Ch. 9 Agreement Protocols

9.1 Introduction

- 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.
  - Ex) In distributed database systems, data managers at sites must agree on whether to commit or to abort a transaction.
- The formal setting for a distributed agreement protocol is the following: There are M processors $P=p_1,\ldots,p_M$ that are trying to reach agreement. A subset F of the processors are faulty, and remaining processors are nonfaulty. Each processor $p_i \in P$ stores a value V_i. During the agreement protocol, the processors calculate an agreement value A_i. After the protocol ends, the following two conditions should hold:

  ① For every pair p_i and p_j of nonfaulty processors, $A_i = A_j$. This value is the agreement value.

  ② The agreement value is a function of the initial values $\{V_i\}$ of the nonfaulty processors (P - F).

9.2 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.
- The communication medium is reliable (i.e., it delivers all messages without introducing any errors) and only processors are prone to failure.

9.2.1 Synchronous vs. Asynchronous Computations

- Synchronous computation
  - A process receives messages (1 round), performs a computation (2 round), and send messages to other processes (3 round).
- 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.

9.2.2 Model of Processor Failures

- A processor can fail in three modes:
  - Crash fault: a processor stops functioning and never resumes operation.
  - Omission fault: a processor "omits" to send messages to some processors.
  - Malicious fault (Byzantine faults): a processor behaves randomly and arbitrarily.

9.2.3 Authenticated vs. Non-Authenticated Messages

- Authenticated message system
  - A (faulty) processor cannot forge a message or change the contents of a received message.
  - A processor can verify the authenticity of a received message.
  - An authenticated message is also called a signed message.
- 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 processor have no way of verifying the authenticity of a received message.
  - A non-authenticated message is also called an oral message.

9.2.4 Performance Aspects

■ Performance of agreement protocols
▪ time: the number of rounds
▪ message traffic: the number of messages exchanged to reach an agreement
▪ storage overhead: the amount of information that needs to be stored at processors during the execution of a protocol.

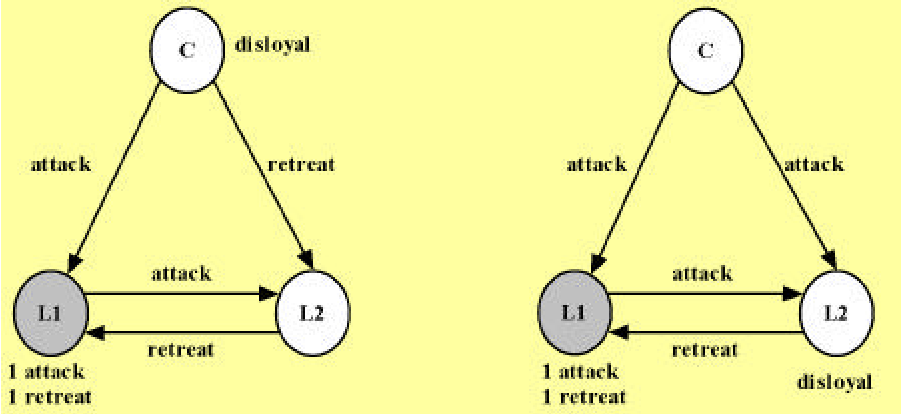
9.3 A Classification of Agreement Protocols

■ Three well known problems
① Byzantine agreement problem
② Consensus problem

Problem	Byzantine	Consensus	Interactive Consistency
Who initiates the value	One processor	All processors	All processors
Final agreement	Single value	Single value	A vector of values

③ Interactive consistency problem
■ Byzantine agreement protocol
▪ A single value, which is to be agreed on, is initialized by an arbitrary processor and all nonfaulty processors have to agree on that value.
■ Consensus problem
▪ Every processor has its own initial value and all nonfaulty processor must agree on a single common value.
■ Interactive consistency problem
▪ Every processor has its own initial value and all nonfaulty processors must agree on a set of common values.
■ The three agreement problems

9.3.1 The Byzantine Agreement Problem
▪ Three generals can not reach Byzantine agreement



- An arbitrarily 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 initial value of the source.
- Two points should be noted:

① If the source processor is faulty, then all nonfaulty processors can agree on any common value.

② It is irrelevant what value faulty processors agree on or whether they agree on a value at all.

9.3.2 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 intial value of every nonfaulty processor is v, then agreed upon common value by all nonfaulty processor must be v.
- Note that if the initial values of nonfaulty processors are different, then all nonfaulty processors can agree on any common value.

9.3.3 The Interactive Consistency Problem

- Every processor broadcasts its initial value to all other processors. The initial values of the processors may be different.
- 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, ..., v_n).
  - Validity: If the *i*th processor is nonfaulty and its initial value is v_i, then the *i*th value to be agreed on by all nonfaulty processors must be v_i.
- Note that if the *j*th processor is faulty, then all nonfaulty processors can agree on any common value for v_j.

9.3.4 Relations among the Agreement Problems

- The Byzantine agreement problem is primitive to the other two agreement problems.

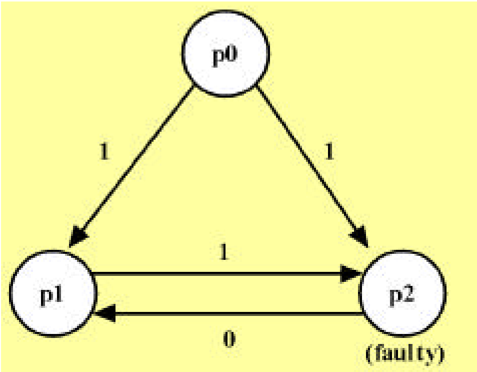
9.4 Solutions to the Byzantine Agreement Problem

- The Byzantine agreement problem is also referred to as the Byzantine *generals* problem.

9.4.1 An Impossibility Result

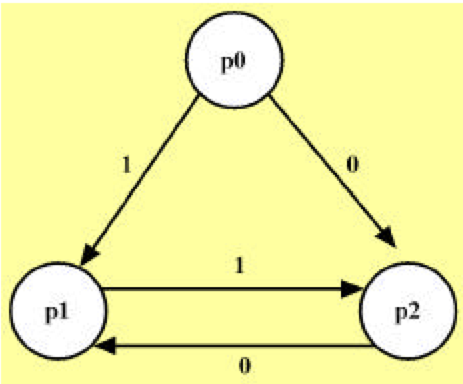
- We now show that a Byzantine agreement cannot be reached among three processors, where one processor is faulty.
  - Consider a system with three processors, p_0, p_1, and p_2. For simplicity, we assume that there are only two values, 0 and 1, on which processors agree and processor p_0 initiates the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

    Since p_0 is nonfaulty, processor p_1 must accept 1 as the agreed upon value if condition 2 is to be satisfied.



- Case Ⅱ: p_0 is faulty

  p_0 will agree on a value of 1 and p_2 will agree on a value of 0, which will violate condition 1 of the solution.



9.4.2 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 denote the total number of processors (clearly, n≥3m+1).
- Algorithm OM(0)

  1. The source processor sends its value to every processor.
  2. 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.

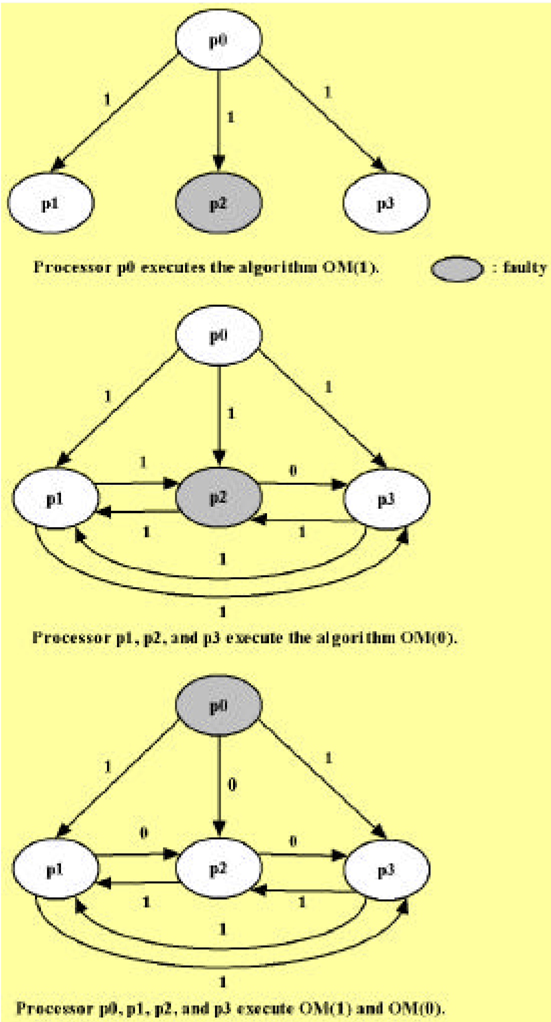
  1. The source processor sends its value to every processor.
  2. For each *i*, let v_i be the value processor *i* receives from the source. (If it receives no value, then it uses a default value of 0.). Processor *i* acts as the new source and initiates Algorithm OM(m-1) wherein it

sends the value v_i to each of the n-2 other processors.

3. For each i and each j ($\neq i$), let v_j be the value processor i received from processor j in Step 2. using Algorithm OM(m-1). Processor i uses the value majority(v_1, v_2, ..., v_{n-1}).

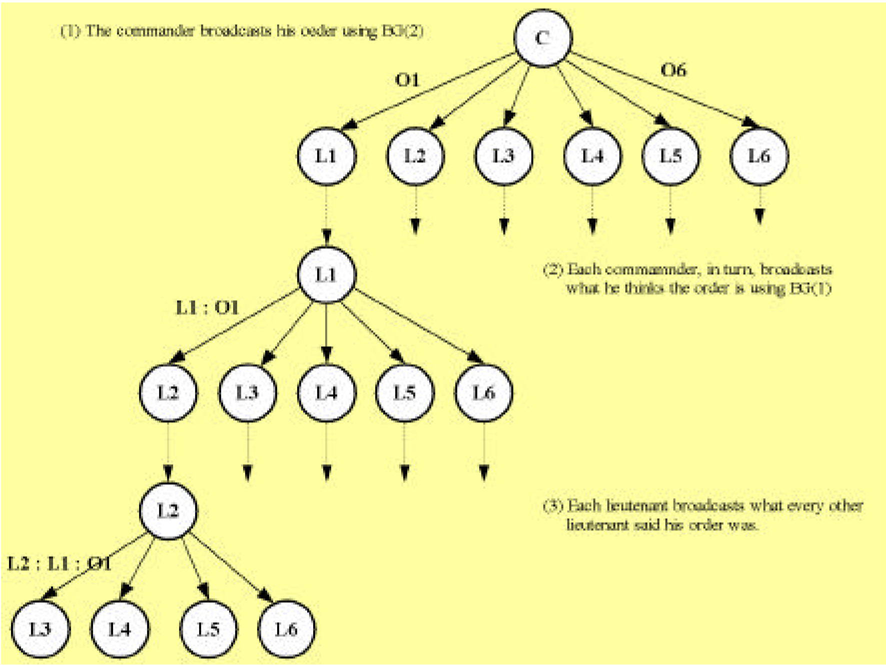
- The message complexity of the algorithm is $O(n^m)$.

■ Example 1



■ An execution of BG(2) on seven generals.

O_i represents the command sent to L_i, and L_i : O_i is L_i's rebroadcast of its command. L_j : L_i : O_i is L_j's rebroadcast of what L_i said his order was.



9.4.3 Dolev et al.'s Algorithm

- The algorithm requires up to $2m+3$ rounds to reach an agreement.
- Data Structure
  - The algorithm uses two thresholds: LOW and HIGH, where LOW:=m+1 and HIGH:=2m+1.
  - The basic idea is that any subset of processors of size LOW will have at least one nonfaulty processor.
  - Any subset of processors of size HIGH includes a majority of processors, that is, $m+1$, that are nonfaulty.
  - The algorithm uses two types of messages: a "*" message and a message consisting of the name of a processor.
    - The "*" denotes the fact that the sender of the message is sending a value of 1 and the name in a message denotes the fact that the sender of the message received a "*" from the named processor.
  - W_x^i : the set of processors that have sent message x to processor i. (Note that x is either a "*" or a processor name.)
    - Each process maintains n+1 numbers of W sets.
    - W_x^i : the set of *witnesses* of message x for processor *i*.
    - A processor j is a *direct supporter* for a processor k if j directly receives "*" from k.

- When processor i receives the message "k" from processor j, it adds j into W_x^i because j is a witness to message "k". Process j is an *indirect supporter* for processor k if $|W_k^j| \geq LOW$;
- A processor j confirms processor k if $|W_k^j| \geq HIGH$;

▪ A process *i* maintains a set, C_i, of confirmed processors.

■ The Algorithm

▪ First round: the source processor sends a "*" message to all processors (including itself) if its value is 1. If its value is 0, it send nothing in the first round. If the processors finally agree on "*", then the agreed upon value is 1. Otherwise, the agreed upon value is 0.

▪ Subsequent rounds: a processor sends its message to all processors, receives messages from other processors, and then decides what messages to send in the next round.

▪ *initiation* operation

- It initiates in the second round if it receives a "*" from the source in round 1.
- It initiates in the K+1st round if at the end of Kth round the cardinality of the set of the confirmed processors (not including the source) at least $LOW + \max(0, \lfloor \frac{K}{2} \rfloor - 2)$ (referred to as the *condition of initiation*).

▪ Four rules

① In the first round, the source broadcasts its value to all other processors.

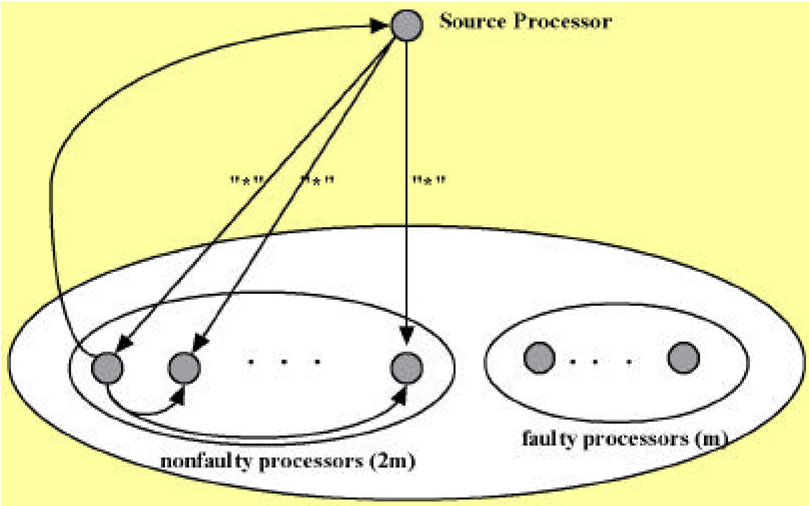
② In a round k>1, a processor broadcasts the names of all processes for which it is either a direct or indirect supporter and which it has not previously broadcast. If the condition of initiation was true at the end of the previous round, it also broadcasts the "*" message unless it has previously done so.

③ If a processor confirms HIGH number of processors, it commits to a value of 1.

④ After round 2m+3, if the value 1 is committed, the processors agree on 1; otherwise, they agree on 0.

■ Example

- processors, 3m+1
  faulty processors, m
  source is nonfaulty.



- The source processor broadcasts a "*" in the first round. In the second round, 2m nonfaulty processors will initiate (i.e., broadcast "*"). In the third round, 2m+1 nonfaulty processors (including the source) will broadcast messages containing the name of the processors informing that they have witnessed a "*" from 2m other nonfaulty processors. Thus, in the fourth round, the witness set of all 2m+1 nonfaulty processors will contain all 2m+1 nonfaulty processors and they all will commit to a value of 1 in the fourth round.

9.5 Applications of Agreement Algorithms

- Fault-Tolerant Clock Synchronization
- Atomic Commit in DDBS