Topic 7: Consensus

- Explain the consensus problem
- Solution in synchronous system
- Explain the Byzantine generals problem.
- Present impossibility result for 3 Byzantine generals, 1 faulty (argue carefully!)
- Present the solution for 4 Byzantine generals, 1 faulty.
- Present clearly your assumptions on system model, failures, and message signing.
- Discuss impossibility in asynchronous systems and practical workarounds.

[DS5ed] 15.5

The Consensus Problem

The Consensus problem is a type of Agreement problem, which generally is for processes to agree on a value after one or more of the processes has proposed what that value should be.

More concretely, all processes begin in the *undecided* state. Each process proposes a value, and proposes a value. The processes communicate, and then each process enters the *decided* state.

Consensus algorithms require the following:

- **Termination**, eventually each correct process P_i sets its decision variable D_i
- **Agreement**, the decision variable value of all correct processes are the same: if P_i and P_j are correct and have entered their decided state, then D_i = D_j (for all i,j ∈ 1..N)
- **Integrity**, if the correct processes all proposed the same value, then any correct process in the decided state has chosen that value. (Could be a weaker definition for certain applications.)

Consensus in a Synchronous System

Very simple, if no failures:

- Each process multicasts its value to all processes
- Waits until a values are received from all processes.
- Makes a decision.
- *f*-resilient consensus algorithm
  - ** Solves consensus for *f* failed proceses
- ** Example: *f+1* rounds
  - **Round 1**:
    - Each process multicasts its value

        - Wait for values from other processes
    - **Round 2**:
        - Each process multicasts the values received from other processes in round 1
        - Wait for values
    - ...
    - **Round *f+1***:
        - Select the minimum value received as the decision value.

- Requires atleast *f+1* rounds, *f* being the number of failed processes.

- When a round is reached with no failure, every process would decide on the same value.

The Byzantine Generals Problem

Three or more generals are to agree on whether they should attack or retreat. One is a commander, and he issues the order. The others are lieutenants, and must decide whether they attack or retreat. A process may be treacherous, in which case it issues different signals to different peers.

This differs from consensus, since there is a single process whose value the others must agree on.

Also requires

- **Termination**: Eventually each correct process sets its decision variable.
- **Agreement**: The decision value of all correct processes is the same.
- **Integrity**: If the commander is correct, then all correct processes decide on the value that the commander proposed.

- ** Is based around the turkish invasion into Byzantium.
- ** generals must agree to *attack* or *retreat*, while only being able to communicate by messengers.
- ** Messages are safe, they cannot be intercepted or tampered with.
- ** Traitor generals can exist
    - Traitors can omit answering messages from generals
    - Send arbitrary messages
        - Byzantine failures = arbitrary faults including:
            - Omission failures, crash failures, send different values to different processes
            - Being intentionally malicious, buggy
- ** Attacking with only one army means defeat.

Cost of Byzantine Generals

- ** Requires f+1 rounds, in the general case, f>=1
- ** Sends O(N^f+n) messages.
    - ** If using digital signatures, a solution exist with only O(N^2) messages, still f+1 rounds)
        - If a process says another process said something else, it can be detected.
- ** As cost is high, only use if threat is great.
- ** If the source of failures is known, use specialized solutions instead.

Impossibility result for the 3 byzantine generals problem, 1 faulty

- ** There are not f-resilient algorithm if the number of processes are N<=3

- - ** Shows that no consensus algorithm for N<=3f exists.

- ** If the commanding process is correct, but one of the receives are faulty, the other receiver would receive conflicting information

  - P1 says V, P2 says P1 said W, P3 received conflict.

- If the commanding process, the one broadcasting the order is the traitor

  - P2 and P3 received different information

- ** A consensus algorithm cannot distinguish between the above scenarios, meaning that both Integrity and Agreement cannot be achieved

Solution for 4 Byzantine generals problem, 1 faulty.

- ** For 4 processes the result is different:
- ** If the issuer, P1 is correct, but P4 is false.:
  - The P2 and P3 will both say that P1 said the same thing, meaning they can reach consensus
- ** If the issuer, P1 is a traitor/ faulty:
  - P2, P3, P4 will receive different values, and will be able to decide on a default value instead

Assumptions about the system model, failures, and message signing.

Collection of processes communicating via message passing. Consensus should be reached despite faults. We consider both Byzantine failures and crash failures. If processes sign their messages, then a faulty process is limited in the harm it can do. I.e. where signing is used, a process cannot make a false claim about the message a correct process sent to it.

Impossibility in Asynchronous Systems

Since we cannot distinguish a failed system from a slow one, we can't make the same assumptions as we can in a synchronous system.

- No algorithm can exist that is *guaranteed* to reach consensus in asynchronous systems.
  - Even if we allow 1 crash without byzantine failures or communication failures
    - Failed processes cannot be distinguished from slow processes.
    - There is always a program continuation that avoids consensus
  - Consensus can be reached, but not guaranteed.

Practical Workarounds

- **Fault Masking**: By using persistent storage and restarting failed programs, all failures are fixed and simply treated like slow programs. This way, we circumvent the impossibility result.
- **Failure Detectors**: Processes deem a suspect process as failed and ignore its future messages.
- **Randomization**: Introduce randomization, so processes become slightly unpredictable. This reduces the power of adversaries to predictably manipulate the system.

- Nearly synchronous systems:
  - Uses Paxos Algorithm '98, completely-safe and largely-live consensus protocol for asynchronous systems.

  - Used "part-time parliament" analogy but took 9 years before it was published, updated in "Paxos made simple" paper.
  - "Asynchronous consensus algorithms like Paxos maintain safety despite asynchrony, but are guaranteed to make progress only when the system becomes synchronous - meaning that messages are delivered in a bounded length of time.“

Paxos

- Paxos is named after the Parliament on the fictitious Greek island of Paxos.
- Paxos is a family of algorithms (by Leslie Lamport) for distributed consensus in an asynchronous system
- In Paxos termination / liveness is not guaranteed, but happens in “reasonable environments”
- Different roles exist:
  - Proposer: Offers proposals, with multiple proposers at once, they instead compete to reach approval first.
  - Acceptors: Accepts or rejects proposals
  - Learners: Simply learns the agreed upon proposals
- Proposals must have majority to be accepted.
- Paxos Consensus works by sending a prepare request to some acceptors (other participants of the blockchain)
  - The acceptors accept the proposal
  - The proposer sends a commit request
  - The acceptors accept the commit.
- Paxos only requires a majority to accept, meaning half the participants of the blockchain can never answer, and Paxos will still work.