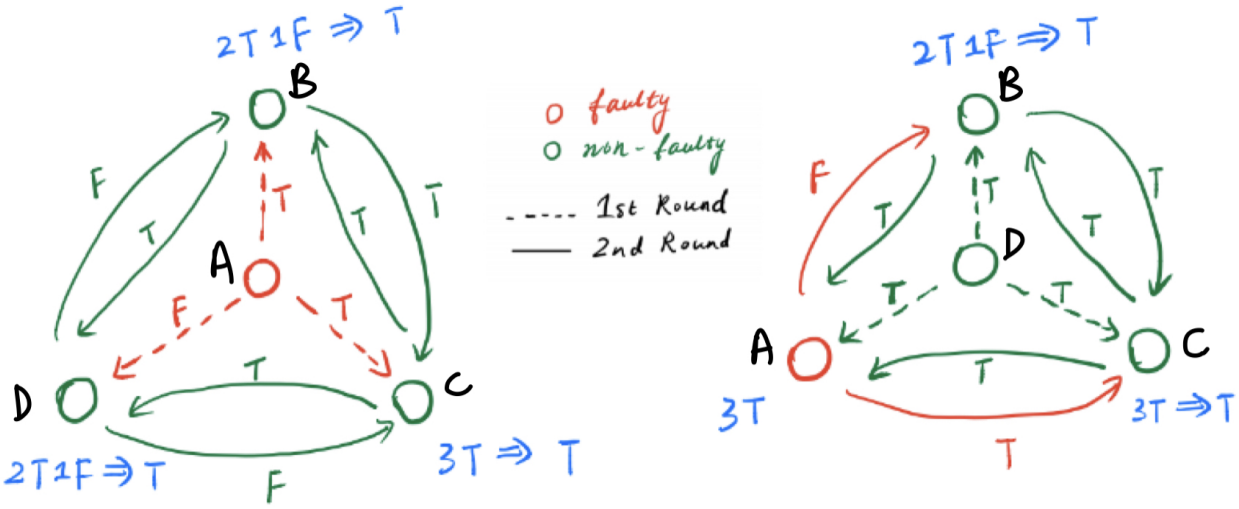
4. Byzantine Generals Problem

Assume that there are a group of generals who are trying to agree on a strategy – whether to attack a city or retreat. Every general must finally agree on a common decision. They must come to a consensus, as an uncoordinated strategy will lead to defeat. However, there are treacherous generals present in the group who may choose the suboptimal strategy that favours the opponents.

Assuming there are f treacherous generals in the group, how many generals must there be in the group to reach consensus? We need at least 3f + 1 generals in the group for consensus.

Assume f = 1, i.e., there is 1 dishonest general amongst 4 generals. In this message exchange, there are two phases. In the first phase the commanding general (leader) shares with the other generals what the strategy is. In the second phase, all generals share with each other what strategy they received, to confirm the strategy.

Referring to the diagram below, let us explore two cases here. To promote reliability and scalability, we use replication in distributed systems. Replication refers to organizing data such that each data element is copied and stored in several components within a system.



Starting with the case where the leader is a careless general (depicted left on diagram), he may send different messages to different generals. Despite this, Generals B, C and D can still come to an agreement as they derive that the "strategy" to take is 'True'.

In the next case(right), even though General A, the bad actor, is no longer the leader, he is also not able prevent the other generals from coming to a consensus. Can you prove this in general, for any f?

5. Limitations and Lower Bounds

CAP Theorem
A distributed system cannot achieve all three of Consistency, Availability or Partition Tolerance. You can only pick two and must sacrifice one of the three.

Definitions from Gilbert and Lynch https://dl.acm.org/doi/10.1145/564585.564601:

- **Consistency**
  "Atomic, or linearizable, consistency is the condition expected by most web services today. Under this consistency guarantee, there must exist a total order on all operations such that each operation looks as if it were completed at a single instant. This is equivalent to requiring requests of the distributed shared memory to act as if they were executing on a single node, responding to operations one at a time."

- **Availability**
  "For a distributed system to be continuously available, every request received by a non-failing node in the system must result in a response. That is, any algorithm used by the service must eventually terminate ... [When] qualified by the need for partition tolerance, this can be seen as a strong definition of availability: even when severe network failures occur, every request must terminate."

- **Partition Tolerance**
  "In order to model partition tolerance, the network will be allowed to lose arbitrarily many messages sent from one node to another. When a network is partitioned, all messages sent from nodes in one component of the partition to nodes in another component are lost. (And any pattern of message loss can be modeled as a temporary partition separating the communicating nodes at the exact instant the message is lost.)"

However, almost all distributed systems require partition-tolerance. For a distributed system to not require partition-tolerance, it would have to guarantee that the network will never drop messages and the nodes are guaranteed to never die. These systems do not exist, so an equivalent statement of the CAP theorem is: In the presence of a network partition, one can only choose between Consistency and Availability.

FLP Impossibility
To understand more, read:
https://decentralizedthoughts.github.io/2019-12-15-asynchrony-uncommitted-lower-bound/