Distributed Consensus Revisited*

Gil Neiger

GIT–CC–93/45

July 26, 1993
Revised October 7, 1993

Abstract

Distributed Consensus is a classical problem in distributed computing. It requires the correct processors in a distributed system to agree on a common value despite the failure of other processors. This problem is closely related to other problems, such as *Byzantine Generals*, *Approximate Agreement*, and *k-Set Agreement*. This paper examines a variant of *Distributed Consensus* that considers agreement on a value that is more than a single bit and requires that the agreed upon value be one of the correct processors' input values. It shows that, for this problem to be solved in a system with arbitrary failures, it is necessary that more processors remain correct than for solutions to *Distributed Consensus* and for cases where agreement is only a single bit. Specifically, the number of processors that must be correct is a function of the size of the domain of values used. Two existing consensus algorithms are modified to solve this stronger variant.

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

*This work was supported in part by the National Science Foundation under grants CCR-9106627 and CCR-9301454.

1 Introduction

A classical problem in distributed computing is that of having processors agree on a common value despite the presence of failures in the system. There are many versions of this problem; one of the best known is *Distributed Consensus*. This problem is specified as follows. Each processor begins with an input value. The processors then execute an algorithm; each execution of the algorithm must satisfy the following conditions:

- Termination. All processors must choose a single output value and halt.
- Agreement. The output values of the correct processors must be identical.
- Validity. If all correct processors begin with the same input value, then they must end with that same value.

Typically, such algorithms can tolerate only a certain number of failures. Generally, t is used to represent the fault tolerance of an algorithm; the three conditions above need hold only for executions in which at most t processors fail. Fischer [14] gives an overview of different forms of this and related problems. The most closely related forms are *Byzantine Generals* [19], in which only one processor (the broadcaster) has an input value, and *Interactive Consistency* [25], in which processors must agree on a vector of output values (one value per processor). More recently, researchers have considered further variants of this problem [9].

The complexity of solutions to these problems depends on properties of the system in which the solutions are to be executed. One important system parameter is the synchrony of message passing. Fischer, Lynch, and Paterson [15] have shown that consensus is impossible in systems with asynchronous message passing even if only a single stopping failure can occur. For that reason, most research (including this paper) concentrates on systems with synchronous message passing. That is, algorithms can be written to execute in a sequence of rounds, in each of which processors first send, then receive messages, and finally execute local computation before passing on to the next round.

Another important system parameter is the type and number of failures that may occur. Most research has focused on systems with *arbitrary* failures (these are also called *malicious* or *Byzantine*). A processor that fails arbitrarily may take any action whatsoever; it may send spurious messages, change its state incorrectly, or take actions completely contrary to the algorithm being run. An algorithm written to tolerate arbitrary failure is desirable, because it makes no assumptions regarding the behavior of faulty processors. (Other research has considered *Distributed Consensus* in the presence of more restricted faulty behavior [13, 17, 20, 22, 23, 24].[1]) If processors may fail arbitrarily, then solutions to *Distributed Consensus* are not possible if too many failures may occur. Specifically, Lamport, Shostak, and Pease [19] show that the total number of processors n must be greater than three times the number of failures to be tolerated

[1] In such systems, the Validity condition is often rephrased to read "If all processors begin with the same input value, then they must end with that same value." In the systems considered here, it does not make sense to consider input values of faulty processors, as their malicious behavior can render their actual inputs irrelevant.

t; that is, fewer than one-third of the processors may be faulty. They also showed that, as long as $n > 3t$, solutions to the problem exist. Other solutions have been exhibited that require even more correct processors [1, 3, 5, 6, 7, 8, 10, 11, 21, 28]; these have significantly lower communication complexity than the original algorithm of Lamport et al. Only recently, has an algorithm been developed that requires only $n > 3t$, while running in an optimal number of rounds with polynomial-time local computation [16].

The original specification of *Distributed Consensus* requires that a specific value be the decision value only when all correct processors begin with that value. (This is the Validity condition.) This is a reasonable constraint if the domain of values considered is only $\{0, 1\}$. That is, the Validity condition is equivalent to stating that the value chosen must be the initial value of some correct processor: if they all begin together, then they must retain their initial value; if they begin with a mix of 0's and 1's, then either value is a legal output. This paper considers cases in which the domain contains more than 2 values. In these cases, different formulations of the Validity condition may have different meanings.

This paper studies *Strong Consensus*, which is specified by retaining the original Termination and Agreement properties and adding the following:

- Strong Validity. The output value of each correct processor must be the input value of some correct processor.

The two Validity conditions are not identical. Suppose, for example, that the domain of legal values is $\{0, 1, 2\}$ and consider a situation in which each correct processor begins with either 0 or 2. According to the original Validity condition, it is permitted for every correct processor to end with 1! This would violate Strong Validity. Strong Validity, which requires the value chosen to be an initial value, seems more natural and algorithms satisfying it will have greater applicability. In fact, Chaudhuri used this formulation of Validity in her definition of *k-Set Agreement* [9]; she showed that versions of this problem with weaker forms of Validity were solvable with trivial algorithms.

There has been earlier work considering *Distributed Consensus* for domains of $m > 2$ values. Dolev et al. [12] showed how $\lceil \log_2 m \rceil$ binary algorithms, running in parallel could yield the bits that compose an overall value. However, this will not guarantee Strong Validity as the bits chosen might be chosen from different values (e.g., if there are initial values $2 = 10_2$ and $1 = 01_2$, the final decision could be $3 = 11_2$). Perry [26] and Turpin and Coan [27] showed how a binary algorithm could be transformed into a multi-valued algorithm with the addition of one round of communication. However, their technique has processors agree on a "default" value (which might not be any correct processor's initial value) if initial disagreement is detected. This may violate Strong Validity. No previous work on multi-valued agreement has explicitly provided Strong Validity.

This paper considers algorithms that provide Strong Validity and shows that, if values are chosen from a domain of m values, then there is no solution unless $n > \max\{mt, 3t\}$. In addition, it shows that modifications of some existing algorithms for *Distributed Consensus* solve *Strong Consensus*. One of these does so if $n > \max\{mt, 3t\}$, so the bound on the number of processors required is tight.

2 Definitions, Assumptions, and Notation

A distributed system is a set P of n processors connected by bidirectional communication links. Processors share no memory; they communicate only by sending messages along these links. Each processor has a local *state*; this state changes with time based on the algorithm run by the processor.

Processors communicate with each other in synchronous *rounds*. In each round, a processor first sends messages, then receives messages sent in that round, and then changes its state. A *distributed algorithm* (or *algorithm*) specifies, for each processor, what messages it should send in a given round (based on its current state) and how it should change its state (based on its previous state and on the messages it receives in that round). A processor is *correct* if it always sends messages and changes state according to its algorithm. A processor that is not correct is *faulty*; a faulty processor may take any action at all. It may send any message whatsoever, and faulty processors may even conspire to confuse the correct processors. Let t be an upper bound on the number of processors that may be faulty in any execution.

This paper is concerned with algorithms that solve *Strong Consensus*. In such an algorithm, processors attempt to reach agreement on a value chosen from some fixed domain D. Let $m = |D|$; clearly, the problem is interesting only if $m \geq 2$. The set of states is divided into two classes: *undecided* states and *decided* states. Every processor begins in an undecided state. Of the undecided states, there are m *initial states*, one corresponding to each value in D. If a processor begins in the initial state corresponding to v, then it is said to have v as its *initial value*. Each decided state also corresponds to some value $v \in D$; the state is said to be *decided for* v. Any algorithm for *Strong Consensus* must be such that, if a processor enters a state that is decided for some value $v \in D$ then, from that point on, the processor enters only states that are decided for v. Furthermore, every run of the algorithm is such that, if all processors begin in initial states, the following three properties hold:

- Termination. Every correct processor enters a decided state.
- Agreement. If two correct processors enter states decided for v_1 and v_2, respectively, then $v_1 = v_2$.
- Strong Validity. If some correct processor enters a state decided for v, then v is the initial value of some correct processor.

The traditional formulation of *Distributed Consensus* uses instead the following weaker third condition:

- Validity. If all correct processors have initial value v and some correct processor enters a state decided for v', then $v' = v$.

Observe that Strong Validity implies Validity. Suppose that Strong Validity holds, that all correct processors have the same initial value v, and that some correct processor enters state decided for v'. By Strong Validity, some correct processor had initial value v'. This implies that $v = v'$ and Validity holds. Thus, any algorithm that solves

Distributed Consensus also solves *Strong Consensus*. The next section shows that the converse is not true.

3 The Impossibility Result

Lamport, Shostak, and Pease [19] proved that *Distributed Consensus* can be solved only if $n > 3t$, that is, only if fewer than one-third of the processors may fail. In addition, they exhibited an algorithm for *Distributed Consensus* that is correct as long as $n > 3t$. It turns out that algorithms for *Distributed Consensus* do not always solve *Strong Consensus*. This section shows that *Strong Consensus* is solvable only in systems for which $n > \max\{mt, 3t\}$, where m is the size of the set D of values from which input values are chosen. The requirement $n > 3t$ results from the fact that any algorithm for *Strong Consensus* also solve *Distributed Consensus* and that the latter requires $n > 3t$. The remainder of this section show that $n > mt$ if *Strong Consensus* is solvable.

Intuitively, the argument proceeds as follows. Suppose that $n = mt$. This means that there may be as few as $(m-1)t$ correct processors. If the initial values of the correct processors include all m values, then Strong Validity is trivially satisfied. Suppose instead that there are at most $m - 1$ initial values held by the correct processors. It is possible that there are at most t correct processors with each of these values. The t faulty processors may all behave as if they were correct but started with the missing mth value. Since there is no way to tell which processors are correct, it is conceivable that the value chosen for agreement may be this mth value. The next paragraphs formalize this argument.

Suppose that there is some algorithm that solves *Strong Consensus* for $n \leq mt$. Assume that $D = \{v_1, v_2, \ldots, v_m\}$. Partition the set P of processors into m subsets P_1, P_2, ..., P_m such that, for all i $(1 \leq i \leq m)$, $0 < |P_i| \leq t$. Let r be the run of the algorithm with the following properties: (1) all processors are correct and (2) for each i $(1 \leq i \leq m)$, all processors in P_i have initial value v_i. By Termination and Agreement above, there is some j, $1 \leq j \leq m$, such that all processor eventually enter a state decided for v_j; that is, they agree on value v_j. Strong Validity is trivially satisfied, as there is a correct processor that begins with each of the m values in D.

Consider now another run r' of the algorithm. All processors in $P - P_j$ are correct; processors in P_j are faulty. Note that, because $|P_j| \leq t$, there are at most t processors that are faulty in r'. For each i $(i \neq j)$, all processors in P_i have initial value v_i. Thus, no correct processor has initial value v_j. The processors in P_j behave exactly as they do in r; that is, each processor in P_j sends exactly the same messages to processors in $P - P_j$ in exactly the same rounds (in other words, each processor in P_j "pretends" that its initial value is v_j). It should be clear that each processor in $P - P_j$ will behave identically in r and r', as it has the same initial value and will receive exactly the same messages in each round (this can be shown by an inductive argument). Thus, each processor correct in r' will enter a state decided for v_j. But no correct processor had v_j as an initial value. Thus, Strong Validity is not satisfied. This contradicts the fact that the algorithm solved *Strong Consensus*. Thus, no such algorithm can exist.

Since any algorithm for *Distributed Consensus* requires $n > 3t$ and any algorithm

for *Strong Consensus* also solves *Distributed Consensus*, any algorithm for *Strong Consensus* requires $n > \max\{mt, 3t\}$.

4 Algorithms for Strong Consensus

Many algorithms for *Distributed Consensus* are written explicitly for the domain $D = \{0, 1\}$. As noted above, the two forms of consensus are equivalent for this domain. The fact that, if $m \gg 3$, *Strong Consensus* may require many more correct processors than *Distributed Consensus* suggests that algorithms for the latter may not correctly solve the former. The remainder of this section considers two algorithms for *Distributed Consensus* and explores their correctness for *Strong Consensus*. Before doing so, however, let us consider the general functioning of consensus algorithms.

Most consensus algorithms have each processor maintain a particular "preferred" value; at first, this is the processor's initial value, while at the end it is the value decided upon. These algorithms function so that, at a certain point, all correct processors have the same preferred value. It is possible, however, that processors will have reached agreement without knowing it; they may continue execution of the algorithm. For this reason, most consensus algorithms find it important to be *agreement-preserving*; that is, once the correct processors have reached agreement on a particular value, they do not change their values. Many algorithms use Agreement Preservation to obtain simple Validity. Recall that Validity is relevant only if all correct processors have the same initial value. If this is the case, then Agreement Preservation will ensure that processors retain and thus decide upon this value. Thus, Validity comes "for free."

Other algorithms, especially those designed with $D = \{0, 1\}$, operate by "detecting disagreement" [5, 26, 27]. Validity requires agreement on a particular value only if there is initial unanimity for that value. As soon as a processor detects that there was some initial disagreement, it can "safely" choose some default value. This may facilitate agreement.

These techniques do not lend themselves to solving *Strong Consensus*. Agreement Preservation does not automatically yield Strong Validity because that condition specifies that certain values may be chosen even if there is no initial unanimity. Similarly, Disagreement Detection does not permit the selection of a predefined default value; if the default value is not the initial value of some process, then deciding upon it would violate Strong Validity.

The following subsections consider two algorithms for *Distributed Consensus*. The first is the original exponential-time algorithm of Lamport, Shostak, and Pease [19] as formulated by Bar-Noy et al. [2]. This algorithm uses neither Agreement Preservation nor Disagreement Detection. As a consequence, a minor modification correctly solves *Strong Consensus* if $n > \max\{mt, 3t\}$. The second algorithm is the phase-king algorithm of Berman and Garay [3]. This algorithm makes use of Agreement Preservation. A small but more substantial modification correctly solves *Strong Consensus* if $n > \max\{2mt, 4t\}$.

4.1 An Exponential-Time Algorithm for Strong Consensus

Consider now the exponential-time algorithm of Lamport, Shostak, and Pease as presented by Bar-Noy et al. Processors store values in a tree of depth $t+1$ that is structured and labeled as follows. Each node of the tree is labeled by a nonrepeating sequence σ of processor identifiers such that the root is labeled by the empty sequence) and the parent of a node labeled by sequence σp is labeled σ. Thus, if a node is labeled by a sequence of length ℓ, then it is at depth ℓ in the tree. Because all sequences are nonrepeating, the root has n children, each child of the root has $n-1$, and nodes at depth t each have $n-t$ leaves as children. If a node is labelled by the sequence σp, then node is said to correspond to processor p (the root corresponds to no processor).

The algorithm operates as follows. For $t+1$ rounds, processors send messages and store values at nodes in their trees. In the first round, processors send their initial values. Processor p stores the value received from processor q in the node labelled q at depth 1 (it stores its own initial value at the node labelled p). In each subsequent round i, a processors sends to all processors the values stored in the nodes of its tree at depth $i-1$. If a processor receives a message from p indicating that it has value v stored at node labeled σ, then the processor stores the value v at its node labeled σp. Recall that the tree does not contain nodes whose label includes the same processor identifier twice; thus, some information received is not stored in the tree. Note the following facts about the algorithms: (1) if the last identifier in σ is that of a correct processor, then all correct processors store the same value for σ and (2) if correct processor p stores v at interior node σ, then all correct processors store v at node σp (if it exists). Both these facts follow from the fact that correct processors send the same messages to all other processors.

After $t+1$ rounds, each processor applies a function *Resolve* to the root of its tree and decides upon the value returned by the function. The function *Resolve* is defined recursively as follows: If applied to a leaf node (i.e., a node at depth $t+1$), it returns the value stored at that node. If applied to an interior node, then the function is applied recursively to the children of that node. If there is a majority among the values returned, then that value is returned; otherwise, a default value is returned. Bar-Noy et al. show that this algorithm solves *Distributed Consensus* as long as $n > 3t$.

This algorithm can be easily modified to solve *Strong Consensus*. The function *Resolve* is modified slightly, yielding a function called *Resolve'*. On leaf nodes, *Resolve'* also returns the value stored at the node. For interior nodes, the value returned is the most common of the values returned (by recursive calls to *Resolve'*) for the node's children. If there is more than one such value, then the value returned is the one that appears first in any fixed enumeration of the values in D: $v_1, v_2, \ldots, v_m$. (All processors must use the same enumeration.) Note that, if $m = 2$ and the default value is first in the enumeration of D, then *Resolve'* = *Resolve*.

We can now show that, if $n > \max\{mt, 3t\}$, modified algorithm solves *Strong Consensus*. (Much of this proof parallels that of Bar-Noy et al.) Observe first that the following holds of *Resolve'*:

Lemma 1: *If a node σ corresponds to a correct processor, then all correct processors*

return the same value for $Resolve'(\sigma)$, and this is the value they originally stored at the node.

Proof: Recall that, because σ corresponds to a correct processor, all correct processors originally store the same value v for σ. The proof is now by induction on the height h of σ. If $h = 0$, σ is a leaf and all correct processors return $Resolve'(\sigma) = v$ by definition. Now assume that the lemma holds for all nodes of height h and let σ be at height $h+1$; σ thus has $n-t+h$ children. Those that correspond to correct processors also have v stored at them originally and, by induction, all correct processors will have $Resolve'$ return v for these nodes. Since $n > 3t$, σ has $n - t + h \geq n - t > 2t$ children at height h. At most t of these correspond to faulty processors, so a majority of these correspond to correct processors. Thus, all correct processors have $Resolve'$ return v for a majority of σ's children and, by the definition of $Resolve'$, they all return v for σ, completing the proof. □

If every path from a node to a leaf contains a node corresponding to a correct processor, the node is said to have a *correct frontier*. Note that every path from the root to a leaf contains nodes corresponding to $t+1$ processors; thus, the root always has a correct frontier.

Lemma 2: *If node σ has a correct frontier, then all correct processors return the same value for $Resolve'(\sigma)$.*

Proof: Again, the proof is by induction on the height h of σ. If $h = 0$, then σ itself corresponds to a correct processor. Lemma 1 implies that all correct processors return the same value for $Resolve'(\sigma)$. Now assume that the lemma holds for nodes of height h and let σ be at height $h+1$. There are two possible cases. In the first, σ corresponds to a correct processor (this cannot be the case if σ is the root). Again, Lemma 1 implies that all correct processors will return the same value for $Resolve'(\sigma)$. The other possibility is that all children of σ (at height h) have correct frontiers. By induction, this means that, for each child σp, all correct processors return the same value v_p for $Resolve'(\sigma p)$. Since $Resolve'$ is defined deterministically and identically for all processors, all correct processors must return the same value for $Resolve(\sigma)$. □

One can now prove that the modified algorithm satisfies *Strong Consensus* when $n > \max\{mt, 3t\}$. Termination is obviously satisfied. Agreement is satisfied by Lemma 2 and the fact that the root always has a correct frontier.

To show that the modified algorithm also satisfies Strong Validity, let $D_0 \subseteq D$ be the initial values of the correct processors. If $D_0 = D$, then Strong Validity is trivially satisfied. If $D_0 \neq D$, then there are at most $m-1$ values among the correct processors. Since $n > mt$, there are at least $n - t > (m-1)t$ correct processors. Thus, for at least one value in D_0, there are more than t correct processors with that initial value.

Consider now what happens when some correct processor applies $Resolve'$ to the root of its tree. It first applies $Resolve'$ to the n children of the root. By Lemma 1, each depth 1 node that corresponds to a correct processor returns the value originally

```
v := initial value

for k := 1 to t + 1 do

    send v to all processors
    receive messages from all processors
    foreach x ∈ D do
        c[x] := number of x's received
    v := any x ∈ D such that ∀y ∈ D(c[y] ≤ c[x])

    if i = k then
        send v to all processors
    receive v_k from p_k
    if c[v] ≤ 3n/4 then
        v := v_k

decide(v)
```

Figure 1: Phase-King Protocol for Processor p_i

stored at that node, which is the initial value of the corresponding processor. By the above observation, some value in D_0 is returned by *Resolve'* for more than t nodes at depth 1. Again by Lemma 1, any depth 1 node for which *Resolve'* returns a value in $D - D_0$ must correspond to a faulty processor. Thus, any such value is returned by *Resolve'* for at most t nodes at depth 1. Thus, the value returned by *Resolve'* for the root must be value in D_0. Since each correct processor decides on this value, Strong Validity is satisfied and the algorithm indeed satisfies *Strong Consensus*. Thus, the exponential-time algorithm solves *Strong Consensus* if $n > \max\{mt, 3t\}$.

Many algorithms for *Distributed Consensus* are derived from the algorithm given above [2, 4, 5, 16, 21, 28]. While it is likely that many of them can be similarly modified to solve *Strong Consensus*, some derive their improved performance by taking advantage of the fact that $D = \{0, 1\}$. A further exploration is necessary to determine which can and which cannot be modified to give algorithms for *Strong Consensus*.

4.2 A Phase-King Algorithm for Strong Consensus

This section discusses the *phase-king* algorithm of Berman and Garay for *Distributed Consensus* [3]. An adaptation for domains of size greater than 2 is given in Figure 1. The algorithm is correct as long as $n > 4t$. The algorithm functions in a series of $t + 1$ phases, each composed of two rounds of communication; thus, there are a total of $2t + 2$ rounds of communication. In the first round of each phase, all processors

exchange their values. A processor then sets its preferred value to be one for which it received a plurality of messages. In the second round, the processor whose identifier is the same as the round number—the "phase king"—sends its value to all. Any processor that is not sufficiently "sure" of its value (i.e., that did not receive an overwhelming number of messages for its value) adopts the king's value instead.

The proof of the algorithm's correctness hinges on two properties. The first is the "good-king property," meaning that, at the end of any phase in which the king is correct, all processors are in agreement. This is because any processor that ignores a good king's message already has the same value as that king. The second property is Agreement Preservation: if all processors begin a phase with the same value, then all will ignore the king and retain this same value. As noted above, the Agreement Preservation ensures that the algorithm satisfies normal Validity.

For domains with size greater than 2, this algorithm does not solve *Strong Consensus* because it might not achieve Strong Validity. Consider the domain $D = \{0, 1, 2\}$ and suppose that half the processors begin with 0 and half with 1. After the first round of the first phase, all correct processors have $v = 0$ or $v = 1$, but all have $c[v] \leq 3n/4$. Suppose now that p_1 (the king of the first phase) is faulty and sends 2 as its value in the second round. All correct processors then set their value to 2. By Agreement Preservation, all eventually decide on this value. A single faulty processor thwarts Strong Validity!

The algorithm can be modified to correctly solve *Strong Consensus* as long as $n > 2mt$ (notice that, since any interesting instance of the problem has $m \geq 2$, this implies $n > 4t$, which was already required by the original algorithm). The modification is obtained by replacing "$c[v] \leq 3n/4$" as the condition for taking the king's value with "$c[v] \leq 3n/4$ **and** $c[v_k] > t$" in Figure 1. Recall that v_k is the value received from the king in the second round of a phase k. Now, the king's value is "believed" only if there were enough "votes" for it in the first round to ensure that at least one correct processor had that value at the beginning of the phase.

The problem with this modification is that we may lose the Good-King Property and, without it, the algorithm might not satisfy Agreement. The requirement $n > 2mt$ remedies this concern. Consider the messages received by correct p_k in the first round of phase k. A simple counting argument shows that the value v_k chosen by p_k must be such that $c[v_k] > 2t$.[2] Since there are at most t faulty processors, any other processor must have $c[v_k] > t$, and will thus accept the king's value (assuming that it also has $c[v] \leq 3n/4$). Note that, if a processor receives v_k from p_k and finds $c[v_k] \leq t$, then p_k must be faulty.

Since the modified algorithm retains the Good-King Property and satisfies Strong Validity, we conclude that it solves *Strong Consensus*.

[2]A processor treats a missing message as if it actually received a message with some value. The actual value chosen is unimportant.

5 Conclusions

This paper has described a stronger version of *Distributed Consensus* called *Strong Consensus*. It differs from the normal form by strengthening the Validity condition: the consensus value chosen must be the initial value of some correctprocessor. While this rephrasing does not change the semantics of the problem when choosing from a domain of only 2 values, it does entail a change for larger domains. In many settings, the stronger version of the problem may be the more natural one. For example, if processors' initial values are the readings of different external sensors measuring, for example, the temperature, it would not make sense to agree on a value that was no processor's initial measurement. In other cases, however, this restriction may not be important. If processors' initial values reflect their output of some previous computation, it may be sufficient to agree on a special value indicating "a problem has occurred" if there is initial disagreement, even if no processor began with this special value.

It makes sense to consider how this strengthening affects the solvability of the problem and the correctness of existing algorithms. Section 3 showed that the size of the domain of possible values determines how many processors are needed to solve the problem. If the size of the domain is m, then solutions must require $n > \max\{mt, 3t\}$, where n is the total number of processors and t is the number that may fail. Section 4 then considered two algorithms for *Distributed Consensus* and determined that each could be modified, in different ways, to yield algorithms for *Strong Consensus*. In one case, an algorithm that required $n > 3t$ for *Distributed Consensus* is modified to require $n > \max\{mt, 3t\}$ for *Strong Consensus*; thus, the bound on the number of processors required is tight. In the second case, an algorithm that required $n > 4t$ is modified to require $n > 2mt$.

Neither of the algorithms considered is optimal in all measures. The first requires exponentially large messages; the second requires twice as many rounds of communication as the first. More efficient consensus algorithms often take advantage of the weakness of the Validity condition of *Distributed Consensus*, satisfying it with standard agreement-preservation or disagreement-detection techniques. These techniques do not help when Strong Validity is required. It remains to be seen whether or not these algorithms can be adapted to yield efficient solutions to *Strong Consensus*.

Acknowledgements

The idea of reexamining the definition of *Distributed Consensus* resulted from a series of discussions with Howard Karloff. I am grateful to Brian A. Coan for directing me to earlier research on multivalued consensus and to Rida A. Bazzi for reading earlier versions of this paper.

References

[1] Amotz Bar-Noy and Danny Dolev. Consensus algorithms with one-bit messages. *Distributed Computing*, 4:105–110, 1991.

[2] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement. *Information and Computation*, 97(2):205–233, April 1992.

[3] Piotr Berman and Juan A. Garay. Asymptotically optimal consensus. In *Proceedings of the Sixteenth International Conference on Automata, Languages, and Programming*, volume 372 of *Lecture Notes on Computer Science*, pages 80–94. Springer-Verlag, 1989.

[4] Piotr Berman and Juan A. Garay. Efficient distributed consensus with $n = (3+\epsilon)t$ processors. In S. Toueg, P. G. Spirakis, and L. Kirousis, editors, *Proceedings of the Fifth International Workshop on Distributed Algorithms*, volume 579 of *Lecture Notes on Computer Science*, pages 129–142. Springer-Verlag, October 1991.

[5] Piotr Berman and Juan A. Garay. Cloture votes: $n/4$-resilient, polynomial time distributed consensus in $t+1$ rounds. *Mathematical Systems Theory*, 26(1):3–20, 1993.

[6] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus. In *Proceedings of the Thirtieth Symposium on Foundations of Computer Science*, pages 410–415. IEEE Computer Society Press, October 1989.

[7] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Bit optimal distributed consensus. In R. Yaeza-Bates and U. Manber, editors, *Computer Science Research*, pages 313–322. Plenum Publishing, New York, 1992.

[8] James E. Burns and Gil Neiger. Fast and simple Byzantine agreement. Technical Report 92/12, College of Computing, Georgia Institute of Technology, March 1992. Submitted for publication.

[9] Soma Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. *Information and Computation*, 103(1):132–158, July 1993.

[10] Brian A. Coan. Efficient agreement using fault diagnosis. In *Proceedings of the Twenty-Sixth Annual Allerton Conference on Communication, Control, and Computing*, September 1988. To appear in *Distributed Computing*.

[11] Brian A. Coan and Jennifer L. Welch. Modular construction of an efficient 1-bit Byzantine agreement protocol. *Mathematical Systems Theory*, 26(1):131–154, 1993.

[12] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for Byzantine agreement without authentication. *Information and Computation*, 52:257–274, 1982.

[13] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990.

[14] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In M. Karpinsky, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes on Computer Science*, pages 127–140. Springer-Verlag, 1983.

[15] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[16] Juan A. Garay and Yoram Moses. Fully polynomial Byzantine agreement in $t + 1$ rounds. In *Proceedings of the Twenty-Fifth ACM Symposium on Theory of Computing*, pages 31–41. ACM Press, May 1993.

[17] Vassos Hadzilacos. Byzantine agreement under restricted types of failures (not telling the truth is different from telling lies). Technical Report 18-83, Aiken Computation Laboratory, Harvard University, 1983. A revised version appears in Hadzilacos's Ph.D. dissertation [18].

[18] Vassos Hadzilacos. *Issues of Fault Tolerance in Concurrent Computations*. Ph.D. dissertation, Harvard University, June 1984. Technical Report 11-84, Aiken Computation Laboratory.

[19] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[20] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1):121–169, 1988.

[21] Yoram Moses and Orli Waarts. Coordinated traversal: $(t + 1)$-round Byzantine agreement in polynomial time. In *Proceedings of the Twenty-Ninth Symposium on Foundations of Computer Science*, pages 246–255. IEEE Computer Society Press, October 1988.

[22] Gil Neiger and Rida Bazzi. Using knowledge to optimally achieve coordination in distributed systems. In Yoram Moses, editor, *Proceedings of the Fourth Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 43–59. Morgan-Kaufmann, March 1992.

[23] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, September 1990.

[24] Gil Neiger and Mark R. Tuttle. Common knowledge and consistent simultaneous coordination. *Distributed Computing*, 6(3):181–192, April 1993.

[25] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[26] Kenneth J. Perry. *Early Stopping Protocols for Fault-Tolerant Distributed Agreement.* Ph.D. dissertation, Cornell University, February 1985. Technical Report 85-662, Department of Computer Science.

[27] Russell Turpin and Brian A. Coan. Extending binary Byzantine agreement to multivalued Byzantine agreement. *Information Processing Letters*, 18:73–76, February 1984.

[28] Arkady Zamsky, Amos Israeli, and Shlomit S. Pinter. Optimal time Byzantine agreement for $t < n/8$ with linear messages. In A. Segall and S. Zaks, editors, *Proceedings of the Sixth International Workshop on Distributed Algorithms*, number 647 in Lecture Notes on Computer Science, pages 136–152. Springer-Verlag, November 1992.