CHAPTER 8

AGREEMENT PROTOCOLS

8.1 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. For example, in distributed database systems, data managers at sites must agree on whether to commit or to abort a transaction [11]. Reaching an agreement typically requires that sites have knowledge about the values of other sites. For example, 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 example, 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. However, when the system is prone to failure, this method does not work. This is because faulty processors can send conflicting values to other processors preventing them from reaching an agreement. In the presence of faults, processors must exchange their values with other processors and relay the values received from other processors several times to isolate the effects of faulty processors. A processor refines its value as it learns of the values of other processors (This entire process of reaching an agreement is called an *agreement protocol*).

In this chapter, we study agreement protocols for distributed systems under processor failures. A very general model of faults is assumed. For example, a faulty processor may send spurious messages to other processors, may lie, may not respond to received

messages correctly, etc. Also, nonfaulty processors do not know which processors are faulty.

In agreement problems, nonfaulty processors in a distributed system should be able to reach a common agreement, even if certain components in the system are faulty. The agreement is achieved through an agreement protocol that involves several rounds of message exchange among the processors.

8.2 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. Thus, the system is logically fully connected.
- 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 failures.

For simplicity, we assume that agreement is to be reached between only two values, 0 and 1. Results can easily be extended to multivalue agreement [23].

Early solutions to agreement problems assumed that only processors could be faulty and that communication links did not fail. Limiting faults solely to the processors simplifies the solution to agreement problems. Recently, agreement problems have been studied under the failure of communication links only [24] and under the failure of both processors and communication links [25] In this chapter, we limit the treatment of agreement problems solely to processor failures.

8.2.1 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 (sent to it in the previous step), performs a computation, and sends messages to other processes (received in the next step). A step of a synchronous computation is also referred to as a *round*. In synchronous computation, a process knows all the messages it expects to receive in a round. A message delay or a slow process can slow down the entire system or computation.

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

In this chapter, synchronous models of computation are assumed. The assumption of synchronous computation is critical to agreement protocols. In fact, the agreement problem is not solvable in an asynchronous system, even for a single processor failure [10].

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.

8.3.2 The Consensus Problem

In 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.

Note that if the initial values of nonfaulty processors are different, then all nonfaulty processors can agree on *any* common value. Again, we don't care what value faulty processors agree on.

8.3.3 The Interactive Consistency Problem

In 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 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.

Note 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.

8.3.4 Relations Among the Agreement Problems

All three agreement problems are closely related [7]. For example, the 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. This is 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. In other words, the Byzantine agreement problem is primitive to the other two agreement problems. For this reason, we will focus solely on the Byzantine agreement problem for the rest of the chapter.

However, it should by no means be concluded that the Byzantine agreement problem is weaker than the interactive consistency problem or that the interactive consistency problem is weaker than the consensus problem. In fact, there is no linear ordering of this sort among these agreement problems. For example, the Byzantine agreement problem can be solved using a solution to the consensus problem in the following manner [7]:

1. The source processor sends its value to all other processors, including itself.
2. All the processors, including the source, run an algorithm for the consensus problem using the values received in the first step as their initial values.

The above two steps solve the Byzantine agreement problem because (1) if the source processor is nonfaulty, then all the processors will receive the same value in the first step and all nonfaulty processors will agree on that value as a result of the consensus algorithm in the second step, and (2) if the source processor is faulty, then the other processors may not receive the same value in the first step; However, all nonfaulty processors will agree on the same value in the second step as a result of the consensus algorithm. Thus, the Byzantine agreement is reached in both cases. However, the $n - 1$ extra messages are sent at the first step.

8.4 SOLUTIONS TO THE BYZANTINE AGREEMENT PROBLEM

The Byzantine agreement problem was first defined and solved (under processor failures) by Lamport et al. [14, 16]. Recall that in this problem, an arbitrarily chosen processor (called the source processor) broadcasts its initial value to all other processors. A protocol for the Byzantine agreement should guarantee that all nonfaulty processors agree on the same value and if the source processor is nonfaulty, then the common agreed upon value by all nonfaulty processors should be the initial value of the source.

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 [16]. 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 Byzantine agreement problem is also referred to as the Byzantine *generals* problem ([4, 14]) 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. Some of the generals are traitors (equivalent to faulty processers) and try to prevent loyal generals from reaching an agreement by deliberately transmitting erroneous information.

8.4.1 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. [16] showed that in a 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. [14] were the first to give a protocol to reach Byzantine agreement that requires $m+1$ rounds of message exchanges (m is the maximum number of faulty processors). Fischer et al. [8] 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.

However, if authenticated messages are used, this bound is relaxed and a consensus can be reached for any number of faulty processors.

8.4.2 An Impossibility Result

We now show that a Byzantine agreement cannot be reached among three processors, where one processor is faulty [14]. For a rigorous treatment of this impossibility result for a higher number of processors, readers are referred to [9] and [16].

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: (1), p_0 is not faulty or (2) p_0 is faulty.

Case I: p_0 is not faulty. Assume p_2 is faulty. Suppose that 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. (This scenario is shown in Fig. 8.1. A faulty processor is depicted by an oval and a nonfaulty processor is denoted by a circle.) However, since p_0 is nonfaulty, processor p_1 must accept 1 as the agreed upon value if condition 2 (of Sec. 8.3.1) is to be satisfied.

Case II: p_0 is faulty. Suppose that processor p_0 sends an initial value of 1 to p_1 and 0 to p_2. Processor p_2 will communicate the value 0 to p_1. (This scenario is shown in Fig. 8.2). As far as p_1 is concerned, this case will look identical to Case I. So any agreement protocol which works for three processors cannot 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 (to satisfy condition 2). However, in Case II, this will work only if p_2 is also made to accept 1 as the agreed upon value.

Using a similar argument, we can show 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

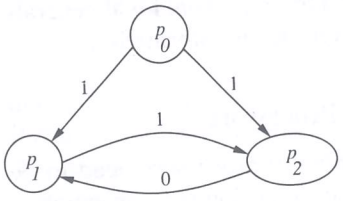

FIGURE 8.1
Processor p_0 is Non-Faulty.

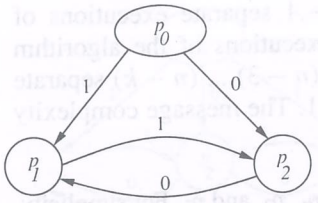

FIGURE 8.2
Processor p_0 is Faulty.

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 (Sec. 8.3.1) of the solution.

Therefore, no solution exists for the Byzantine agreement problem for three processors, which can work under single processor failure.

8.4.3 Lamport-Shostak-Pease Algorithm

Lamport et al.'s algorithm [14], 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, $n \geq 3m + 1$). The algorithm is recursively defined as follows:

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($m-1$). (If it receives no value, then it uses a default value of 0.). Processor i uses the value *majority*(v_1, v_2, ..., v_{n-1}).

This algorithm is evidently quite complex. The processors are successively divided into smaller and smaller groups and the Byzantine agreement is recursively achieved within each group of processors (Step 2 of "Algorithm OM($m-1$)"). Step 3 is executed during the folding phase of the recursion, where a *majority* function is applied to select the majority value out of the values received in a round of message exchange (Step 2). The function *majority* (v_1, v_2, ..., v_{n-1}) computes a majority value of the values v_1, v_2, ..., v_{n-1} if it exists (otherwise, it returns the default value 0).

The execution of the algorithm OM(m) invokes $n-1$ separate executions of the algorithm OM($m-1$), each of which invokes $n-2$ executions of the algorithm OM($m-2$), and so on. Therefore, there are $(n-1)(n-2)(n-3) \ldots (n-k)$ separate executions of the algorithm OM($m-k$), k = 1, 2, 3, ..., $m+1$. The message complexity of the algorithm is O(n^m).

Example 8.1. Consider a system with four processors, p, p_1, p_2, and p_3. For simplicity, we assume that there are only two values 0 and 1; furthermore, we assure that processor p_0 initiates the initial value and that processor p_2 is faulty.

To initiate the agreement, processor p_0 executes algorithm OM(1) wherein it sends its value 1 to all other processors (Fig. 8.3). At Step 2 of the algorithm OM(1), 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). These executions are shown in Fig. 8.4. Processors p_1 and p_3 are nonfaulty and send value 1 to processors $\{p_2, p_3\}$ and $\{p_1, p_2\}$, respectively. Faulty processor p_2 sends value 1 to p_1 and a value 0 to p_3 (see Fig. 8.4).

After having received all the messages, processors p_1, p_2, and p_3 execute Step 3 of the algorithm 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. Thus, both conditions of the Byzantine agreement are satisfied.

Example 8.2. Figure 8.5 shows a situation where processor p_0 is faulty and sends conflicting values to the other three processors. These three processors, under Step 1 of OM(0), send the received values to the other two processors.

After having received all the messages, processors p_1, p_2, and p_3 execute Step 3 of the algorithm OM(1) to decide on the majority value. Note that all three processors have received values (1, 0, 1), whose majority value is 1, Thus, all three nonfaulty processors agree on the same value and the required conditions of the Byzantine agreement are satisfied.

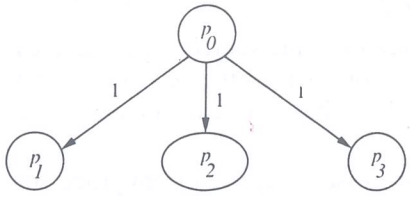

FIGURE 8.3
Processor p_0 executes the algorithm OM(1)

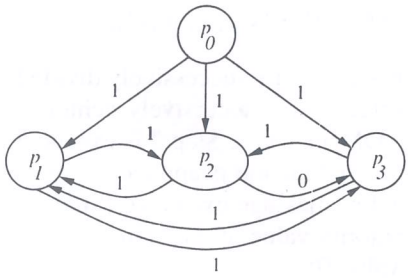

FIGURE 8.4
Processors p_1, p_2, and p_3 execute the algorithm OM(0)