DISTRIBUTED COMMIT WITH BOUNDED WAITING

D. Dolev

H. R. Strong

IBM Research Laboratory
San Jose, CA 95193

ABSTRACT

Two-Phase Commit and other distributed commit protocols provide a method to commit changes while preserving consistency in a distributed database. These protocols can cope with various failures occurring in the system. But in case of failure they do not guarantee termination (of protocol processing) within a given time: sometimes the protocol requires waiting for a failed processor to be returned to operation. It happens that a straightforward use of timeouts in a distributed system is fraught with unexpected peril and does not provide an easy solution to the problem. In this paper we will combine Byzantine Agreement with Two-Phase Commit, using observations of Lamport to provide a method to cope with failure within a given time bound. An extra benefit of this combination of ideas is that it handles undetected and transient faults as well as the more usual system or processor down faults handled by other distributed commit protocols.

1. INTRODUCTION

Consistently committing changes to distributed data is an impossible task in some cases and a very "long" process in others [Li,RSL]. The main difficulty is the lack of a reliable mechanism for shared knowledge or agreement: one processor cannot see messages delivered to another processor and each processor has only its message interface to tell it about global events. Such agreement problems are exacerbated in the presence of undetected or transient failures.
The various commit protocols are designed to preserve data consistency in the distributed system in case of failure or delay in message transmission, assuming the system has the ability to recover after failure. A typical commit protocol requires that each process wait in some prepared to commit state until some participating site, especially the transaction coordinator has recovered. We will present algorithms that overcome multiple failures and guarantee a unanimous commit or abort among all the correctly operating processors, subject only to certain limits on the number of failures that can occur. These algorithms will complete processing within a fixed time known in advance. Thus the protocols we present are nonblocking in the terminology of Skeen [Sa,Sb].

In order to guarantee speedy system recovery, one must find a way to agree distributively on failures and to decide on appropriate corrective action. Using timeouts appears at first to be a reasonable solution, but unless all the clocks are totally synchronized and advance at exactly the same rate, a straightforward application of timeouts can lead to undesirable system partition and inconsistency because of a lack of systemwide agreement. For example, suppose we want to modify a two phase commit protocol to include timeout. Assume the transaction coordinator has received a "prepared to commit" message from all participants and begins to send the commit message when there is a transient communication failure. Assume that the participants will abort the transaction unless they receive a "commit" message within a fixed time. If any participating site receives the "commit" message before its timeout, then all must. So what we need is a broadcast protocol that guarantees that if one participant receives the message, then not only will all eventually receive it but they will all receive it within time-out time. Otherwise, some may receive the message before and some after the timeout. But what does "at the same time" mean unless the clocks are synchronized exactly?

The results of Lamport [La] enable us to find agreement on an ordering of events but not the complete synchronization needed for application of timeouts. In [G] Garcia-Molina apparently ignores this difficulty in his distributed election algorithms. Lamport [Lb] suggests using time instead of timeout to overcome this problem. However, to agree on time one needs some sort of Byzantine Agreement [PSL, DSb, DFFLS].

In this paper we will combine the best results on reaching Byzantine Agreement with Lamport's [Lb] observations to obtain a distributed commit protocol. Our assumptions about the nature of errors in the system will be presented in the next section. They are based on those in [Lb] and [G]. In general we allow detectable and undetectable faults in sending or forwarding messages. We allow individual clocks to differ. We do not assume anything about the clock of a faulty processor. Later we will discuss trade-offs involving coping with large numbers of arbitrary types of failures at the cost of increasing the number of messages exchanged.

CH1792-1/82/0000/0053$00.75 © 1982 IEEE

The algorithms we describe here are more robust and comprehensive than the other distributed commit protocols in the literature. The incremental costs in time, messages, or other usual measures of complexity are surprisingly small. Of course we cannot guarantee that our algorithms will cope with every possible combination of errors of every possible type, but, given probability distributions on the various kinds of error possible, we can tune our protocols to achieve any desired level of confidence that is actually achievable by protocols for distributed commit that complete processing <u>within a fixed time bound</u>.

Another nonblocking protocol was presented by Skeen in [Sb]. The presentation is sketchy in details but it seems to rely on a model of failure that assumes failures shut down a processor immediately (see Assumption 2 below) and that there are no intermittent or transient processor or communication link failures. For this model, Fischer and Lamport have developed a simpler, straightforward nonblocking protocol [FL]. However, we believe we are presenting the first nonblocking distributed commit protocol that makes no such assumptions about the behavior of faulty processors.

We use the term Byzantine Agreement to refer to distributed agreement protocols that are resilient to a bounded number of failures without making any assumption about the behavior of faulty components. Other algorithms for reaching Byzantine Agreement are discussed in [DLM, Da, Db, DR, DSc, FFL]. These references include several lower bound results on the time and amount of information exchange required to achieve Byzantine Agreement. Variations on the type of agreement are discussed in [Da] and [La].

2. THE BASIC ASSUMPTIONS

We envision a distributed system consisting of n processors loosely connected by a message interface via a communication medium that can be conveniently analyzed as a collection of processor to processor links. Our distributed commit protocols will be tuned to an assumption about the number of coincident failures of certain types that they are intended to handle. First we present a fairly restrictive set of assumptions that we have adapted from those common to the literature on distributed commit algorithms. Then we will discuss ways to adapt the algorithms in order to relax the assumptions.

Assumption 1 [G]: The network topology is known to the participants in the algorithm, and correct processors obey the algorithm.

The first assumption does not imply that the system should remain fixed forever, but it does require that prior to applying the algorithm the participants should agree on the set of processors participating in the algorithm. This assumption can be relaxed somewhat so that only some of the processors are required to know the complete topology.

Assumption 2 [G]: When a processor fails, it immediately halts all processing. Thus a failure does not cause a processor to deviate from its algorithm and behave in an unpredictable manner.

This assumption is a very strong one. It eliminates from consideration many difficult to handle but usually improbable events. We have included it because most distributed commit protocols must make this assumption in order to make any assurance of consistency. We will devote a later section of the paper to its relaxation.

Assumption 3 [G]: If a processor p receives a message M from processor q, then the message M was sent earlier by processor q.

Here, since we make no assumption about an absolute time frame, we must explain that we mean "earlier" in the relativistic sense that any observer would observe the sending of message M happening before its receipt. In particular, if processor p returns some acknowledgement to message M, then that acknowledgement is received by processor q at a time later than that at which it sent message M according to its own clock (provided that processor q including its clock is operating correctly).

Assumption 3 implies that a processor cannot impersonate another and that messages are not spontaneously generated. Moreover, it implies that the content of a message cannot be altered by the communication medium. There are several methods for justifying Assumption 3 that all fall under the broad heading of authentication protocols and can be tuned to supply a sufficient amount of redundancy to provide any desired level of confidence in the assumption. See [DSb] and [PSL] for a discussion of these methods.

Note that we do not assume that a message sent is necessarily received. We simply assume that if it is received and accepted it is exactly the same message that was sent. Communication links may fail, but their only undetected failures are complete cessations of communication. From the point of view of a processor at one end of a communication link, the failure of the link is not immediately distinguishable from a failure of the processor at the other end. In Section 3 we will discuss how to account for communication failures in setting the maximum number of failures the distributed commit protocol will handle and also how to reduce consideration of communication link failures to a model in which communication is perfect and only processors can fail.

Any use of time to measure missing events is based upon the following assumption.

Assumption 4 [Lb]: There is a δ such that if event e occurs at time T and causes processor p to send message M to processor q and if processors p and q and the communication link joining them are nonfaulty, then the message arrives at processor q by time T + δ (The time is measured in both cases by the same clock.)

We assume that the value of δ is uniform for all processors and unaffected by load or time. We also assume that it is known in advance to each processor. For our purposes, the event e will typically be the receipt of another message, so it should be noted that δ includes processing time as well as transmission time.

The main weakness of the assumption about δ is that it should not be affected by load in the individual processors and in the communication link. But any use of timeout depends on some upper bound on communication time. Whenever the load drives the communication time to exceed this specified upper bound, the event will count as a communication error. We will be careful in handling those errors, sometimes allowing a processor to change its mind after the timeout if there is enough evidence that the missing event did occur.

The main danger in using δ as a basis for a timeout method is that clocks may drift and time is not absolute. Thus one processor may register a timeout while another does not. This discrepancy is the key problem for consistency in distributed systems. To overcome it we will need to add some additional communication between individual processors. As Lamport [Lb] observed, we need to bound the rate of relative drift between clocks and the amount of time by which they differ. In this way we will be able to deduce from an individual time clock something about global time.

Assumption 5 [Lb]: At any time, the clocks of any two nonfaulty processors differ by at most ε .

This assumption does not require absolute synchronization, but it does require a fairly close correspondence in times. It can be relaxed by instead bounding the rate of drift between clocks and by periodically running a Byzantine Agreement on time. An ε satisfying Assumption 5 can be computed from the bound on the rate of drift and the length of the period for Byzantine Agreement on time. Lamport and Melliar-Smith discuss these issues in their recent paper [LM]. This assumption simplifies the relation between individual time clocks, as the following simple lemma shows.

Lemma 1 [Lb]: If processors p and q and the link between them are nonfaulty, and if processor q receives message M from processor p at time T on the clock of processor q, then M was sent by processor p at a time no earlier than T-(δ + ε) on the clock of processor p.

The lemma follows immediately from the above assumptions. It provides a way for us to define the phases of our algorithms for Byzantine Agreement.

3. BYZANTINE AGREEMENT

The assumptions of the previous section limit the types of errors that might occur to either not sending (or forwarding) a message required by the protocol or to losing the synchronization with the rest of the system specified in Assumption 5. This limitation provides an easier setting for reaching Byzantine Agreement than its usual context. For example, we can use the algorithms of [DSb] designed for use with an authentication protocol without any extra protocol beyond that required to satisfy Assumption 2.

Byzantine Agreement is defined in the context of a set of processes including one called the <u>transmitter</u> that is supposed to transmit some value to the others in the presence of some bounded amount of faulty behavior. Byzantine Agreement is <u>reached</u> when:

(I) all correctly operating processes agree on the same value, and

(II) if the transmitter operates correctly, then all correctly operating processes agree on its value.

Reaching Byzantine Agreement was expensive and probably impractical until the discovery of polynomial algorithms with and without authentication [DSb], [DFFLS]. The best known algorithm with authentication is [DSb], which requires t+1 phases of sending information and O(nt) messages to cope with t faults. Without authentication the best algorithm requires 2t+3 phases and $O(nt+t^3 \log t)$ bits of information exchange [DFFLS].

As remarked before, the usual context for Byzantine Agreement includes the assumption that all failures are processor failures, i.e. attributable to a processor. The upper limit t on the number of such failures is a parameter of the algorithm. Thus we must provide a way to account for communication link errors as processor errors in order to make sense of our current application of Byzantine Agreement. All that is required is to provide a way to count the number of actual errors of all types assumed possible and convert this to some number to be compared with the parameter t. We are not required to blame any specific processor for a communication failure. If the number of "errors" we compute is no larger than t then we can guarantee Byzantine Agreement among all correctly operating processors in spite of any failures in communication links to them, provided that they are not isolated (cf. [SS]). A processor isolated from all correctly operating processors will be considered to be a failed processor.

The number of equivalent processor faults is defined to be the number of failed processors plus the smallest number of additional processors required to cover all communication link failures, where one processor covers a communication link when it is one end point or terminal on that link. Our algorithms for Byzantine Agreement are designed to handle up to t equivalent processor faults. Later we will need to modify the definition of a failed processor to include those isolated from a specific subset of the otherwise correct processors. To simplify matters and give one operational definition, we make our definition relative to the parameter t. A processor will be said to have failed if it fails to follow the algorithm or if communication link or other failures prevent it from communicating correctly with more than t other processors. If a processor has not failed, it will be said to be correct, i.e. operating correctly. Note that in spite of Assumption 2, the second type of failure is not always detectable by the processor. Later, associated with Algorithm 3, we will give an operational definition of failure that can be used by an otherwise correctly operating processor to determine when it has lost the connectivity required for agreement.

The following algorithm is an example adapted from [DSb] to meet our current assumptions. It is designed to handle a single equivalent processor fault (i.e. t=1). Note that this is more powerful than handling any single failure but not more powerful than handling any pair of failures.

Since in our distributed commit application we are only interested in agreeing on whether a particular event occurred (e.g. the issuance of the "commit" message by the transaction coordinator or a change in the configuration), we will describe Byzantine Agreement with respect to a paradigmatic event called the "GO" event. In this context Byzantine Agreement is reached when:

(I') all correctly operating processes agree whether or not the "GO" event occurred, and

(II') if the transmitter operates correctly, then all correctly operating processes agree on "GO" if and only if it was sent by the transmitter.

BYZANTINE AGREEMENT COPING WITH SINGLE EQUIVALENT PROCESSOR FAULT:

If a processor receives a message directly from the transmitter, it cannot know that all other processors received that message. Our algorithm will arrange that if the message is received via a backup processor specifically chosen to be a relay processor then it can be assumed that all processors will eventually receive the message. Let the transmitter be s and let two other processors a and b be chosen to be backups for s. We will refer to a and b as the active processors. All processors other than a, b, and s will be called passive processors. The algorithm is presented by rules of correct operation for each type of processor. Note here that a processor must be presumed failed and no longer operating correctly if it is isolated by communication link failures from both active processors. To simplify the statement of the algorithm, we assume that whenever a processor sends a message to another it also sends that message to itself. Let τ be $\delta + \varepsilon$ from the assumptions. We naturally assume that if a processor receives a message from itself, it receives it at a time no later than δ after the time at which it was sent.

Algorithm 1

Transmitter: To send GO the transmitter sends to the two active processors the message
GO, the time is T,
where T is the time on its clock.

Active processors: On receipt of
<GO, the time is T>,
from the transmitter at time T' with $T' \leq T + \tau$, send to every processor the message
<The transmitter sent "GO" at time T>.

All processors: On receipt of
<The transmitter sent "GO" at time T>,
from an active processor at time T" with $T'' \leq T + 2\tau$, decide that the event "GO" did indeed occur.

Theorem 1: Following Algorithm 1 the system reaches Byzantine Agreement on the event "GO" when no more than one equivalent processor fault occurs.

Outline of proof:
If the transmitter correctly sends "GO, the time is T" to all active processors and there is no more than one equivalent processor fault during the entire process, then all correctly operating processors will have received "the transmitter sent 'GO' at T" from at least one active processor by time $T+2\tau$. If any processor receives "the transmitter sent 'GO' at T from an active processor before time $T+2\tau$, then some

active processor must have sent the message and by Assumption 2 it must have received "GO, the time is T" from the transmitter before time $T + \tau$. An exhaustive analysis of the possibilities for one equivalent processor fault shows that in this case each correctly operating processor must receive "the transmitter sent 'GO' at time T" before time $T+2\tau$ as measured on its clock. Thus if by time $T+2\tau$ a correctly operating processor has not received an appropriate message from either active processor, it can conclude that the event had not occurred by time T and that all other correctly operating processors will be in agreement on this fact within ε as measured on its clock. □

The above algorithm is fairly simple and the number of messages exchanged is about 2n, where n is the total number of processors. Algorithm 1 ensures that within time 2τ a decision about the critical event will be achieved. In the next section we will show how to use Algorithm 1 to obtain a distributed commit protocol with a fixed time bound. Later we will generalize Algorithm 1 to cope with any given number of equivalent processor faults with the required number of messages only proportional to the number of faults handled.

4. DISTRIBUTED COMMIT PROTOCOL

Our distributed commit protocol uses Algorithm 1 twice: once for "prepare to commit" and once for "commit." If the "commit" event does not occur in time, then all correct processors will decide to abort the transaction in question.

Algorithm 2

1. Using algorithm 1, the transaction coordinator broadcasts "prepare to commit" (as a "GO" event at time T) to all the participating processors.

2. After deciding that the event "prepare to commit" has occurred, every processor that is ready to commit sends the message "ready to commit" to the transaction coordinator. Those preferring to abort need send nothing.

3. If by time $T+3\tau$ the transaction coordinator has received "ready to commit" from all the participants, then it broadcasts the "commit", using Algorithm 1.

4. If by time $T+5\tau$ on its clock a processor has not decided to commit then it decides to abort.

After the event "prepare to commit" each processor has the time T required in the rest of the algorithm. Some improvement is possible by running Algorithm 1 on an "abort" event as well as on the "commit" event, but the performance in the worst case will be the same.

So far we have been concerned with reaching agreement among the correctly operating processors and we have defined "correctly operating" in such a way that, in spite of Assumption 2, a processor may continue in operation unaware that it is regarded by the system as a whole as having failed (because of transient communication link failures). This situation can present difficulties when we want to reintegrate a failed processor into the system. Byzantine Agreement algorithms can be used to detect and reintegrate such failed systems, but these applications are beyond the scope of the present paper. Here we will add an assumption in the spirit of Assumption 2 and recognize that when we relax either this assumption or Assumption 2 we may admit the possibility of an undetected failure that could lead to inconsistency among all operating processors though not among those actually operating correctly.

Assumption 6: If event e at time T causes processor p to send a message to processor q and processor q fails to receive it by time $T + \tau$ and if both processors are otherwise operating correctly, then both processors will detect the failure by time $T + \tau$.

If we also assume that the backup processors for the transmitter in Algorithm 1 are chosen in advance, then Assumption 6 implies that an otherwise correctly operating processor will detect the situation resulting from its isolation from the active processors and can halt further distributed processing as in Assumption 2 until it is "repaired" and reintegrated with the system.

Assumption 6 could be satisfied by some low level acknowledgement protocol together with frequent periodic checking of the links with messages in both directions. We can weaken it somewhat by asking that only the sender detect the failure; but, in this case, we must require that any single failure to send be sufficient to consider the processor failed. An unfortunate consequence of this requirement would be that a single communication link failure could result in two equivalent processor faults. Later we will discuss relaxing the assumption in a different way so that both ends of the link must discover the failure but not within any fixed time. The relaxed assumption could be satisfied by a simple acknowledgement protocol together with notification after repair.

Theorem 2: Algorithm 2 ensures that all correct processors either commit or abort the transaction within a time bound of 5τ and that if any correct processor commits then all do, provided there is at most one equivalent processor fault. Moreover, if there are no faults and all processors are ready to commit then the transaction will be committed.

The proof of the theorem is straightforward from Theorem 1 and the Assumptions 1 through 6.

So far we have not discussed recovery and reintegration for failed processors. The details of reintegration are beyond the scope of this paper. Here we simply compare our ability in principle to reintegrate with that of a typical distributed commit. Our algorithms emphasize speed of processing for the correct processors at some expense to those that have failed. In the usual two phase commit the log of the Transaction Coordinator determines whether a transaction was committed. Thus any reintegration of any failed processor is dependent on the availability of that log, but only that log need be consulted. In our context there is no central repository for the commit record. Instead a recovering processor must consult the logs of at least t+1 other processors and it must have a consistent report from t+1 of them before it can decide the status of the transaction in question. If we assume in the spirit of Assumption 2 that only the logs of correct processors are available, then we can relax this requirement and consistent with Algorithm 2, we need ask only one correct processor. However, in the next section we will present an algorithm that is robust enough to cope with significant relaxations of our assumptions and when the assumptions are relaxed a consistent report from t+1 processors will be required.

5. BYZANTINE AGREEMENT SUSTAINING ANY T EQUIVALENT PROCESSOR FAULTS

We present now the extension of Algorithm 1 that will be able to cope with any t faults. As in Algorithm 1, we assume that the transmitter is known. We continue the convention of Algorithm 1 that when a processor sends a message to others, it also sends a copy to itself.

For simplicity of presentation we assume that the communication network is complete so that except for failures each processor can communicate with each other within the time δ . Otherwise we would have to adjust the algorithm to include nondirect communication and apply the methods of [Da] and [Db] to effectively satisfy the assumption.

To run Algorithm 3, choose 2t+1 processors to be active (in the position of backup to the transmitter s) and let all the rest be passive. We present the algorithm again as rules of correct operation for the various types of processors.

Algorithm 3

Transmitter: To send "GO" at time T the transmitter sends

<GO, the time is T,s>,

to all active processors.

Active processors: If at time T' with $T'-T\le k\tau$ active processor p receives the message

<GO, the time is $T,s,p_1,p_2,\ldots,p_k$>,

from p_k, and if p did not relay the message "GO" before, then p sends to every processor the message

<GO, the time is $T,s,p_1,p_2,\ldots,p_k,p$>.

All processors: If by time T', where $T'\le T+(t+1)\tau$, a processor has received messages of the form

<GO, the time is $T,s,p_1,p_2,\ldots,p_i$>,

and the total number of different names appended to them is at least t+1,
then decide that the event "GO" did indeed occur. Let a be the number of active processors. If the number of distinct signatures does not exceed t but the sum of the number of distinct signatures and the number of detected communication link faults (on links adjacent to the processor) exceeds a-t-1, then the processor must consider itself failed.

Theorem 3: Following Algorithm 3 the system reaches Byzantine Agreement on the event "GO" provided no more than t equivalent processor faults occur.

Outline of proof:
The proof is an adaptation of the proof of Theorem 6 of [DSb]. Let F be the set of failed processors and let E be a set including F that covers all link failures. Thus $|E|\le t$. Let sig(p) be the number of distinct signatures received by processor p; and let clf(p) be the number of communication link failures adjacent to p. It is straightforward to adapt the argument from [DSb] to show that, if some p not in F has $sig(p)>t$ and if q is not in F, then $sig(q)\ge \min(t+1,a-t-clf(q))$. But if $sig(q)<t+1$ and $sig(q)+clf(q)>a-t-1$ then q is in F. Thus if $sig(p)>t$ then $sig(q)>t$. □

Algorithm 3 can be simplified at the cost of an increase in the number of messages so that all processors are active. In this case again communication with at least t+1 correct processors and ability to follow the algorithm will allow a processor to reach agreement.

6. RELAXATION OF ASSUMPTIONS

A simpler algorithm directly generalizing Algorithm 1 would have satisfied Theorem 3 under Assumptions 1 through 6. However Algorithm 3 is sufficiently robust that we can relax Assumptions 2 and 6 to allow undetected faults and still reach Byzantine Agreement. In order to return an incorrect processor to a state consistent with the rest of the system, it is sufficient to eventually identify the incorrect processor and the time that it became incorrect. Thus with the much weaker assumption that faults will eventually be detected and that a checkpoint prior to the occurrence of the fault can be identified, we can provide the same protection of consistency via transaction logs that is provided by the distributed commit protocols that allow indefinite waiting in the "prepared" state.

We have already remarked that existing authentication protocols are available to satisfy Assumption 3. Moreover, given Assumption 4 and a bound on the rate of drift of clocks, periodic application of Byzantine Agreement using algorithms presented in [DSb] can satisfy both Assumption 1 and Assumption 5. Thus we can relax our set of assumptions to the eventual location of the position and time of faults and bounds on processing time, transmission time, and rate of drift.

We leave for future research the elaboration of algorithms for identification and correction of faults including the reintegration of failed processors, suggesting only that the key to such algorithms lies in Byzantine Agreement.

ACKNOWLEDGEMENTS

The authors would like to thank Won Kim, John McPherson, Mario Schkolnick, Robin Williams, and the referees for many helpful comments on earlier drafts of this manuscript.

REFERENCES

[DH] W. Diffie and M. Hellman, "New direction in cryptography," IEEE Trans. on Inform. IT-22,6(1976), 644-654.

[DLM] R. A. DeMillo, N. A. Lynch, and M. Merritt, "Cryptographic Protocols," proceedings, the 14th ACM SIGACT Symposium on Theory of Computing, May, 1982.

[Da] D. Dolev, "The Byzantine Generals Strike Again," Journal of Algorithms, vol. 3, no. 1, 1982.

[Db] D. Dolev, "Unanimity in an Unknown and Unreliable Environment," 22nd Annual Symposium on Foundations of Computer Science, pp. 159-168, 1981.

[DR] D. Dolev and R. Reischuk, "Bounds on Information Exchange for Byzantine Agreement," Proceedings, ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Aug. 1982.

[DSa] D. Dolev and H. R. Strong, "Polynomial algorithms for multiple processor agreement," proceedings, the 14th ACM SIGACT Symposium on Theory of Computing, May 1982.

[DSb] D. Dolev and H. R. Strong, "Authenticated Algorithms for Byzantine Agreement," IBM Research Report RJ3416 (1982).

[DSc] D. Dolev and H. R. Strong, "Requirements for Agreement in a Distributed System," Proceedings, the Second International Symposium on Distributed Data Bases, Berlin, Sep. 1982.

[DFFLS] D. Dolev, M. Fischer, R. Fowler, N. Lynch, and R. Strong, "Efficient Byzantine Agreement Without Authentication," subbmitted for publication.

[FFL] M. Fischer, R. Fowler, and N. Lynch, "A Simple and Efficient Byzantine Generals Algorithm," this proceedings.

[G] H. Garcia-Molina, "Elections in a Distributed Computing System, " IEEE Trans. on Computers, vol. C-31, no. 1, 1982.

[La] L. Lamport, "The Weak Byzantine Generals Problem," JACM, to appear.

[Lb] L. Lamport, "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems," Technical Report, Computer Science Laboratory, June 1981.

[LM] L. Lamport, and P. M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," Technical Report, Computer Science Laboratory, March 1982.

[LSP] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," ACM Trans. on Programing Languages and Systems, to appear.

[Li] B. G. Lindsay, et. al., "Notes on distributed databases," IBM Research Report RJ2571 (1979).

[LF] N. Lynch, and M. Fischer, "A Lower Bound for the Time to Assure Interactive Consistency," Information Processing Letters, to appear.

[PSL] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," JACM, vol. 27, no. 2, pp. 228-234, 1980.

[RSL] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "System level concurrency control for distributed database systems," ACM Trans. on Database Systems 3:2 (1978), pp. 178-198.

[Sa] D. Skeen, "A Quorum-based Commit Protocol," Proceedings, the 6th Berkeley Workshop on Distributed Data Management and Computer Networks, May 1982, pp. 69-80.

[Sb] D. Skeen, "Nonblocking Commit Protocols," Proceedings, SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, 1981, pp. 133-142.

[SS] D. Skeen and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," Proceedings, the 5th Berkeley Workshop on Distributed Data Managment and Computer Networks, May 1981, pp. 129-142.