UNDERSTANDING PROBABILISTIC BYZANTINE AGREEMENT

Russell Turpin and Jim Dutton

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-10 April 1987

Abstract

The current variety of probabilistic protocols for reaching Byzantine agreement can be subsumed by a simple, general algorithm. After presenting and proving this algorithm, several protocols in the literature are examined in its light.

Contents

Introduction

The problem of reaching distributed agreement in the presence of completely arbitrary faults has been recognized as one of the central problems in fault-tolerant computing. The Byzantine agreement problem, first introduced in [6], is the generally accepted abstraction of this consensus problem. A large number of algorithmic solutions to this idealized problem have been published in the past several years. In addition to these algorithmic solutions, a number of theoretical results have been published.

- Pease, Shostak, and Lamport proved in an early paper [6] that Byzantine agreement without authentication (cryptography) cannot be secured unless fewer than one third of the processes are faulty.

- Recently, Fischer, Lynch, and Paterson proved asynchronous agreement impossible for deterministic processes [4].

- Lynch and Fischer put a lower bound on the number of communication rounds required to reach deterministic Byzantine agreement [5]. The lower bound is t+1, where t is the number of faulty processes.

- Turpin and Coan provided an efficient reduction of multivalued agreement to binary agreement, allowing research attention to focus on binary agreement [11].

The second and third results above are limitations of deterministic protocols -- those protocols which are guaranteed to reach agreement within some fixed time. Recently, a number of researchers have overcome these limitations using probabilistic protocols. There are now several probabilistic protocols that can reach Byzantine agreement in an asynchronous environment with an expected time that is a constant independent of the number of processes or faulty processes.

The probabilistic protocols published to date exhibit several common features. They are usually much simpler than their deterministic cousins in their algorithmic description. They all require each process to iteratively execute the protocol through a sequence of **phases**. And they are ahistoric in the sense that processes must only carry their current proposed agreement value across phases.

It is the thesis of this paper that these similarities arise because the various protocols are all variants of a single general algorithm. This general algorithm is described below and proved correct. Despite their algorithmic simplicity, most of the probabilistic protocols leave one wondering why they work at all. The conceptual framework built below to describe the general algorithm provides a basis for the understanding of these

probabilistic protocols. Several of the published protocols, representing a cross-section of the field, are examined in the light of this framework.

The Model

The problem of Byzantine agreement concerns **n** independent processes in a completely connected communications network. The network is assumed to be reliable, so that any transmitted message is eventually delivered exactly once. Furthermore, the communications system correctly identifies the sender of any message to the recipient of the message. The model presents a synchronous environment if an upper bound can be placed on the time required to deliver a message, otherwise, it presents an asynchronous environment.

The processes engage in a distributed protocol with the goal of agreeing on some piece of information, which is assumed to be a binary bit (0 or 1). A process that follows the prescribed steps of the protocol without error is said to be **correct**; otherwise, a process is said to **fail** or be **faulty**. No constraints are placed on the content of messages sent by faulty processes.

Faulty processes can be assumed malicious and to have perfect knowledge of the state of the distributed system. However, faulty processes do not know future states of the system. (This assumption, which is unnecessary for deterministic protocols, is vital for probabilistic protocols which rely on random information not determined before certain times.) Similarly, faulty processes are not able to determine encrypted information without the required keys. The parameter **t** describes the maximum number of faulty processes, and the fault-tolerance of any specific protocol is measured as an inequality involving t and n.

Each process begins a Byzantine agreement protocol with an **initial value** and terminates having decided on a **decision value**. A Byzantine agreement protocol must satisfy two criteria:

Validity If all correct processes begin with the same initial value, then each correct process decides on that value.

Agreement Each correct process decides the same value.

Deterministic protocols always terminate and satisfy the above criteria. Probabilistic protocols show weaker properties; they may only have some probability of termination or achieving the above criteria.

The Basic Idea

The validity condition for Byzantine agreement is very weak. It restricts the agreement value only when all correct processes are initially unanimous. Why then is achieving Byzantine agreement so hard? After exchanging initial values, each correct process can attempt to determine the value held by a majority of correct processes. This attempt is always successful if the processes are initially unanimous, and each process knows if it fails in this attempt. A subtly wrong algorithm for reaching Byzantine agreement proceeds as follows. In the first round, processes exchange values and each process attempts to determine the value held by a majority of correct processes. In the second round, all processes either agree on the majority value, or on a default value if any correct process failed in its attempt to determine the majority value.

The flaw in the above algorithm is that the choice between a default value and the majority value requires each correct process to determine whether **any** correct process failed in its attempt to determine the majority value. This transmission of a global state to each process requires Byzantine agreement. Thus, the above "solution" is circular in its description and moves us no closer to reaching Byzantine agreement. Surprisingly, a slight modification to the above algorithm yields a probabilistic solution to the Byzantine agreement problem.

After exchanging values, the correct processes fall into two sets: those that successfully determined the majority value, and those that failed to do so. Suppose there were a reliable way to transmit to each process the value of a global random bit. The processes that failed to detect the majority value would accept this random bit as their proposed agreement value. This procedure has a one in two chance of creating agreement amongst both the processes that determined the majority value and those that accepted the random bit, assuming that the random bit favors neither value and is independent of the majority value. The probability of reaching agreement can be made as high as desired by iterating this procedure. The central theme of this paper is that this algorithm describes all previously published probabilistic Byzantine agreement algorithms, which differ primarily in their methods for generating random bits.

A Rigorous Description

The general algorithm is iterative and each process proceeds through sequential **phases**. The state of process i at the beginning of each phase is described by its current **proposed value, V(i)**, which at the start of the first phase contains the process's initial value. During each phase, a process's proposed value may be reset. However, if all processes begin a phase with

some common value, they will all end the phase with that value; that is, agreement is persistent. In addition, if processes begin a phase with different values, there is some probability that proposed values will be reset so that all processes end the phase with a single common value. The probability of eventually reaching agreement can be made as likely as desired by continued iterations.

During each phase, each process generates the values of two variables that provide some information about the global state of the system. These variables determine the proposed value for the next phase. The first variable is calculated through a series of communications that involve exchanging proposed values, but may entail much more. The second variable is the result of a (pseudo-) random coin toss. The different random Byzantine agreement algorithms described in the literature differ primarily in how these two variables are calculated, but all rely on the properties described below to insure algorithmic correctness.

The first variable, **F(i)**, is a boolean variable with an additional "undetermined" state (0, 1, and *). This variable is called the **forced value** because the process must accept it as the new proposed agreement value unless its value is undetermined (*). Two properties limit how the various protocols can prescribe the forced value.

(1) If all processes begin the phase in agreement, then each process's forced value is the common proposed value at the beginning of the phase.

(2) If any process generates a forced value of "1", no process may generate a forced value of "0".

Property (2) above guarantees that in each phase all correct processes fall into two sets: those that have some common binary value (0 or 1) for their forced value, and those whose forced value is undetermined. It therefore makes sense to speak of the forced value for the system during a particular phase, that being the binary value (0 or 1) generated by any process for its forced value, or undetermined (*) if the forced value for all processes is undetermined.

The second variable, **C(i)**, is a simple boolean (0 or 1) and is called the **coin toss value** because it ideally represents a global random coin toss. The property required of the coin toss value to insure correctness is a subtle one.

(3) There is a positive constant **c** such that for each phase the probability of all correct processes having a coin toss value of 1 is at least c and independent of the forced value for the system in that phase, and similarly for a coin toss value of 0.

Note that while there is no guarantee that there will be a global coin toss value during each phase, as there is for the

forced value, there is at least a probability that there is a global coin toss value. A final rule concerning the generation of new proposed values is required.

(4) Each process selects its proposed value for the next phase by choosing its forced value unless it is undetermined, in which case it chooses its coin toss value.

The above four constraints on the generation of each process's forced value, coin toss value, and new proposed value are sufficient to prove correctness of the general probabilistic Byzantine agreement algorithm. The selection rule (4) together with property (1) of the forced value secures validity and persistence of agreement. Property (2) and the selection rule (4) divides processes during each phase into two sets, either of which may be empty: those that choose a common forced value for their next proposed value, and those that choose their coin tosses. Property (3) guarantees that there is a minimum probability that these coin tosses are the same and equal to the forced value. Thus, if the processes begin a phase not in agreement, there is a minimum probability, c, that they end the phase in agreement. The probability that processes do not agree after r phases is less than:

$$(1 - c)^r$$

This probability decreases exponentially in r, and can be made as small as desired by choosing large values of r. Detection of agreement and successful termination are discussed below.

The constant c must be independent of the computation trace, but may depend on the number of processes in the system and maximum number of faulty processes. Because the probability of continued disagreement is exponentially decreasing, the expected number of rounds to reach agreement is constant for any system configuration. However, if this constant increases with the number of processes, the protocol is considered to perform poorly.

Termination

A probabilistic protocol that operates according to the general algorithm described above can be executed in two distinct fashions. First, it can run for a predetermined number of phases, in which case there is some probability that the protocol will terminate with processes not in agreement. The probability depends on the number of phases allowed, and decreases to zero exponentially as this number is increased.

A second mode of operation is to allow execution until agreement is detected. In this case, the protocol terminates with probability 1 and the expected number of phases before termination is constant for a fixed number of processes.

Two different techniques are used to detect agreement. First, a correct process detects imminent agreement if it determines that in the next phase all correct processes will use the determined forced value (not *). The criteria for detecting this must be met if processes are already in agreement. Second, a correct process detects agreement if it determines that the coin tosses for the phase are unanimous and equal to the forced value. This method is only used by the protocols that generate a global coin toss value. The termination logic must insure that processes that detect agreement continue to execute the protocol until agreement is detectable by all correct processes.

Asynchronous Communication

Probabilistic Byzantine agreement can be achieved in an asynchronous environment. This is an important advantage over deterministic agreement, which was shown by Lynch and Fischer [4] to be impossible in an asynchronous environment. The general algorithm described above progresses each process through a series of phases. The global constraints concern the states of different processes in any one phase, but do not restrict the occurrence of a phase to any particular time. That is, there is no assumption that any two processes are in the same phase at the same time. Thus the general algorithm describes both synchronous and asynchronous protocols.

In a synchronous protocol, all processes progress through the same phase simultaneously. Any communication involved in the generation of forced values or coin tosses only applies to the current phase. An asynchronous protocol must take steps to prohibit cross-phase communication. The (by now) classical method for doing so is to append the phase number to each message. A process in a particular phase discards messages it receives from processes in prior phases, immediately applies messages it receives from processes in the same phase, and saves messages it receives from processes in later phases.

Faulty processes may, of course, append any phase number to their messages, or die incommunicado. To prevent deadlock, an asynchronous protocol must never wait for messages from more than n-t processes and must require each process to transmit the message that helps resolve a wait state (in other processes) before entering that wait state itself. A more subtle effect of asynchrony is the difficulty of securing some independence between the coin tosses and the forced value. This is discussed further in the next section.

A Pernicious Effect of Asynchrony

In a synchronous algorithm, the following is a sufficient procedure to generate the forced values. First, processes exchange their proposed agreement values. Then, each process sets its forced value to 0 or 1 if it detects that a majority of correct processes have initial proposed values of 0 or 1, respectively. A process detects that a majority of correct processes hold a value if it receives more than (p+t)/2 messages containing the value. A process that does not detect a majority value has an undetermined forced value. This procedure is easily seen to meet the requirements of (1) and (2) when $n > 3t$.

Faulty processes cannot change the forced (majority) value after all processes enter a particular phase, but they may be able to influence which processes detect the forced value and which resort to using their coin toss values to update their proposed values for the next phase. If the faulty processes can determine some coin toss values for the next phase, they can use their influence on processes in the current phase to affect the forced value for the next phase and thereby cause disagreement to continue.

This kind of cross-phase subterfuge is a concern for all the published asynchronous protocols. To secure some probability of a unanimous coin toss equal to the forced value, as required by (4), the potential control of the faulty processes over the forced value is limited in several ways. Most of the asynchronous protocols reviewed here use a stronger criteria for determining forced value than detection of a correct majority. Toueg [10] uses authentication in determining the forced value.

All these protocols prevent any correct process from revealing its coin toss for the phase until it has received communication for determining the forced value from n-t processes that appear to be in the phase. The protocols by Toueg [10], Ben-Or [1], and Bracha [2] allow a correct process to accept for this communication only messages that have been echoed by a majority of correct processes, thus limiting the ability of faulty processes to send differing messages. The proof that these techniques can effectively limit the power of the faulty processes and secure the independence requirement of (4) is beyond the scope of this paper, as this is one aspect in which the published protocols vary significantly.

All the protocols reviewed here use either local coin tosses or a reliable dealer to distribute a global coin toss. Thus, the faulty processes cannot influence coin toss values, and only their influence on the forced value must be checked.

Review of Published Protocols

We will review several of the probabilistic protocols for Byzantine agreement published in the literature [1,2,3,7,8,10]. The techniques each uses for generating the forced value and coin tosses are discussed. Of course, the authors of these papers do not describe their algorithms as variants of a common, general algorithm, and each uses their own notation and terminology. A major theme of this paper is that these algorithms do fit into the general framework described above and that this framework can be used to understand these diverse algorithms and prove them correct.

One group of protocols uses Shamir's method for sharing secrets [9] to provide a global sequence of random bits for the coin toss values. A reliable dealer process generates a sequence of random bits. Each random bit is encoded into "pieces" that are numbered and signed and distributed to the processes. The pieces are constructed so that the random bit may be reconstructed from any t+1 pieces, but that fewer pieces provide no information about the random bit. Any t+1 processes can cooperate in reconstructing the desired random bit in the sequence, but at least one of these processes must be correct. Because the pieces are numbered and signed, the faulty processes cannot introduce spurious pieces.

The Byzantine agreement protocols that use Shamir's shared secrets technique require correct processes to accept successive random bits as their coin tosses. The coin toss values of correct processes are therefore the same, and this value is independent of the forced value providing no correct process reveals its piece of the coin toss for the phase until the forced value is determined. The constant c is one-half, guaranteeing probable quick agreement regardless of the number of processes. This ingenious methodology was first proposed by Rabin [8]. Note that this methodology requires processes to decrypt the pieces of the coin toss values that are signed by the dealer, but does not necessarily require processes engaged in an agreement protocol to encrypt messages or decrypt messages other than received pieces of the coin toss values.

Rabin's protocol functions asynchronously and allows a correct process to reveal its piece of the random bit for the phase as soon as it receives the proposed agreement values from n-t processes. Rabin did not view the selection of a new proposed value as a choice between a forced value and coin toss value. Instead, each process uses the random bit to select between a weak and strong criterion for using the plurality of received agreement values as its new proposed agreement value. In terms of the framework expounded here, a process's coin toss value is either a default value or a function of the proposed agreement values it receives from other processes, depending on the value of the random bit. This allows the faulty processes to attempt

influence on both future forced values and coin toss values. Because of this weakness, Rabin's protocol only works when fewer than a tenth of all processes are faulty.

Perry [8] recast Rabin's protocol within the framework described here. He realized the importance of using the random bit directly as the alternate for the new proposed values, and carefully selected his criteria for choosing a forced value. Because of this, his asynchronous protocol requires only $n > 6t$, and his synchronous protocol reaches the known bound of $n > 3t$ for unauthenticated algorithms. (Perry's description of his algorithm was the major impetus behind our description of a general probabilistic algorithm, and he is responsible for the term "forced value" and was the first to clearly explain the importance of limiting the influence of faulty processes over the forced value in an asynchronous environment.)

Toueg [10] extends Perry's work through the use of authentication to achieve optimal limits of fault tolerance, requiring $n > 2t$ for his synchronous protocol and $n > 3t$ for his asynchronous protocol. As previously mentioned, his asynchronous protocol uses an echo broadcast procedure, where each process accepts only messages echoed by a majority of correct processes, to limit the ability of faulty processes to spread differing messages.

Ben-Or [1] published the first algorithm for reaching asynchronous Byzantine agreement, and it remains the simplest. Like Toueg, he uses an echo broadcast procedure in choosing the forced value, though he does not call it such. Processes make entirely local coin tosses, and only luck causes them to coincide. Thus, the constant c depends on both the number of processes and the number of faulty processes. However, he proves that c does not increase with n if t increases no faster than the square root of n. For correct operation, his algorithm requires only that fewer than a fifth of all processes fail.

Bracha [2] extends Ben-Or's protocol by putting even more severe checks on the ability of faulty processes to disseminate confusing information during the calculation of the forced value. The calculation of the forced value requires three communication exchanges, and a validate function is used to reject any message from a process that conflicts with that process's past message history during previous exchanges. This allows Bracha's protocol to achieve the maximum amount of fault tolerance; he requires only that $n > 3t$. The cost is greater message traffic.

The protocols of both Ben-Or and Bracha perform poorly in the sense that the expected time to terminate increases exponentially in the number of processes involved. (If faulty processes increase as the square root of total processes, the expected time for these protocols remains constant.) Coan and Chor [3] develop a method for generating coin toss values that is neither as haphazard as entirely local tosses, nor requires the cost of decrypting signatures as required by a global "shared

secrets" coin toss. They divide processes into groups, which alternate the responsibility for generating coin tosses for different phases. Each correct process accepts as its coin toss what it perceives to be the majority value of the coin tosses of the group for the phase. Different processes may perceive different majorities if the current group contains faulty processes, but the authors show that this method limits the power of faulty processes to influence the coin toss over consecutive phases. The drawback to this protocol is that because groups may be small and their messages must be received, it is necessarily synchronous.

Conclusion

In this paper we have developed a general framework for the understanding and proof of the currently known probabilistic protocols for reaching Byzantine agreement. Individual protocols were reviewed and their differences discussed.

An immediate question suggests itself: are there probabilistic protocols that cannot easily be fit into the general algorithm described here? In particular, can any advantage be obtained by probabilistic protocols that make use of historic information about previous phases?

Within the framework developed, reaching Byzantine agreement depends on the ability to calculate a global random coin toss. This highlights the importance of this latter problem, which plays a crucial role in many probabilistic, distributed algorithms (not just Byzantine agreement).

Thus, the second area indicated for future research is the development of further protocols within the general framework, particularly ones that do not rely on a reliable dealer to distribute secret random bits. The protocols of this type discussed above are handicapped either by poor performance (Ben-Or and Bracha) or restriction to a synchronous environment (Chor and Coan). A method for efficiently generating (probably) global coin tosses in an asynchronous environment in the presence of faults is the key to the development of such a protocol.

References

[1] M. Ben-Or
Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract)
Proc. 2nd ACM PODC (1983)

[2] G. Bracha
An Asynchronous [(n-1)/3]-resilient Consensus Protocol
Proc. 3rd ACM PODC (1984)

[3] B. Chor and B. Coan
A Simple and Efficient Randomized Byzantine Agreement Algorithm
IEEE order number CH2082-6/84/0000/0098$01.00 (1984)

[4] J. Fischer, N. Lynch, and M. Paterson
Impossibility of Distributed Consensus with One Faulty Process
JACM 32(2) (Apr 1985)

[5] N. Lynch and M. Fischer
A Lower Bound for the Time to Assure Interactive Consistency
Inf. Proc. Ltrs. 14(4) (Apr 1982)

[6] M. Pease, R. Shostak, and L. Lamport
Reaching Agreement in the Presence of Faults
JACM 27(2) (Apr 1980)

[7] K. Perry
Randomized Byzantine Agreement (Extended Abstract)
Cornell University

[8] M. Rabin
Randomized Byzantine Generals
Proc. 24th FOCS (1983)

[9] A. Shamir
How to Share a Secret
CACM 22(11) (Nov 1979)

[10] S. Toueg
Randomized Byzantine Agreements
Proc. 3rd ACM PODC (1984)

[11] R. Turpin and B. Coan
Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement
Inf. Proc. Ltrs. 18(2) (Feb 1984)