UNIT - 3

Agreement Protocols →

- In distributed systems, where sites (or processors) often compete as well as cooperate to achieve a common goal, it is often required that sites reach mutual agreement.
- Reaching an agreement typically requires that sites have knowledge about the values of other sites.
- For eg. in distributed commit, a site should know the outcome of local commit at each site.
- When the system is free from failures, an agreement can easily be reached among the processors (or sites).
- For eg., processors can reach an agreement by communicating their values to each other and then by taking a majority vote or a minimum, maximum, mean, etc of those values.
- This method ~~can~~ does not work with faulty processors because they can send conflicting values to other processors preventing them from reaching an agreement.

The System Model

Agreement problems have been studied under the following system model:

- There are n processors in the system and at most m of the processors can be faulty.
- The processors can directly communicate with other processors by message passing.
- A receiver processor always knows the identity of the sender processor of the message.
- This communication medium is reliable and only processors are prone to failure.

Classification of Agreement Problems

There are three agreement problems in distributed system -

(i) Byzantine agreement problem
(ii) Consensus problem
(iii) Interactive consistency problem

In all three problems, all nonfaulty processors must reach a common agreement.

The Byzantine Agreement Problem →

- In the Byzantine agreement problem, an arbitrary chosen processor, called the source processor, broadcasts its initial value to all other processors.
- A solution to the Byzantine agreement problem should meet the following two objectives :-

Agreement - All nonfaulty processors agree on the same value.

Validity - If the source processor is nonfaulty, then the common agreed upon value by all nonfaulty processors should be the initial value of the source.

Two points are very important —

(a) If the source processor is faulty, then all non faulty processors can agree on any common value.

(b) It is irrelevant what value faulty processors agree on or whether they agree on a value at all.

(ii) The Consensus problem :→

- Every processor broadcasts its initial value to all other processors.
- Initial values of the processors may be different.

- A protocol for reaching consensus should meet the following conditions.

Agreement - All nonfaulty processors agree on the same single value.

validity - If the initial value of every nonfaulty processor is v, then the agreed upon common value by all nonfaulty processors must be v.

Important point is that if the initial values of nonfaulty processors are different, then all nonfaulty processors can agree on any common value.

- Again don't care what value faulty processors agree on.

(iii) The Interactive Consistency Problem →

- Every processor broadcasts its initial value to all other processors.
- The initial values of the processors may be diff. erent.
- A protocol for the interactive consistency problem should meet the following conditions :-

Agreement - All nonfaulty processors agree on the same vector, $(v_1, v_2, \ldots v_n)$.

Validity - If the ith processor is nonfaulty and its initial value is v_i, then the ith value to be agreed on by all nonfaulty processors must be v_i.

→ Important point is that if the jth processor is faulty, then all nonfaulty processors can agree on any common value for v_j.

- It is irrelevant what value faulty processors agree on.

Problem	Byzantine Agreement	Consensus	Interactive Consistency
Who initiates the value	one processor	All processors	All processors
Final Agreement	Single value	single value	A vector of values

Relations among the agreement problems -

- All three agreement problems are closely related.
- Byzantine agreement problem is a special case of the interactive consistency problem, in which the initial value of only one processor is of interest.
- conversely, if each of the n processors runs a copy of the Byzantine agreement protocol, the interactive.

consistency problem is solved.

- Likewise, the consensus problem can be solved using the solution of the interactive consistency problem. because all nonfaulty processors can compute the value that is to be agreed upon by taking the majority value of the common vector that is computed by an interactive consistency protocol, or by choosing a default value if a majority does not exist.

- Thus solutions to the interactive consistency and consensus problems can be derived from the solutions to the Byzantine agreement problem.

Aspects of Recognizing The Agreement Protocol :→

(1) Synchronous Vs Asynchronous Computations :-

- In a synchronous computation, processes in the system run in lock step manner where in each step, a process receives messages. performs a computation and sends message to other processes.

- In asynchronous computation, the computation at processes does not proceed in lock steps.
  - A process can send and receive messages and perform computation at any time.

- In fact the agreement problem is not solvable in an asynchronous system, even for a single processor failure.

(ii) Model of Processor Failures :-

- A processor can fail in three models :
  - Crash fault
  - Omission fault
  - Malicious Fault

In crash fault, a processor stops functioning and never resumes operation.

In omission fault, a processor 'omits' to send messages to some processors.

In malicious fault, processor behaves randomly and arbitrarily.

- E.g. a processor may send fictitious messages to other processors to confuse them.
- Malicious faults are also referred to as 'Byzantine faults'.

(iii) Authenticated Vs Non-authenticated messages :-

There are two types of messages : authenticated and non-authenticated.

- In an authenticated message system, a (faulty) processor can not forge a message or change the contents of a received message.
- A authenticated message is also called a signed message.

- In a non authenticated message system, a faulty processor can forge a message and claim to have received it from another processor or change the contents of a received message before it relays the message to other processors.
- A non-authenticated message is also called an oral message.

(iv) Performance Aspects :→

- The performance of agreement protocols is generally determined by the following three metrics - Time, message Traffic and storage overhead

- Time refers to the time taken to reach an agreement under a protocol.
- message Traffic is measured by the number of messages exchanged to reach an agreement.
- storage overhead measures the amount of information that needs to be stored at processors during the execution of a protocol.

Solutions to the Byzantine Agreement Problem

- The Byzantine agreement problem was first defined and solved by Lamport.
- It is also called as the Byzantine generals problem because the problem resembles a situation where a team of generals in an army is trying to reach agreement on an attack plan.
- The generals are located at geographically distant positions and communicate only through messengers.
- It is obvious that all the processors must exchange the values through messages to reach a consensus.
- Processors send their values to other processors and relay received values to other processors.
- During the execution of the protocol, faulty processors may confuse other processors by sending them conflicting values or by relaying to them fictitious values.

The solution of such problem is based on the presence of faulty and nonfaulty processors. some of them are —

1. The upper bound on the number of faulty processors
2. An Impossibility Result
3. Lamport - Shostak - Pease Algorithms
4. Dolev et al's Algorithm

1. The upper bound on the number of faulty Processors

- In order to reach an agreement on a common value, nonfaulty processors need to be free from the influence of faulty processors.
- If faulty processors dominate in number, they can prevent nonfaulty processors from reaching a consensus.
- Thus, the number of faulty processors should not exceed a certain limit if a consensus is to be reached.
- Pease et al showed that in fully connected network, it is impossible to reach a consensus if the number of faulty processors, m, exceeds $\lfloor (n-1)/3 \rfloor$.
- Lamport et al. were the first to give a protocol to reach Byzantine agreement that requires $m+1$ rounds of message exchanges.

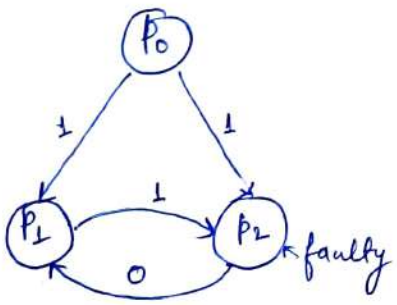
  m = max. number of faulty processors
- Fischer et al. showed that $m+1$ is the lower bound on the number of rounds of message exchanges to reach a Byzantine agreement in a fully connected network where only processors can fail.

An Impossibility Result →

- It may show that a Byzantine agreement cannot be reached among three processors, where one processor is faulty.
- Consider a system with three processors, p_0, p_1 and p_2.
- For simplicity, we assume that there are only two values, 0 and 1, on which processors agree.
- And processor p_0 initiates the initial value.
- There are two possibilities:
  - p_0 is not faulty
  - p_0 is faulty.

p_0 is not faulty → (case I) Assume p_2 is faulty. Suppose p_0 broadcasts an initial value of 1 to both p_1 and p_2.

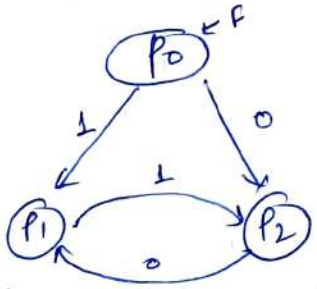
- Processor p_2 acts maliciously and communicates a value of 0 to processor p_1.
- Thus p_1 receives conflicting values from p_0 and p_2.
- However, since p_0 is nonfaulty, processor p_1 must accept 1 as the agreed upon value if all nonfaulty processors agree on the same value.



processor p_0 is nonfaulty

p_0 is faulty → (case II)

- Suppose that processor p_0 sends an initial val of 1 to p_1 and 0 to p_2.
- Processor p_2 will communicate the value 0 to p_1.
- Any agreement protocol which works for three processors can not distinguish between the two cases and must force p_1 to accept 1 as the agreed upon value whenever p_1 is faced with such situations (All nonfaulty processors agree on the same value) (Condition 1)
- In case II, this will work only if p_2 is also made to accept 1 as the agreed upon value.



processor p_0 is faulty

- Using a similar agrument we can shows that if p_2 receives an initial value of 0 from p_0, then it must take 0 as the agreed upon value, even if p_1 communicates a value of 1.
- However, if this is followed in case II, p_1 will agree on a value of 1 and p_2 will agree on a value of 0, which will violate condition 1 of the sol.
- Therefore, no solution exists for the Byzantine agreement problem for three processors.

Lamport - Shostak - Pease Algorithm →

- Lamport et al.'s algorithm referred to as the Oral message algorithm $OM(m)$, $m > 0$, solves the Byzantine agreement problem for $3m+1$ or more processors in the presence of at most m faulty processors.
- Let n denotes the total no. of processors.

Algorithm OM(0) -

(a) The source processor sends its value to every processor.

(b) Each processor uses the value it receives from the source. (If it receives no value, then it uses a default value of 0.)

Algorithm OM(m), $m > 0$

(a) The source processor sends its value to every processor.

(b) For each i, let v_i be the value processor i receives from the source. (Otherwise uses a default value of 0).

- Processor i acts as the new source and initiates Algorithm $OM(m-1)$ wherein it sends the value v_i to each of the $n-2$ other processors.

(c) For each i and each $j (\neq i)$, let v_j be the

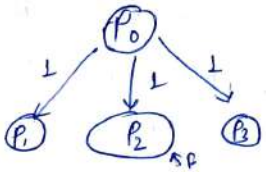
value process i received from processor j in step 3 of algo.

(b) using Algorithm OM(m-1). (If it receiv no value, then it uses a default value

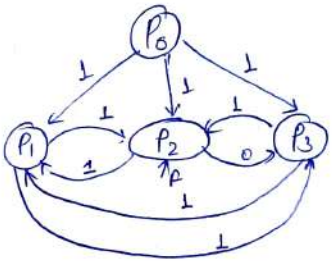
- Processor i uses the value majority $(v_1, v_2 \ldots v_n)$.

Ex. consider a system with four processors, p, p_1, p_2, p_3
- for simplicity, there are only two values 0 and 1.
- Processor p_0 initiates the initial value and p_2 is faulty.

Sol: To initiate the agreement, processor p_0 executes algo. OM(1) wherein it sends its value 1 to all processors.



- At step 2, after having received the value 1 from the source processor p_0, processors p_1, p_2 and p_3 execute the algorithm OM(0).



- processors p_1 and p_3 are nonfaulty and sends value 1 to processors $\{p_2, p_3\}$ and $\{p_1, p_2\}$.
- Faulty processor p_2 sends value 1 to p_1 and a value 0 to p_3.

After having received all the messages, processors p_1, p_2 and p_3 execute step (c) of the algo. OM(1) to decide on the majority value.

- Processor p_1 has received values (1, 1, 1), whose majority value is 1, processor p_2 has received values (1, 1, 1) whose majority value is 1, and processor p_3 has received values (1, 1, 0) whose majority value is 1.
- Hence both conditions of the Byzantine agreement are satisfied.

4. Dolev et al.'s Algorithm →

- Dolev et al. have given a polynomial algorithm for reaching Byzantine agreement.
- The algorithm requires up to $2m+3$ rounds to reach an agreement.
- Thus, there is a trade off between message complexity and time delay (rounds).

Applications of Agreement Algorithms

There are two such applications -

1. Fault-Tolerant clock Synchronization
2. Atomic commit in DDBS

1. Fault-Tolerant clock Synchronization →

- It is often necessary that sites (or processes) maintain physical clocks that are synchronized with one another.
- Since physical clocks have a drift problem, they must be periodically resynchronized.

such periodic synchronization becomes extremely difficult if the Byzantine failure are allowed.

- Because a faulty process can report different clock values to different processes.

According to Lamport and Melliar-Smith, we make the following assumptions regarding the system:

A1: All clocks are initially synchronized to approximately the same values.

A2: A nonfaulty process's clock runs at approximately the correct rate.

A3: A nonfaulty process can read the clock value of another nonfaulty process with at most a small error ϵ.

A clock synchronization algorithm should satisfy the following two conditions:

- At any time, the values of the clocks of all nonfaulty processes must be approximately equal.
- There is a small bound on the amount by which the clock of a nonfaulty process is changed during each resynchronization.

There are two clock synchronization Algorithm :-

(i) The interactive Convergence Algorithm –

- Algo. assumes that the clocks are initially synchronized and they are resynchronized often enough so that two nonfaulty clocks never differ by more than δ.

The Algo. –

- Each process reads the value of all other processes' clocks and sets its clock value to the average of these values.
- If a clock value differs from its own clock value by more than δ, it replaces that value by its own clock clock value when taking the average.
- It does not safeguard against the problem of two faced clocks wherein a faulty clock reports different values to different processes.
- The algo. brings the clocks of nonfaulty processes closer.
- Let two processes p and q, respectively, use C_{pr} and C_{qr} as the clock values of a third process r when computing their averages.

(ii) The interactive Consistency Algorithm →

The interactive consistency algorithm adds two improvements:

(i) It takes the median of the clock values rather than the mean. (provides a good estimate of the clock value, as the number of bad clocks will be low)

(ii) It avoids the problem of two faced clocks (which reports different values to different processes) by using a more sophisticated technique to obtain clock values of the processes.

Two processes will compute approximately the same median if they obtain approximately the same set of clock values for other processes. Therefore, the following conditions apply:

C1: Any two processes obtain approximately the same value for a process p's clock (even if p is faulty).

C2: If q is a nonfaulty process, then every nonfaulty process obtains approximately the correct value for process q's clock.

Thus if a majority of the processes are nonfaulty, the median of all the clock values is either approximately equal to a good clock's value or it lies between the values of two good clocks.

Atomic Commit in DDBS →

- In the problem of atomic commit, sites of a DDBS must agree whether to commit or abort a transaction.
- In the first phase of the atomic commit, sites execute their part of a distributed transaction and broadcast their decisions (commit or abort) to all other sites.
- In second phase, each site, based on what it received from other sites in the first phase, decides whether to commit or abort its part of the distributed transaction.
- Since every site receives an identical response from all other sites, they will reach the same decision.
- If some sites behave maliciously, they can send a conflicting response to other sites, causing them to make conflicting decisions.
- In these situations, we can use algorithm for the Byzantine agreement.

- In the first phase, after a site has made a decision it starts the Byzantine agreement.
- In the second phase, processors determine a common decision based on the agreed vector of values.