Agreement in Faulty Systems

Organizing replicated processes into a group helps to increase fault tolerance. As we mentioned, if a client can base its decisions through a voting mechanism, we can even tolerate that k out of $2k + 1$ processes are lying about their result. The assumption we are making, however, is that processes do not team up to produce a wrong result.

In general, matters become more intricate if we demand that a process group reaches an agreement. Agreement is needed in many cases. Some examples are: electing a coordinator, deciding whether or not to commit a transaction, dividing up tasks among workers, and synchronization. When the communication and processes are all perfect, reaching such agreement is often straightforward, but when they are not, problems arise.

The general goal of distributed agreement algorithms is to have all the nonfaulty processes reach consensus on some issue, and to establish that consensus

within a finite number of steps. Different cases are possible depending on system parameters, including whether or not communication is reliable, or the crash-failure semantics for processes.

Before considering the case of faulty processes, let us look at the "easy" case of perfect processes but where communication lines can lose messages. There is a famous problem, known as the **two-army problem**, which illustrates the difficulty of getting even two perfect processes to reach agreement about 1 bit of information. The red army, with 5000 troops, is encamped in a valley. Two blue armies, each 3000 strong, are encamped on the surrounding hillsides overlooking the valley. If the two blue armies can coordinate their attacks on the red army, they will be victorious. However, if either one attacks by itself it will be slaughtered. The goal of the blue armies is to reach agreement about attacking. The catch is that they can only communicate using an unreliable channel: sending a messenger who is subject to capture by the red army.

Suppose that the commander of blue army 1, General Alexander, sends a message to the commander of blue army 2, General Bonaparte, reading: "I have a plan—let's attack at dawn tomorrow." The messenger gets through and Bonaparte sends him back with a note saying: "Splendid idea, Alex. See you at dawn tomorrow." The messenger gets back to his base safely, delivers his messages, and Alexander tells his troops to prepare for battle at dawn.

However, later that day, Alexander realizes that Bonaparte does not know if the messenger got back safely and not knowing this, may not dare to attack. Consequently, Alexander tells the messenger to go tell Bonaparte that his (Bonaparte's) message arrived and that the battle is set.

Once again the messenger gets through and delivers the acknowledgement. But now Bonaparte worries that Alexander does not know if the acknowledgement got through. He reasons that if Bonaparte thinks that the messenger was captured, he will not be sure about his (Alexander's) plans, and may not risk the attack, so he sends the messenger back again.

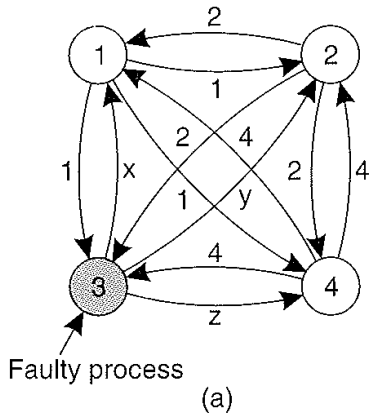
Even if the messenger makes it through every time, it is easy to show that Alexander and Bonaparte will never reach agreement, no matter how many acknowledgements they send. Assume that there is some protocol that terminates in a finite number of steps. Remove any extra steps at the end to get the minimum protocol that works. Some message is now the last one and it is essential to the agreement (because this is the minimum protocol). If this message fails to arrive, the war is off.

However, the sender of the last message does not know if the last message arrived. If it did not, the protocol did not complete and the other general will not attack. Thus the sender of the last message cannot know if the war is scheduled or not, and hence cannot safely commit his troops. Since the receiver of the last message knows the sender cannot be sure, he will not risk certain death either, and there is no agreement. Even with nonfaulty processes (generals), agreement between even two processes is not possible in the face of unreliable communication.

Now let us assume that the communication is perfect but the processes are not. The classical problem here also occurs in a military setting and is called the **Byzantine generals problem**. In this problem the red army is still encamped in the valley, but n blue generals all head armies on the nearby hills. Communication is done pairwise by telephone and is instantaneous and perfect, but m of the generals are traitors (faulty) and are actively trying to prevent the loyal generals from reaching agreement by feeding them incorrect and contradictory information (to model malfunctioning processes). The question is now whether the loyal generals can still reach agreement.

For the sake of generality, we will define agreement in a slightly different way here. Each general is assumed to know how many troops he has. The goal of the problem is for the generals to exchange troop strengths, so that at the end of the algorithm, each general has a vector of length n corresponding to all the armies. If general i is loyal, then element i is his troop strength; otherwise, it is undefined.

A recursive algorithm was devised by Lamport et al. (1982) that solves this problem under certain conditions. In Fig. 7-4 we illustrate the working of the algorithm for the case of $n = 4$ and $m = 1$. For these parameters, the algorithm operates in four steps. In step one, every general sends a (reliable) message to every other general announcing his troop strength. Loyal generals tell the truth; traitors may tell every other general a different lie. In Fig. 7-4(a) we see that general 1 reports 1K troops, general 2 reports 2K troops, general 3 lies to everyone, giving x, y, and z, respectively, and general 4 reports 4K troops. In step 2, the results of the announcements of step 1 are collected together in the form of the vectors of Fig. 7-4(b).



(a)

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

Figure 7-4. The Byzantine generals problem for three loyal generals and one traitor. (a) The generals announce their troop strengths (in units of 1 kilosoldiers). (b) The vectors that each general assembles based on (a). (c) The vectors that each general receives in step 3.

Step 3 consists of every general passing his vector from Fig. 7-4(b) to every other general. Each general gets three vectors, one from each other general. Here,

too, general 3 lies through his teeth, inventing 12 new values, *a* through *l*. The results of step 3 are shown in Fig. 7-4(c). Finally, in step 4, each general examines the *i*th element of each of the newly received vectors. If any value has a majority, that value is put into the result vector. If no value has a majority, the corresponding element of the result vector is marked *UNKNOWN*. From Fig. 7-4(c) we see that generals 1, 2, and 4 all come to agreement on

(1, 2, *UNKNOWN*, 4)

which is the correct result. The traitor cannot corrupt the information from the loyal generals; he was not able to gum up the works.

Now let us revisit this problem for $n = 3$ and $m = 1$, that is, only two loyal generals and one traitor, as illustrated in Fig. 7-5. Here we see that in Fig. 7-5(c) neither of the loyal generals sees a majority for element 1, element 2, or element 3, so all of them are marked *UNKNOWN*. The algorithm has failed to produce agreement.

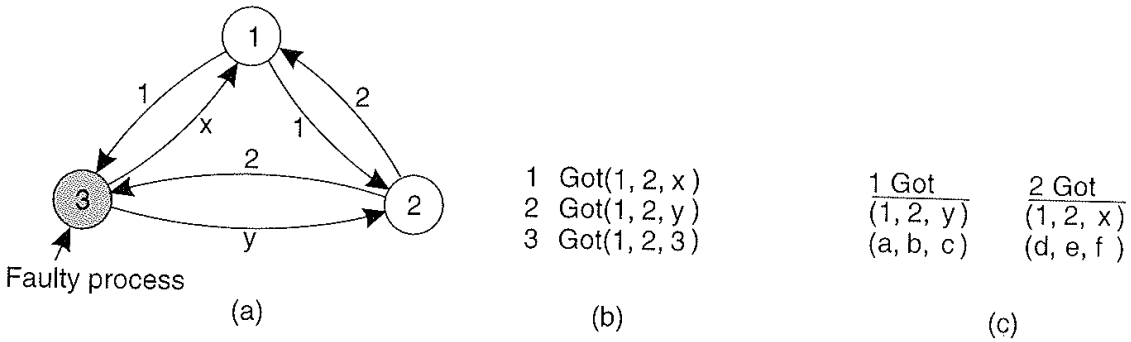


Figure 7-5. The same as Fig. 7-4, except now with two loyal generals and one traitor.

In their paper, Lamport et al. (1982) proved that in a system with *m* faulty processes, agreement can be achieved only if $2m + 1$ correctly functioning processes are present, for a total of $3m + 1$. Put in slightly different terms, agreement is possible only if *more* than two-thirds of the processes are working properly.

Another way of looking at this problem, is as follows. Basically, what we need to achieve is a majority vote among a group of loyal generals regardless of whether there are also traitors among their midsts. If there are *m* traitors, we need to ensure that their vote, along with that of any loyalists who have been mislead by the traitors, still corresponds to the majority vote of the loyalists. With $2m + 1$ loyalists, this can be achieved by requiring that agreement is reached only if more than two-thirds of the votes are the same. In other words, if more than two-thirds of the generals agree on the same decision, this decision corresponds to the same majority vote by the group of loyal generals.

Unfortunately, reaching agreement can be even worse. Fischer et al. (1985) proved that in a distributed system in which messages cannot be guaranteed to be

delivered within a known, finite time, no agreement is possible if even one process is faulty (albeit if that one process fails silently). The problem with such systems is that arbitrarily slow processes are indistinguishable from crashed ones. Many other theoretical results are known about when agreement is possible and when it is not. Surveys of these results are given in (Barborak et al., 1993; and Turek and Shasha, 1992).