On the Reliability of Fault-Tolerant Distributed Computing Systems*

Özalp Babaoğlu

TR86-738
February 1986
(Revised May 1986)

Department of Computer Science
Cornell University
Ithaca, NY 14853

* Partial Support for this work was provided by the National Science Foundation under Grants DCR-86-01864 and MCS 82-10356.

On the Reliability of Fault-Tolerant Distributed Computing Systems*

Özalp Babaoğlu

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

ABSTRACT

The designer of a fault-tolerant distributed system faces numerous alternatives. Using a stochastic model of processor failure times, we investigate design choices such as replication level, protocol running time, randomized versus deterministic protocols, fault detection and authentication. We use the probability with which a system produces the correct output as our evaluation criterion. This contrasts with previous fault-tolerance results that guarantee correctness only if the percentage of faulty processors in the system can be bounded. Our results reveal some subtle and counterintuitive interactions between the design parameters and system reliability.

May 20, 1986

* Partial support for this work was provided by the National Science Foundation under Grants DCR-86-01864 and MCS 82-10356.

1. Introduction

The potential for highly reliable computing is an often-cited benefit of distributed systems [Stan84]. The collection of autonomous processors in such systems provides a natural environment for replicating computations so that failures are masked. To realize this style of fault tolerance, protocols are needed to provide coordination among the replicas in the presence of faulty processors. *Distributed Consensus*† protocols have been proposed as appropriate mechanisms for implementing such coordination [PSL80, LSP82]. The question of structuring highly reliable distributed computing systems based on this paradigm has attracted some interest [Lamp84, SL85, GPD84, CASD85].

The *reliability* of a computing system is the probability that the output it produces is correct. In this paper, we study the reliability of distributed systems that rely on replication and consensus for fault tolerance. Evaluating distributed systems based on this definition of reliability is novel and much more in tune with that of a system designer than the usual criterion where the number of faulty processors during certain periods of execution is assumed to be bounded. In systems designed with the latter premise, no assertions can be made about system properties for executions where the bounds are violated,‡ yet, the reliability calculation of the system must account for events that may violate the bounded-number-of-faulty-processors assumption.

In the next section, we present a model where processors fail independently after an exponentially distributed random time of being correct. No assumptions about faulty behavior are made—once processors fail, they can exhibit arbitrary behavior including collusion. Based on this failure model of processor failures (which has strong empirical support), we compare different alternatives for structuring fault-tolerant distributed systems. Our results reveal some subtle and counterintuitive interactions between system design parameters and system reliability.

For example, in Section 3.1 we characterize those system executions in which consensus can be achieved. We show that a consensus protocol can tolerate failures far in excess of its resiliency if the failures occur in certain patterns. We quantify the probability with which such patterns arise in our failure model. In Section 3.2 we focus on the reliability of a replicated fault-tolerant system based on consensus. We show that as the level of replication increases, the system exhibits more pronounced bimodal

† There are many variants of the *consensus* problem. Collectively, they are often referred to as *Byzantine Agreement*. See [Fisc83] and [SD83] for two brief surveys.

‡ Some protocols, such as those in [MS85] and [LM84] exhibit "graceful degradation" up to a certain number of faulty processors.

behavior—the system is highly reliable until a well-defined time, and highly unreliable thereafter. In Section 3.3 we consider the question "for how many rounds should a consensus protocol be run?" Surprisingly, the answer is not necessarily "for as long as possible." While the probability of the system achieving consensus is an increasing function of the running time, the system reliability is not. In Section 3.4 we quantify the effect of being able to detect all of the failures that have occurred up to some intermediate time in the computation. We describe how consensus protocols can be used to implement a fault detector suitable for this purpose, and show that a significant increase in reliability results if fault detection is performed at the appropriate times.

2. System Structure

Our goal is to reliably execute a single program that performs some arbitrary computation. Without loss of generality, assume that the application cyclically reads input data from external sensors, performs a deterministic computation based on this data, and writes outputs to an external actuator. This structure for a computation has been called a *state machine* [Lamp78].

2.1. Replication and Correctness

To achieve fault tolerance, the state machine (perhaps including the sensor) is replicated and each instance executes on its own processor [Lamp84, SL85, GPD84]. In the rest of the paper, we refer to a processor executing a state machine instance simply as a *processor*. The entire ensemble of n processors is called the *system*. There exists an application-dependent threshold Ψ such that at least Ψ processors must successfully complete the computation in order for the system to remain "correct." More formally, let $F\colon \mathbf{I} \rightarrow \mathbf{O}$ denote the deterministic function that is being computed by each replica of the application. We assume that each processor has been programmed correctly in the sense that if it has not failed by time $t+\Delta$, it computes $F(v(t))$ for all $v(t) \in \mathbf{I}$, representing the input value at time t. Furthermore, a correct processor has to be able to compute the function for all possible input values within some bounded delay Δ.

In the replicated system, let $v_i(t)$ denote the input value at processor i at time t. In general, $v_i(t) \neq v_j(t)$ even if there is a single, perfectly reliable source for inputs to the system since it is impossible for the processors to read a possibly time-varying input in perfect synchrony. Therefore, if each processor were to compute in isolation, there would be no way to reconcile the possibly conflicting outputs even in the absence of failures. To cope with this problem, we introduce a *decision function* G: $\mathbf{I}^n \rightarrow \mathbf{I}$ that can reconcile a vector of n input values into a single value. In this manner, the original function F can be applied to the output of G. Some common decision functions are arithmetic mean, median and majority. Now, if processors compute $F \cdot G$: $\mathbf{I}^n \rightarrow \mathbf{O}$ based on the (same) n values corresponding to the original inputs at each of the replicas, all the correct ones will generate the same output. We say that the *system is correct* at time t if at least Ψ processors generate outputs that are consistent with F as applied to the input vector at time $t-\Delta$. The *reliability* of a system at time t is the probability that it is correct at time t, given that it was initially correct. For some applications, the maximum real time that can elapse between input and output may be critical. Even though a system may be correct according to our definition above, the output may have missed a real time deadline. In so-called "hard real time systems," such an event could be interpreted as an alternate failure mode. Although our present analysis does not address this concern, it could be easily extended to do so.

2.2. Consensus Protocols

Before the computation of function F can commence, the processors must execute a protocol to exchange each others input values. A *distributed consensus* protocol is what is needed to accomplish this. Formally, let v_i, $i=1,2,...,n$ denote the n local input values of processors at some time t. The processors execute a protocol, at the conclusion of which, each processor i has obtained an n-element vector $\mathbf{W}_i$ such that

IC1. If i and j are two correct processors, then $\mathbf{W}_i = \mathbf{W}_j = \mathbf{W} = (w_1, w_2, \cdots, w_n)$.

IC2. If processor i is correct, then $w_i = v_i$.

Note that the protocol requirements do not specify the value for w_i for faulty processor i, as long as IC1 is satisfied.

If the desired response from the system is a single output such as activating an external actuator, the correctness threshold for the replicated system has to be majority ($\Psi = \lceil n/2 \rceil$) as determined by a single device called the output *voter* [GPD84]. Fortunately, for most applications, the output voter can be constructed much more reliably than the processors or it can removed from the computing system and made part of the physical actuator mechanism.† The numerical result we present in this paper are for such systems (i.e., majority threshold).

---

† [SG84] describes an interesting example where voting is done through "physics" at the control surfaces of air foils.

2.3. Processor Failures

Each of the n processors in the system can be in two states: correct and faulty. Each processor starts out in the *correct state* where its behavior conforms to the specification encoded as a program. After some time, a processor *fails* and becomes faulty thereafter. Note that "correctness" and "faultiness" are only classifications of the internal state of a processor. A *faulty* processor may deviate from its specification in any arbitrary manner. This behavior includes choosing not to display its faultiness to the other processors until some later time or colluding with other faulty processors (through an adversary scheduler).

We assume that the times processors spend in the correct state before becoming faulty are independent and identically distributed exponential random variables with rate λ. In other words, each processor fails independently of all the others at a common constant failure rate. The exponential distribution assumption for failure times has been empirically justified for a large class of components, including electronic hardware, that do not "age" [SS82]. The independence of failures is typically achieved through physical and electrical isolation of the processors. Without loss of generality, we assume that $\lambda=1$ in the rest of the paper by scaling all time variables in our discussion by $1/\lambda$. Since $1/\lambda$ is the expected value of an exponential random variable with rate λ, the unit of time in our results is the mean-time-to-failure interval of a single processor. We do not consider the possibility of repairing faulty processors. We assume that communication in the system is through a perfectly reliable, fully connected network.

The standard "Byzantine Failure Assumption" where a system has to confront the maximum allowable number of faulty processors at the outset is unrealistic and leads to overly pessimistic designs. In real systems, there is usually a time when we have high confidence that the system has no faulty components. As the system operates, failures occur. Consequently, the usual malicious-processors-being-guided-by-an-adversary-scheduler assumption has to be augmented with a reasonable characterization of the time sequence of failures. Our characterization, which obliges processors to remain correct for independent and exponentially distributed random times, is one possibility. Undoubtably there are other characterizations that need to be explored [DM85].

3. Reliability of State Machine Ensembles

In this section we derive expressions for the reliability of a system consisting of n processors. As the origin of our time axis, we select any point before the beginning of the computation such that the system contains no faulty processors.

3.1. The Input Consensus Phase

We first consider the outcome of the input dissemination phase in a system that uses a consensus protocol $\Pi(\rho)$ where an authentication mechanism is used such that faulty processors cannot forge messages on behalf of correct ones [DH76, LSP80]. The parameter ρ is called the *resiliency* of the protocol if Π can achieve consensus in the presence of up to ρ faulty processors. It is well known that an authenticated consensus protocol can tolerate any number of faulty processors ($\rho < n$) [LSP80]. It is also well known that for a consensus protocol to be ρ-resilient, it has to execute for $\rho + 1$ rounds of message exchange in the worst case [DS83]. In this paper, we will be considering consensus protocol executions that are m rounds. Consequently, such executions will be at least $m-1$ resilient.

Let $\tau(m) = \alpha m$ denote the elapsed time for an m-round execution of protocol $\Pi(m-1)$ where α denotes the length of a round. We will characterize such executions by the sequence of positive random variables $Y_1 Y_2 \cdots Y_m$ where Y_i denotes the number of processors that failed *during* round i. The random variable $N_i = \sum_{j=1}^{i} Y_j$ represents the total number of processors that have failed *by the end of* round i.

The property that $\Pi(\rho)$ is a ρ-resilient deterministic consensus protocol can be formally stated as

$$\mathbf{P}\{\Pi(m-1) \textit{ achieves consensus in } m \textit{ rounds} \mid N_m < m\} = 1 .$$

In other words, given that fewer than m processors fail by the end of the mth round, $\Pi(m-1)$ will achieve consensus among n processors with certainty. Now we examine the case where there are m or more failures during an m-round execution (i.e., $N_m \geq m$).

The success of consensus in executions where the protocol resiliency is violated depends on the time sequence of the failures. Intuitively, if there are a sufficient number of failures early on in an execution, the faulty processors can exhibit their faultiness at the most inopportune moments so as to prevent consensus. For example, in an m-round execution where $Y_1 = m$, consensus is impossible as dictated by the lower bound results [DS83, Hadz84]. On the other hand, if failures occur sufficiently late in the execution, there will be a reasonable chance for consensus despite the fact that resiliency may be violated. What we need to do is obtain a characterization of those executions that permit consensus for any N_m.

Definition 1. *(Minimal Lean Prefix):* The first k rounds of an m-round execution $Y_1 Y_2 \cdots Y_m$, denoted $[1, k]$, is called a *length k minimal lean prefix* iff

(i) $N_k < k$ and

(ii) $N_{k-1} \geq k-1$.

This definition formalizes our intuition above that desired "not too many failures early on."

Definition 2. *(Lean Prefix):* An m-round execution *has a lean prefix*, denoted $LP(m)$, iff there exists some $k \leq m$ such that $[1, k]$ is a minimum lean prefix.

Lemma 1. In a distributed system with n processors, consensus can be achieved in an m-round execution iff the execution has a lean prefix.

Proof: For brevity, we give an outline of the proof here. The details can be found in [Baba86]. We consider the "only if" case first.

Consider an m-round execution that does not have a lean prefix. For each round $1 \leq i \leq m$, it must be that $N_i \geq i$. In other words, at any point in the execution, there is an abundance of faulty processors. In particular, for any round in the execution, the faulty processors can behave such that a different processor exhibits faulty behavior in every round. Such executions characterize precisely the "serial faultiness" behavior that is at the heart of the lower bound proof of Dolev and Strong for the time complexity of agreement protocols with authentication [DS]. Therefore, since agreement cannot be achieved in such an execution, consensus cannot be achieved either.

Now we consider the "if" case for the lemma. If $N_m < m$, the execution has a lean prefix by definition. Under this condition, we know that $\Pi(m-1)$ achieves consensus. The remaining case is $N_m = h \geq m$ and the execution has a lean prefix. Let f be the length of the minimal lean prefix for this execution. In other words, $N_f = f-1$, round f is failure free and $h-f+1$ additional failures occur in rounds $f+1$ through m. Consider a consensus protocol that consists of n concurrent "early stopping" (eventual) agreement protocols [DRS82]. Each processor initiates one of the executions to disseminate its local input value. In addition, each processor participates in $n-1$ other agreement protocols initiated by the other processors. Now, given that round f terminates a minimal lean prefix in this execution, each agreement protocol will encounter a round $i \leq f$ such that the number of actual failures during the i rounds is less than i. This is precisely the characterization for the time at which the last processor agrees on the value being disseminated in an early stopping protocol† [DRS82, Hadz84].

Once a correct processor agrees on the value being disseminated by another processor, it stops participating in that protocol but continues to participate in all others for which it has not yet agreed. Since the longest that any of the n early stopping protocols will run is f rounds, by round $f+1$ all of the correct processors will have achieved consensus on the n input values. During rounds $f+1$ through m the correct processors remain silent and ignore any messages they may receive from other (faulty) processors. Thus waiting it out until round m serves to synchronize the correct processors with respect to the termination of the protocol. Finally, note that the vector **W** obtained by all of the correct processors at round f continues to satisfy the requirements for consensus at round m. The $h-f+1$ additional processors that fail by round m already have their values committed in the consensus vector and are immune to further requirements by round m.

□

Next we derive the probability with which an m-round execution achieves consensus. The following probability distribution will be important in our derivations:

Definition 3. *(Binomial Distribution):* If a trial has probability p of succeeding independent of all others, the probability of obtaining exactly i successes in n trials is given by

$$Bin(n,i,p) = \binom{n}{i} p^i (1-p)^{n-i}, \quad 0 \le i \le n.$$

We state without proof the following result. The proof can be found in [Baba86].

Lemma 2. In a system with n processors, an m-round consensus protocol execution during which f processors fail has a lean prefix with probability

$$\Lambda(m,f) = \mathbf{P}\{LP(m) \mid N_m = f\}$$

$$= \sum_{k=1}^{m} \sum_{\vec{Y} \in \mathbf{Y}} \frac{f!}{y_1! y_2! \cdots y_{k-1}! (f-k+1)!}$$

$$\left[\frac{1-e^{-\alpha(m-k)}}{1-e^{-\alpha m}} \right]^f \left[\frac{1-e^{-\alpha}}{1-e^{-\alpha(m-k)}} \right]^{k-1} \frac{e^{-\alpha k f}}{e^{-\alpha(k y_1 + (k-1) y_2 + \cdots + 2 y_{k-1})}}$$

where
$\vec{Y} = (Y_1 = y_1, Y_2 = y_2, \cdots, Y_m = y_m)$ and

---

† More precisely, the stopping time of an early stopping protocol is $f+2$ rounds given that the execution encountered f failures. By requiring only $f+1$ rounds to stop, we are be-

$\mathbf{Y} = \{\vec{Y}: N_{k-1} = k-1,\ y_k = 0,\ N_m = f\}$
is the set of all m-round executions with f failures and a length k minimal lean prefix.

Theorem 1. In a system with n processors, an m-round consensus protocol achieves consensus with probability

$$\Gamma(m) = \sum_{i=0}^{m-1} Bin(n, i, 1-e^{-\alpha m}) + \sum_{f=m}^{n-1} \Lambda(m, f) Bin(n, f, 1-e^{-\alpha m}).$$

Proof: By Lemma 1, the probability of consensus in an m-round execution is equal to the probability that the execution has a lean prefix

$$\Gamma(m) = \mathbf{P}\{LP(m)\}.$$

Conditioning on the number of failures during the execution, we obtain

$$\mathbf{P}\{LP(m)\} = \mathbf{P}\{N_m < m\} + \sum_{f=m}^{n-1} \mathbf{P}\{LP(m) \mid N_m = f\} \mathbf{P}\{N_m = f\}$$

since the probability of a lean prefix in m rounds given $N_m < m$ is one. The number of processors to fail during m rounds has a binomial distribution with the probability of "success" (corresponding to a processor having failed by the end of m rounds) given as $1-e^{-\alpha m}$. Substituting the binomial distribution expressions and the result of Lemma 2 yields the desired result.

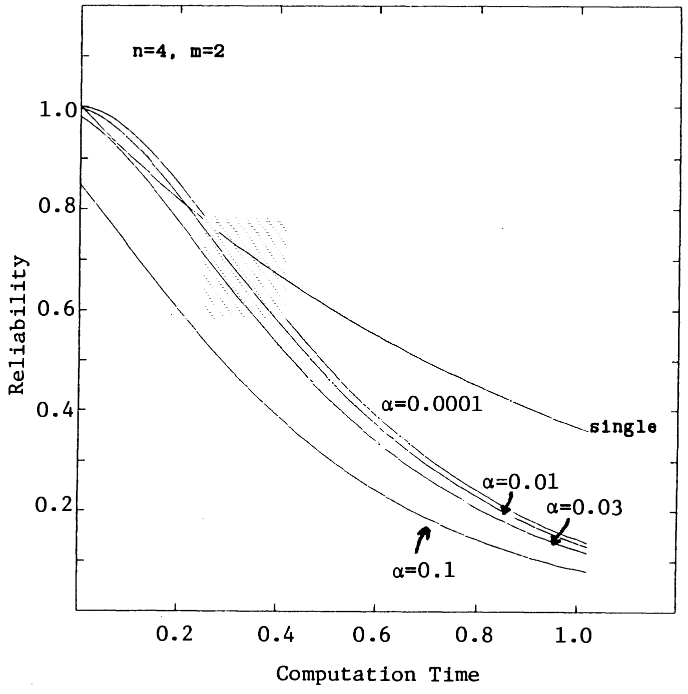
□

3.2. The Computation Phase

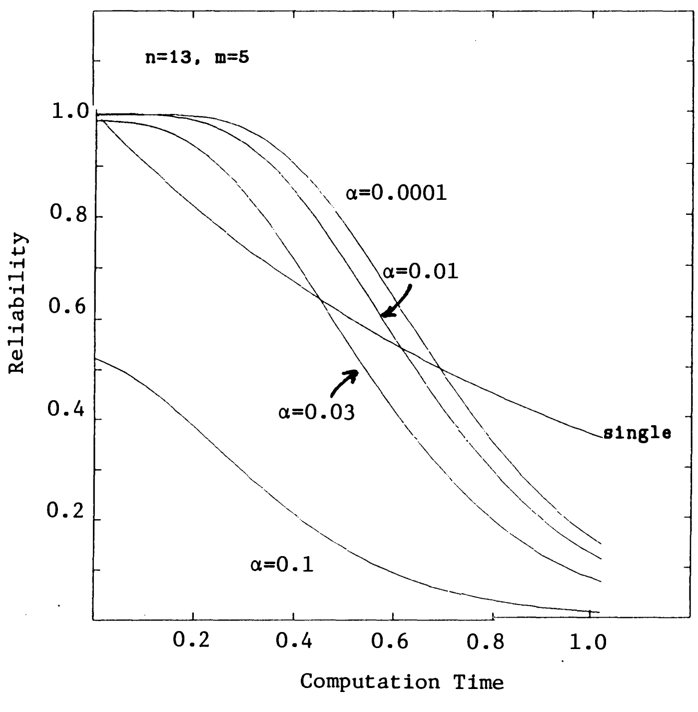
Let us now examine the correctness of the system after computing for T units towards the application function F. For the output to be correct, two conditions must be satisfied: (i) the input consensus protocol must have succeeded, and (ii) at least Ψ processors must remain correct by the end of the computation. If the input data is disseminated using an m-round consensus protocol, these conditions are equivalent to there being a lean prefix during the m rounds and the number of failures by time $\alpha m + T$ not exceeding $n - \Psi$.

---

ing slightly optimistic in our analysis.



(a)



(b)

Fig. 1. Reliability of state machine ensembles.

Theorem 2: At computation time T, the reliability of a system with n processors that uses an m-round consensus protocol for input data dissemination is given by

$$R(n,m,T) = \sum_{f=0}^{n-\Psi} \Lambda(m,f) Bin(n,f,1-e^{-\alpha m}) \sum_{i=0}^{n-\Psi-f} Bin(n-f,i,1-e^{-\alpha T})$$

Proof: The desired expression is obtained by conditioning first on N_m, the number of failures during consensus, and then on the number of failures during the computation. Note that if $N_m > n-\Psi$, the system cannot be correct even if consensus can be achieved. As we noted earlier, the number of processors to fail by the end of the consensus protocol has a binomial distribution. Given that $N_m = f$, the distribution of the number of failures during the computation is also binomial with parameters $n-f$ and $1-e^{-\alpha T}$. This is a consequence of the processor failure times being independent and exponentially distributed.

□

Figure 1 is the reliability of systems with various levels of replication and round lengths. Although all of the numerical results we present in this paper are for systems that require a majority threshold for correctness, our conclusions apply to all systems with correctness thresholds that are linear functions of n of the form $\Psi = \psi n$ for some positive $\psi < 1$. Recall that the unit of time in our results is the mean-time-to-failure of a single processor $(1/\lambda)$. With today's technology, it is trivial to achieve mean-time-to-failure for processors in excess of 100 hours. Typically, processor and communication network speeds allow rounds to be realized in at most several seconds. Consequently, realistic normalized values for α are of the order 10^{-5}. Our results also include larger values for the round length simply to explore as large a design space as possible.

While the reliability of the single processor (denoted "single" in the figures) is always one at computation time zero, the state machine ensembles can have reliability less than one even at the outset due to the input consensus failing. Depending on the system parameters and the length of computation, there are regions where the non-replicated system ("single") has greater reliability than the replicated design. Furthermore, after a sufficiently (unrealistically) long computation, the single processor option has uniformly higher reliability than any replicated alternative. As the system size grows larger, the computation time that delineates highly reliable systems from highly unreliable systems becomes more sharply defined. This behavior is a consequence of the following asymptotic results. Let $\overline{T}$ denote a positive random variable such that $\overline{T} = t$ if the replicated system is incorrect by time t.

Proposition 1. As the round length $\alpha \to 0$, the computation time by which a replicated system with n processors and correctness threshold Ψ becomes incorrect has expectation

$$\mathbf{E}(\bar{T}) = \sum_{j=0}^{n-\Psi} \frac{1}{(n-j)} .$$

As the replication level of the system increases, the expected failure time has the limit

$$\mathbf{E}(\bar{T}) \to \ln\left[\frac{n}{\Psi-1}\right] \quad \text{as } n \to \infty .$$

Proof: Let the positive random variable X_i denote the time of the ith processor failure. In other words, $X_i = t$ if there are $i-1$ failures in the interval up to t and there is a failure at time t. The random variables $X_1 X_2 \cdots X_n$ are called the *order statistics* of the n independent exponential random variables denoting the time to failure of the processors [Fell71]. In the limiting case where the input consensus is instantaneous, the time when the system becomes incorrect has the distribution of $X_{n-\Psi+1}$. The result follows from this observation.

□

In a system where the correctness threshold is majority, the above limit becomes $\ln(2) = 0.693$ as $n \to \infty$. We note that some of the above observations were also made by Tay [Tay84].

As the replication level of the system increases, not only does the expected time for the system to become incorrect approach a constant, the transition time from a correct system to an incorrect one becomes more deterministic.

Proposition 2. As the round length $\alpha \to 0$, the computation time by which a replicated system with n processors and correctness threshold Ψ becomes incorrect has variance

$$\mathrm{Var}(\bar{T}) = \sum_{j=0}^{n-\Psi} \frac{1}{(n-j)^2} .$$

As the replication level of the system increases, the variance has the limit

$$\mathrm{Var}(\bar{T}) \to \frac{n-\Psi+1}{n(\Psi-1)} \quad \text{as } n \to \infty .$$

Proof: Follows directly from the observation that the distribution of $X_{i+1}-X_i$ has an exponential distribution with rate $(n-i)$. Furthermore, the pairwise differences of the n order statistics are independent [Fell71].

□

Note that in systems where the correctness threshold is $\Psi = \psi n$, the above limit is zero. For example, in a system with majority threshold, for sufficiently high levels of replication, the system will be highly reliable for application computation times up to 0.693 units and highly unreliable thereafter. This observation will be useful when we consider choices for times to perform fault detection.

3.3. Selecting the Running Time of Consensus Protocols

In Section 3.1 we observed that an m-round consensus protocol execution is at least $m-1$, and perhaps much more, resilient depending on the pattern of processor failures. In this section we will explore the tradeoffs that are involved in selecting the running time of the input consensus protocol.

All other factors being equal, an $(m+1)$-round execution has a higher probability of achieving consensus than an m-round execution. This follows from the observation that the probability of an execution having a lean prefix is a monotone increasing function of the execution length. What is of interest to us is the rate of this increase. To gain some insight into this question, we have evaluated the expression for $\Gamma(m)$ given in Theorem 1 as a function of m for a system with 13 processors and various round lengths (α). The results are displayed in Figure 2. Note that for small (realistic) values of α, a 2 to 3-round execution defines the point diminishing returns— the system achieves consensus with high probability for this execution length and any additional rounds of execution result in negligible gain. However, for large values of α, each additional round of execution increases the probability of consensus by a significant amount.

Next we investigate the effect of input consensus protocol execution time on the reliability of a state machine ensemble. Although the consensus protocol itself always benefits from additional rounds of execution, this benefit is gained at the expense of having fewer correct processors available for the application computation. Thus, the increase in the probability of the consensus phase succeeding may be offset by the decrease in the probability of the system achieving the correctness threshold. In Figure 3 we plot the reliability of the state machine ensemble, as given by $R(n,m,T)$ of Theorem 2, as a function of m and for various computation times T. For this 13-processor system, when the computation time is short with respect to the

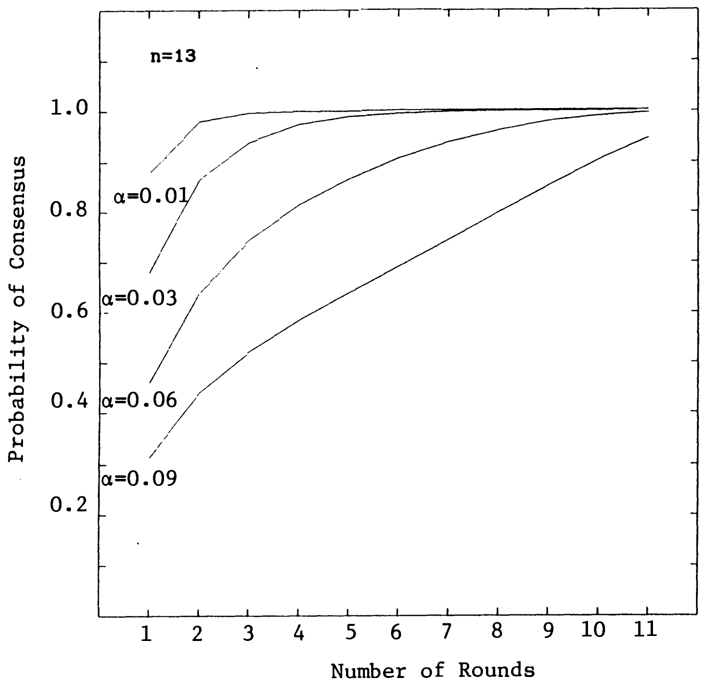


Fig. 2. Probability of consensus versus protocol execution length.

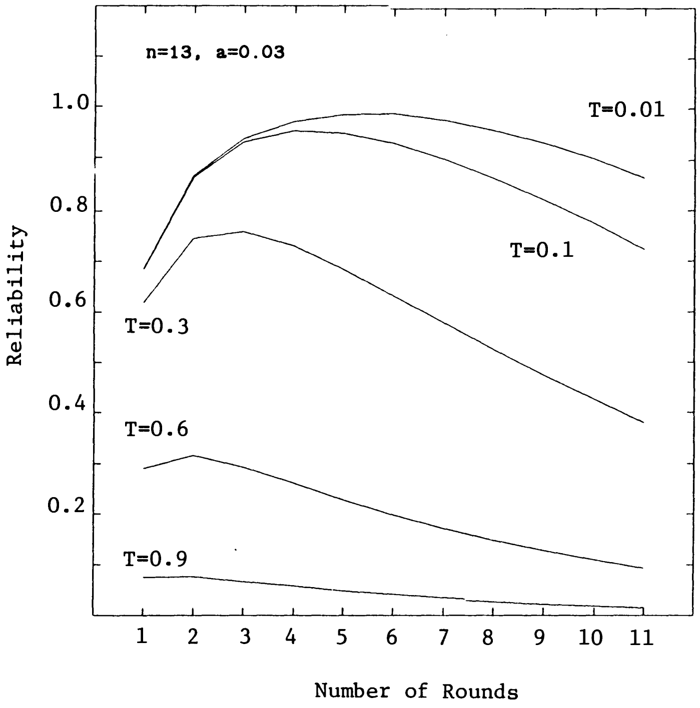


Fig. 3. System reliability versus protocol execution length.

consensus phase, the system reliability increases for up to about 5 rounds of consensus execution and then starts to decrease. For computation times that are longer, the consensus protocol execution length that maximizes system reliability becomes shorter. In fact, for sufficiently long computations, the reliability is a monotone decreasing function of m suggesting that the system is better off not wasting its time with consensus and commencing the computation immediately after a single round of input data exchange.

3.4. Randomized Consensus Protocols

A large number of *randomized* consensus protocols have been proposed that permit nondeterministic steps in the protocol computation [BenO83, Rabi83, CC85, Brac85]. Randomized protocols are often touted as fast alternatives to deterministic protocols.

Consider a system with randomized consensus protocol resilient to ρ faulty processors. In such a system

$$\mathbf{P}\{\textit{consensus is achieved after m rounds} \mid N_m < \rho\} = (1-p)^{m-1}p, \quad m \geq 1$$

for some probability p. In other words, provided that the resiliency requirement has not been violated (more than ρ faulty processors), the protocol has a fixed probability p of achieving agreement in the current round. Note that such a protocol will reach agreement with probability p even after the first round. Analogous to Theorem 1, we are interested in the unconditional probability that a randomized protocol achieves consensus by a certain round.

Theorem 3: In a system with n processors, a randomized consensus protocol achieves consensus after m rounds with probability

$$\Gamma(\rho, m) = \sum_{i=1}^{m} (1-p)^{i-1} p \sum_{f=0}^{\rho} Bin(n, f, 1-e^{-\alpha m}).$$

Proof: Follows after conditioning on the number of processors that fail during m rounds.

□

As a concrete example, consider the randomized protocol presented in [CC85]. The heart of the protocol is a distributed, fault-tolerant realization of a "fair coin tossing" procedure. The protocol does not require authentication and can tolerate up to $n/3$ faulty processors. In the presence of $n/3$ or fewer faulty processors, each attempt at tossing a coin succeeds with probability (at least) 1/3. The protocol requires two rounds to implement each coin tossing epoch and guarantees consensus two rounds after the first successful toss. In Figure 4 we illustrate $\Gamma(\rho, m)$ for this system with 13

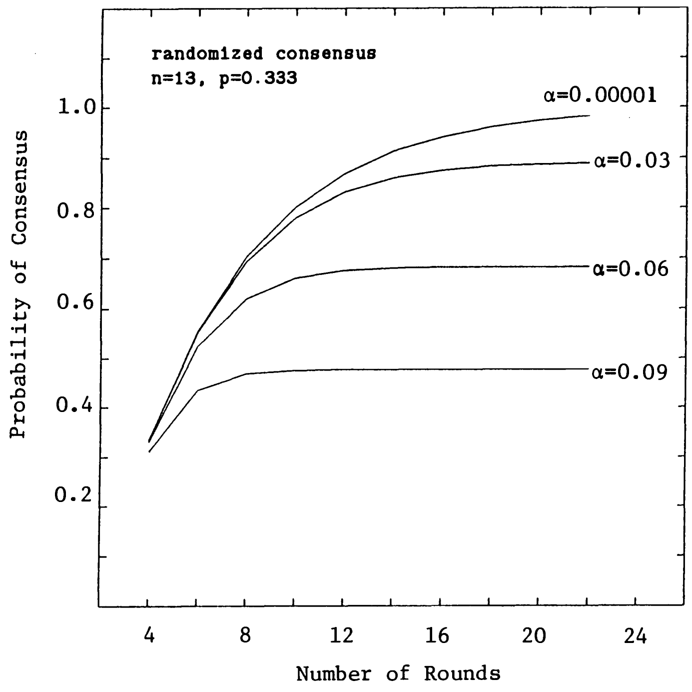


Fig. 4. Probability that randomized protocol achieves consensus.

processors. Note that this protocol has substantially different behavior than the deterministic authenticated protocol studied earlier. First, the probability of consensus starts of small (1/3) and increases gradually. Second, for large enough round lengths, the probability of consensus does not tend to one for arbitrarily long executions. This is due to the increased likelihood of the resiliency requirement being violated before the coin toss procedure ever succeeds. Given the stochastic model for processor failures, a finer analysis would be required to characterize those executions for which $N_m \geq \rho$ yet the distributed coin tossing procedure remains operational.

3.5. Fault Detection

In the absence of a "repair and reintegration" facility for faulty processors, a replicated system becomes highly unreliable in the long run since failures accumulate. There comes a time such that it takes only one additional processor failure (the $(n-\Psi+1)st$) that makes a correct system incorrect. In this section, we consider the effectiveness of fault detection mechanisms that can identify and remove from the system those processors that are faulty. It will be convenient for us to redefine the the correctness threshold as the smallest fraction of the total number of processors in the system that have to be correct. Thus, given that at least Ψ of the initial n processors need to be correct, we will assume that the system will remain correct as long as the ratio of correct processors to total processors is greater than Ψ / n.

We first investigate the possibility of detecting faulty processors as a byproduct of the consensus protocol. Unfortunately, standard consensus protocols do not guarantee this property. However, most consensus protocols at least detect certain types of processor failures that occur during the first round of their execution.

As an upper bound for the effectiveness of this technique, we recompute the reliability of the system depicted in Figure 1(b) assuming that the input consensus protocol can detect *all* failures that occur during its execution. The desired reliability expression is identical to that of Theorem 2 except for the limit of the second summation. Whereas before the number of additional failures during the computation was bounded by $n-f-\Psi$, now it can be as

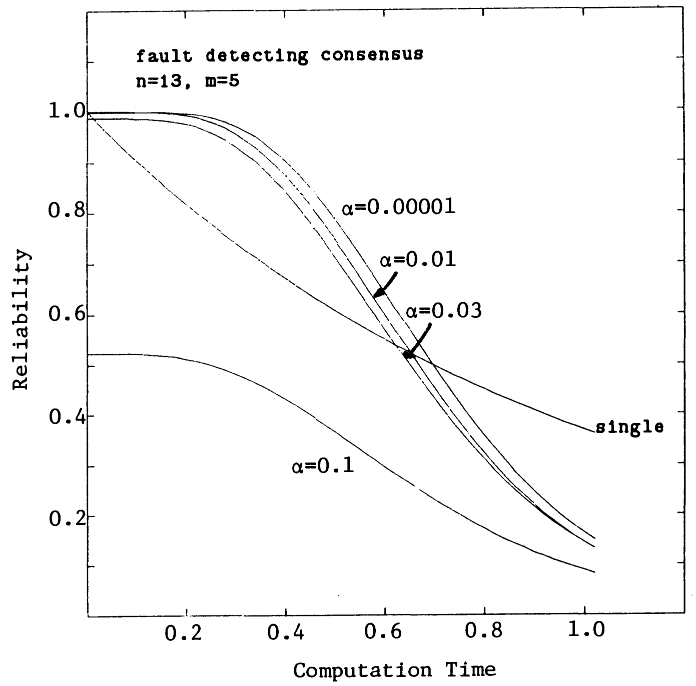


Fig. 5. Fault detecting consensus protocol.

large as $\lfloor (n-f)(1-\Psi/n) \rfloor$ without violating the correctness threshold. Not surprisingly, the system in Figure 5 has higher reliability than its counterpart in Figure 1(b). The increase in reliability is most marked for larger round length where the expected number of processors to fail during the consensus phase is larger. Obviously, for consensus protocols that can detect some fraction of the processors that fail during their execution, the results will be bounded by the two extremes depicted in Figures 1(b) and 5.

Next we study system reliability in the presence of a mechanism that can detect and remove from the system all processors that have failed by time θ after the the beginning of the application computation.

Theorem 4. At computation time T, the reliability of a system with n processors that uses an m-round consensus protocol for input data dissemination and fault detection at time θ is given by

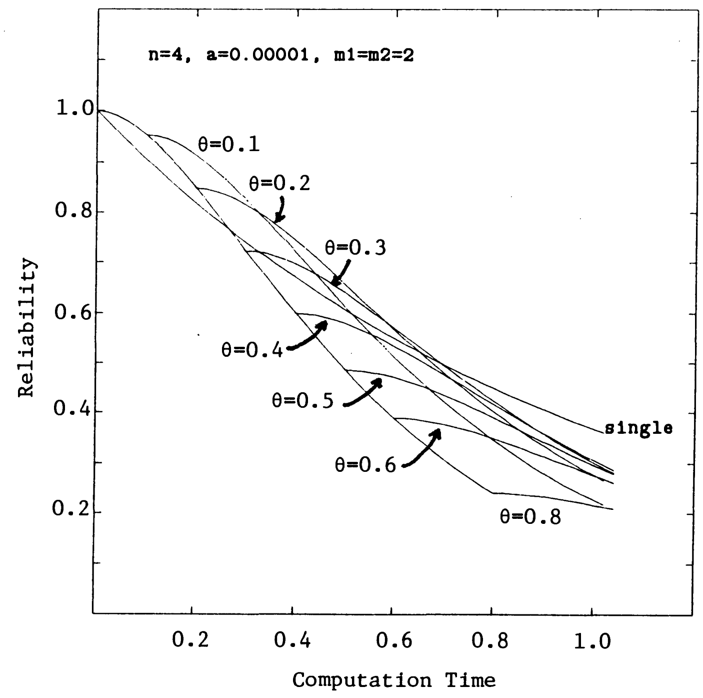
$$R(n,m,\theta,T) = \sum_{f=0}^{n-\Psi} \Lambda(m,f) Bin(n,f,1-e^{-\alpha m}) \sum_{i=0}^{n-\Psi-f} Bin(n-f,i,1-e^{-\alpha\theta})$$

$$\sum_{j=0}^{(n-f-i)(1-\Psi/n)} Bin(n-f-i,j,1-e^{-\alpha(T-\theta)})$$

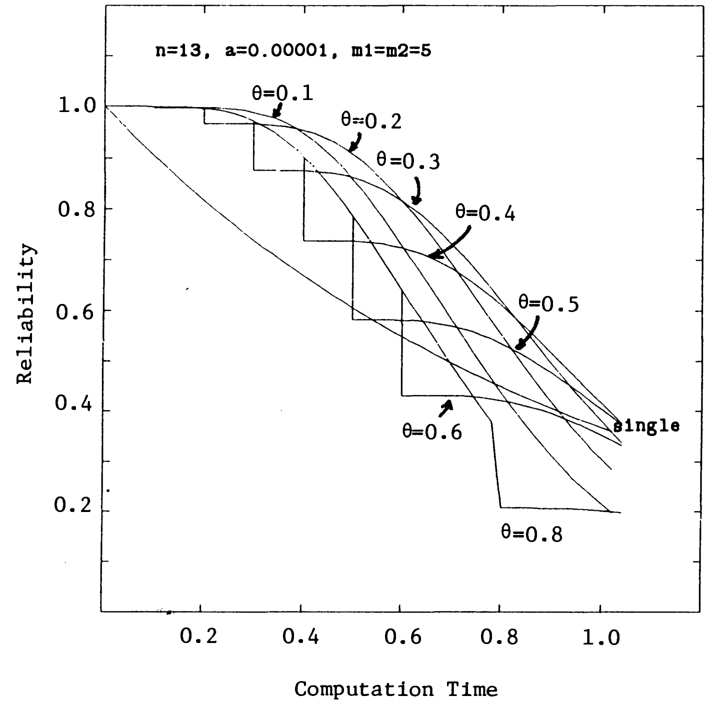
Proof: The proof is very similar to that of Theorem 2 except we condition on the number of failures during the θ and $T-\theta$ segments of the computation separately.

□

One possible implementation of a fault detector is to augment the computation such that processors execute a consensus protocol on some representation of their partial result after θ time units. If by the time the fault detector is invoked, the number of faulty processors is at a minority, then they can be identified by all of the correct processors. For applications that are structured as sequences of tasks that compute on the results of previous tasks, the consensus value could be such an intermediate result. In general, the processors could use the location and value of each memory update operation for consensus [Schn84]. If the frequency of memory updates is high, processors could compute a hash function on their state only after a fixed number of iterations of a loop or after the passage of some length of time to generate a suitably compact consensus value [GPD84]. Clearly, for this proposal to be effective, the faulty processors must expose their faultiness either in the computation of the partial result or during the consensus protocol execution. If faulty processors can effectively hide their faultiness from the correct ones until the very end of the computation, then no mechanism can serve as a fault detector.



(a)



(b)

Fig. 6. The effect of fault detection through partial result consensus.

Let $m1$ denote the number of rounds that the input consensus protocol executes and $m2$ the number of rounds that the fault detector consensus protocol executes. Figure 6 illustrates the effectiveness of this mechanism on the reliability of systems with 4 and 13 processors. Note that these are the sam systems depicted in Figure 1 without fault detection. In both systems of Figure 6, the two consensus protocols execute for the same number of rounds. We notice that at the time the fault detection takes place, the system ceases to suffer rapid loss of reliability. Instead, the current reliability is maintained almost without loss for some time and then the system resumes to degrade. Note that in the system with 13 processors, the cases where fault detection takes place late in the computation result in a significant loss in reliability. This is due to keeping the fault detector consensus protocol execution time fixed at 5 rounds. Intuitively, the expected number of faulty processors in the system by the time the fault detector is executed increases with increasing θ. Consequently, the probability of the fault detector consensus succeeding decreases. Clearly, $m2$ should be increased as θ increases.

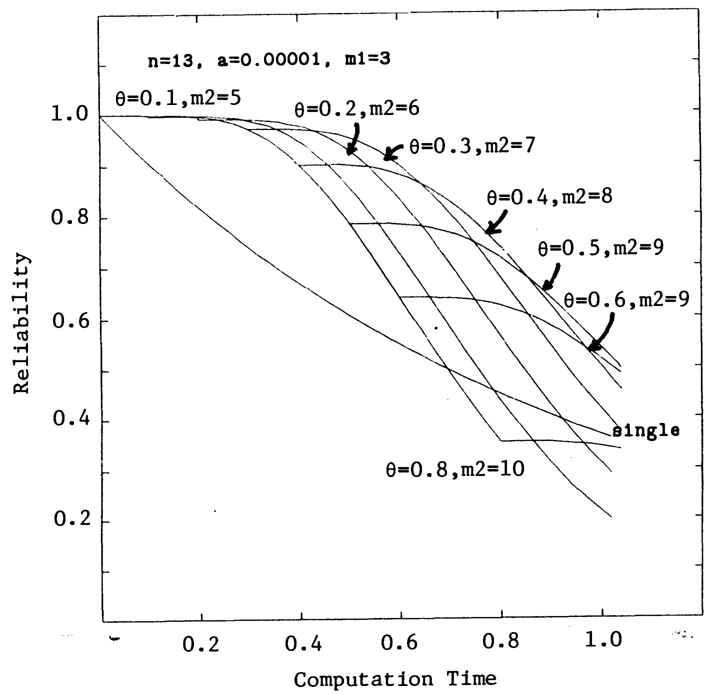


Fig. 7. The effect of variable execution length fault detection.

Given n processors that fail according to our stochastic model, the expected number of faulty processors by time θ is $n(1-e^{-\theta})$. In Figure 7 we recompute the reliability of the system depicted in Figure 6(b) where $m2=\lceil n(1-e^{-\theta})\rceil+3$. Now the effect of the fault detector is much more positive in that the system remain uniformly more reliable with fault detection than without.

4. Lack of Authentication

Consider a fault-tolerant system design constrained to not use authentication. This system is different from the ones studied so far in two regards. First, of the n processors, at most $n/3$ could be faulty for consensus to be achievable in any number of rounds [PSL80]. Second, the best known consensus protocol for this environment requires an execution time of $\rho+1$ phases to guarantee ρ-resiliency where each phase involves two rounds of message exchange [ST84].

We can reason about the unauthenticated system by considering an authenticated system. This follows from the manner in which the unauthenticated protocol of [ST84] was developed. Starting with an authenticated consensus protocol, Srikanth and Toueg replace each authenticated phase with two unauthenticated rounds that preserve the semantics of authentication. Consider an m-phase execution of the unauthenticated system. If there is minimal lean prefix of length at most $n/3$, then the execution will achieve consensus. If, on the other hand, the minimal lean prefix is longer than $n/3$ rounds, there must have been more than $n/3$ failures during the execution. Without a finer characterization of the failure sequence, no guarantee can be made towards consensus. Which failure sequences longer than $n/3$ processors permit consensus remains an open question.

5. Conclusions

We have presented an analytical evaluation of the reliability of a general fault-tolerant distributed computing structure based on replicated state machines. While we made no assumptions about the behavior of processors once they failed, we modeled the times at which they fail as a stochastic process with empirical foundations. We claim that this is a much more realistic framework to study fault tolerance than the typical one where the system is assumed to contain up to some fixed number of faulty processors *at the outset*. Typically, fault-tolerant distributed systems are designed with the expectation that there exists some time when we have very high confidence that all of the components are correct. Then, the goal of fault tolerance is to provide protection against failures that may happen some time in the future with respect to this reference time.

Within this framework, we have explored the reliability of systems as a function of the various design parameters. Our results include some counterintuitive properties about the selection of the running times of consensus protocols and about the behavior of randomized protocols. Use of consensus protocols at intermediate steps in the application computation is an effective technique to detect and remove faulty processors from the system. We explored the choices for times to perform fault detection.

Our analysis for the computation augmented with a single execution of a fault detector can be extended in an obvious way to consider multiple executions. If fault detection is performed infrequently, a large enough number of faulty processors may accumulate and the system reliability will be low. If it is performed very frequently, the system will contain mostly correct processors, however, considerable time will be devoted to fault detection rather than computing and again system reliability will suffer. The question of determining the optimal frequency for fault detection during the computation is closely related to that of inserting checkpoints in a sequence of tasks [TB84]. This is an obvious topic for further investigation.

Our results are a first step towards a methodology for designing fault-tolerant distributed system that satisfy certain reliability specifications. For example, a specification of the form "$R(n, m, \theta, T) \geq P_{max}$ for all $T \leq T_{max}$" sets a lower bound on the system reliability during a mission that lasts T_{max} units of time. By generating results similar to Figure 7, a designer can easily evaluate various system design alternatives with respect to the reliability specification.

Acknowledgments

I am grateful to Pat Stephenson for many discussions of this material and also for his help in generating the figures. The presentation has benefited from many valuable suggestions by Fred Schneider.

References

Baba86 Ö. Babaoğlu, Stopping times of consensus protocols: a probabilistic analysis, Department of Computer Science, Cornell University, Ithaca, New York, in preparation.

BenO85 M. Ben-Or, Fast synchronous Byzantine Agreement, *Proc. 4th ACM Symposium on Principles of Distributed Computing*, Minaki, Ontario, Canada, pp. 149-151, August 1985.

Brac85 G. Bracha, An $O(\lg n)$ expected rounds randomized Byzantine Generals protocol, *Proc. 17th ACM Symposium on Theory of Computing*, Providence, Rhode Island, pp. 316-326, May 1985.

CASD85 F. Cristian, H. Aghili, R. Strong and D. Dolev, Atomic broadcasts: From simple message diffusion to Byzantine Agreement, *Proc. 15th Symposium on Fault Tolerant Computing*, Ann Arbor, Michigan, pp. 200-206, June 1985.

CC85 B. Chor and B. A. Coan, A simple and efficient randomized Byzantine Agreement algorithm, *IEEE Trans. Software Eng.*, vol. SE-11, pp. 531-538, June 1985.

DH76 W. Diffie and M. Hellman, New directions in cryptography, *IEEE Trans. on Inf. Theory*, vol. IT-22, pp. 644-654, 1976.

DM85 C. Dwork and Y. Moses, Knowledge and common knowledge in a Byzantine Environment I: Crash failures, Private Communication (January 1986).

DRS82 D. Dolev, R Reischuck, and H. R. Strong, 'Eventual' is earlier than 'immediate', *Proc. 23rd Symposium on Foundations of Computer Science*, Chicago, Illinois, pp. 196-203, November 1982.

DS83 D. Dolev and H. R. Strong, Authenticated algorithms for Byzantine Agreement, *SIAM J. Comput.*, vol. 12, no. 4, pp. 656-666, November 1983.

Fell71 W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley & Sons, New York (1971).

Fisc83 M. J. Fischer, The consensus problem in unreliable distributed systems (A Brief Survey), Tech. Rep. YALEU-DCS-RR-273, Dept. of Computer Science, Yale University, New Haven, Connecticut, June 1983.

FL82 M. J. Fischer and N. Lynch, A lower bound for the time to assure interactive consistency. *Inform. Proc. Letters 14*, 4, pp. 183-186, April 1982.

GPD84 H. Garcia-Molina, F. Pittelli and S. Davidson, Applications of Byzantine Agreement in database systems, Tech. Rep. TR 316, Princeton University, Princeton, New Jersey, June 1984.

Hadz84 V. Hadzilacos, Issues of fault tolerance in concurrent computations. Ph.D Thesis, Tech. Rep. TR-11-84, Aiken Computation Laboratory, Harvard University, Cambridge, Mass., June, 1984.

Lamp78 L. Lamport, Time, clocks and the ordering of events in a distributed system, *Communications of the ACM*, vol. 21, no. 7, pp. 558-565, July 1978.

LSP82 L. Lamport, R. Shostak, and M. Pease, The Byzantine Generals problem, *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382-401, July 1982.

Lamp84 L. Lamport, Using time instead of timeout for fault-tolerant distributed systems, *ACM Transactions on Programming Languages and Systems*, vol. 6, no. 2, pp. 254-280, April 1984.

LM85 L. Lamport and P. M. Melliar-Smith, Synchronizing clocks in the presence of faults, *Journal of the ACM*, vol. 32, no. 1, pp. 52-78, January 1985.

MS85 S. R. Mahaney and F. B. Schneider, Inexact agreement: Accuracy, precision, and graceful degradation, *Proc. 4th ACM Symposium on Principles of Distributed Computing*, Minaki, Ontario, Canada, pp. 237-249, August 1985.

Peas80 M. Pease, R. Shostak and L. Lamport, Reaching agreement in the presence of faults, *Journal of the ACM*, vol. 27, no. 2, pp. 228-234, April 1980.

Rabi83 M. Rabin, Randomized Byzantine Generals, *Proc. 24th Symposium on Foundations of Computer Science*, Tucson, Arizona, pp. 403-409, November 1983.

Schn84 F. B. Schneider, Byzantine generals in action, *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 145-154, May 1984.

SD83 H.R. Strong and D. Dolev, Byzantine agreement. In *Digest of Papers, Spring Compcon 83*, San Francisco, California, pp. 77-81, March 1983.

SL85 F. B. Schneider and L. Lamport, Paradigms for distributed programs, In *Distributed Systems: Methods and Tools for Specification*, Paul, M. and Siegert H.J. (Eds.), Springer-Verlag Lecture Notes in Computer Science Vol. 190.

SS82 D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reli-*

able System Design, Digital Press, Belford, Mass. (1982).

SG84 A. Z. Spector and D. Gifford, Case study: the space shuttle primary computer system, *Communications of the ACM*, vol. 27, no. 9, pp. 874-900, September 1984.

Stan84 J. A. Stankovic, A Perspective on distributed computer systems, *IEEE Trans. Comp.*, vol. C-33, no 12, pp. 1102-1115, December 1984.

ST84 T.K. Srikanth and S. Toueg, Simulating authenticated broadcasts to derive simple fault-tolerant algorithms, Tech. Rep. 84-623, Department of Computer Science, Cornell University, Ithaca, New York, July 1984.

Tay84 Y. C. Tay, The reliability of (k,n)-resilient distributed systems, *Proc. 4th Symposium on Reliability in Distributed Software and Database Systems.* Silver Spring, Maryland, pp. 119-122, October, 1984.

TB84 S. Toueg and Ö. Babaoğlu, On the optimum checkpoint selection problem, *SIAM Journal of Computing*, vol. 13, no 3, pp.
g,,ilie