On the Reliability of Consensus-Based Fault-Tolerant Distributed Computing Systems

ÖZALP BABAOĞLU
Cornell University

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.

Categories and Subject Descriptors: B.1.3 [**Control Structures and Microprogramming**]: Control Structure Reliability, Testing and Fault-Tolerance—*redundant design*; B.3.4 [**Memory Structures**]: Reliability, Testing and Fault-Tolerance—*redundant design*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Computer Systems Organization**]: Performance of Systems—*reliability, availability, and serviceability*

General Terms: Design, Performance, Reliability

Additional Key Words and Phrases: Byzantine Agreement, deterministic and randomized protocols, distributed consensus, fault detection, interactive consistency, stochastic failures

1. INTRODUCTION

The potential for highly reliable computing is an often-cited benefit of distributed systems [28]. The collection of autonomous processors in such systems provides a natural environment for replicating computations so that failures can be 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*[1] protocols have been proposed as appropriate mechanisms for implementing such coordination [20, 22]. The question of structuring highly reliable

[1] There are many variants of the *consensus* problem. Collectively, they are often referred to as *Byzantine Agreement*. See [13] and [30] for two brief surveys.

Partial support for this work was provided by the National Science Foundation under grants DCR-86-01864 and MCS 82-10356 and AT&T under a Foundation Grant.
Author's address: Department of Computer Science, Cornell University, Ithaca, NY, 14853-7501.

distributed computing systems based on this paradigm has attracted some interest [7, 15, 18, 25].

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 the objectives 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[2], 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 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 behavior—the system is highly reliable until a well-defined time, and highly unreliable thereafter. In Section 3.3 we consider the following 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. Section 3.4 examines the implications of using a randomized consensus protocol rather than a deterministic one. In Section 3.5 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

[2] Some protocols, such as those in [19] and [21] exhibit "graceful degradation" up to a certain number of faulty processors.

based on this data, and writes outputs to an external actuator. This structure for a computation has been called a *state machine* [17].

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 [15, 18, 25]. 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: $\mathbf{I} \rightarrow \mathbf{O}$ 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 the beginning of the computation (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 $v_i(t_i)$ denote the input value at processor i when it reads the sensor(s). In general, $v_i(t_i) \neq v_j(t_j)$ 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 possible 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 $F \cdot G$ 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 ([26], p. 7). 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 some 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 (the application program) can commence, processors must disseminate their local input values to each other. In the presence of failures, a *distributed consensus* protocol is what is required to accomplish this. Formally, let $v_1, v_2, \ldots, v_n$ denote the local input values where v_i is the input to processor i. Processors execute a protocol, at the conclusion of which each

processor i obtains an n-element vector $\mathbf{W}_i$ such that

IC1. If i and j are two correct processors, then $\mathbf{W}_i = \mathbf{W}_j = \mathbf{W} = (w_1, w_2, \ldots, w_n)$,
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 a faulty processor i, as long as each correct processor obtains the same value for it. The above formulation of the distributed consensus problem is also known as *interactive consistency* [22]. If the desired outcome of the protocol is a single value (rather than a vector), it is called the *agreement problem* [20]. Applying a decision function (such as $\mathbf{G}$) to the consensus vector provides one method to obtain a solution for the agreement problem given a solution to the above.

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 as determined by an output *voter* [15]. Fortunately, for most applications, the output voter can be constructed much more reliably than the processors or it can be removed from the computing system entirely and be made part of the physical actuator mechanism.[3] The numerical results we present in this paper are for such systems (i.e., majority threshold).

2.3 Processor Failures

In this section we define how processors fail and what they can do once they fail. 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 so-called Byzantine or malicious behavior of a faulty processor includes choosing not to display its faultiness to the other processors until some later time or colluding with the other faulty processors.

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" [26]. 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 with $1/\lambda$. 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.

[3] [27] describes an interesting example where voting is done through "physics" at the control surfaces of air foils.

The standard "Byzantine Failure Assumption" where a system has to confront the maximum allowable number of malicious processors at the outset is unrealistic and leads to overly pessimistic designs. In more realistic situations, the system starts out with some small number of potentially faulty components. As the system operates, additional failures occur. Consequently, a failure model must not only specify what faulty processors can and cannot do, but also characterize when they can become faulty. Our characterization, which obliges each processor to remain correct for some random length of time, is one possibility. Undoubtably there are other characterizations that need to be explored [11].

Even though our model imposes the above stochastic structure on the formation of the set of faulty processors, and is thus more realistic than the standard Byzantine model, it might still be perceived as being overly pessimistic since it allows Byzantine behavior at all. In support of permitting Byzantine behavior in this limited manner in our model, which might be called "probabilistic Byzantine," we put forward the following arguments:

—The "Byzantine assumption" is in fact a nonassumption—a design based on it does not exploit any presumed behavior of faulty components. In many life-critical applications, this is the only realistic option available to the designer—not because of any malice on the part of faulty components, but because of our inability to characterize what constitutes possible worst-case faulty behavior.

—Certain documented failures in large distributed systems have been catastrophic and could not have been avoided under any failure assumption more restrictive than Byzantine [6]. Cohn describes how, on two different occasions, a seemingly innocent hardware failure has disrupted the entire Arpanet for several hours. Due to the nature of the system and the routing algorithm, the corruption of a single bit in a critical field of a message packet caused all of the nodes to "collaborate" towards the escalation of the problem to catastrophic levels.

—By understanding the properties of designs based on the Byzantine fault model, we gain extremely valuable insight into what is attainable in other systems where faulty behavior can be restricted. Furthermore, such designs establish upper bounds for the various cost measures of a system. If these costs are comparable to those associated with more restrictive failure models, we may choose to incur the additional cost to get the additional coverage.

3. RELIABILITY OF STATE MACHINE ENSEMBLES

In this section we derive expressions for the reliability of a system consisting of n processors. We assume that communication in the system is through a perfectly reliable, fully connected network. Furthermore, we assume that the system is *synchronous*—processor relative speeds (equivalently, clocks) and message delivery times can be bounded such that the consensus problem indeed has a deterministic solution. In such a system, we find it convenient to describe distributed computations progressing in rounds. Informally, a *round* is some (fixed-length) time interval during which each processor can exchange messages with every

other processor in the system as well as perform some (fixed) amount of local computation.

3.1 The Input Consensus Phase

We first consider the outcome of the input dissemination phase in a system that uses a consensus protocol $\prod(\rho)$ where an authentication mechanism is used such that faulty processors cannot forge messages on behalf of correct ones [8, 10, 20]. The parameter ρ is called the *resiliency* of the protocol if $\prod$ 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 ($\rho < n$) [20]. It is also well known that for a consensus protocol to be ρ-resilient, it has to execute for $\rho + 1$ rounds in the worst case [10]. 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 $\prod(m - 1)$ where α denotes the length of a round. We will characterize such executions by the sequence of positive random variables $(Y_1, Y_2, \ldots, Y_m)$ 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. Without loss of generality, we assume that $N_0 = 0$.

Now, the property that $\prod(\rho)$ is a ρ-resilient deterministic consensus protocol can be formally stated as

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

In other words, given that fewer than m processors fail by the end of the mth round, $\prod(m - 1)$ 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 [10, 14, 16]. 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. (*Lean Prefix*): An m-round execution has a *lean prefix*, denoted $LP(m)$, if and only if there exists some $k \leq m$ such that $N_{k-1} = N_k = k - 1$.

This definition formalizes our intuitive notion of "not too many failures occurring early on." Clearly, every lean prefix terminates in a failure-free round.

As an example, consider the two 6-round executions (2, 1, 1, 0, 0, 4) and (4, 1, 1, 0, 0, 0). According to our definition, the first one has a length 5 lean prefix since round 5 is the first round where the total number of failures by that round

is less than the round number and round 5 is failure free. On the other hand, the second execution does not have a 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.*

Proofs of our technical results are presented in the Appendix.

This lemma is a structural characterization of those executions that permit consensus. Note that it is a distribution-free result in the sense that it is not dependent on the exponentially distributed failure times assumption of processors. Also note that its proof relies on the Byzantine behavior of processors once they fail. With more restrictive failure models (e.g., processors simply halt when they become faulty), there may be many nonlean executions that also permit consensus to be achieved [2].

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

Definition 2. (*Binomial Distribution*): If a random trial succeeds with probability p independent of other trials, the probability of obtaining exactly i successes in n trials is given by

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

The next result quantifies the likelihood of an execution having a lean prefix out of all possible executions consistent with the exponential failure-time assumption.

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*

$$\begin{aligned}\Lambda(m, f) &= \mathbf{P}\{LP(m) \mid N_m = f\} \\ &= \sum_{k=1}^{m} \sum_{\vec{Y} \in \mathbf{Y}} \frac{f!}{y_1! y_2! \cdots y_{k-1}! \, (f-k+1)!} \\ &\quad \left[\frac{1 - e^{-\alpha(m-k)}}{1 - e^{-\alpha m}}\right]^f \left[\frac{1 - e^{-\alpha}}{1 - e^{-\alpha(m-k)}}\right]^{k-1} \\ &\quad \frac{e^{-\alpha k f}}{\exp(-\alpha(k y_1 + (k-1) y_2 + \cdots + 2 y_{k-1}))}\end{aligned}$$

where

$$\vec{Y} = (Y_1 = y_1, Y_2 = y_2, \ldots, Y_m = y_m) \quad \textit{and}$$
$$\mathbf{Y} = \{\vec{Y}: N_{k-1} = k-1, y_k = 0, N_m = f\}$$

is the set of all m-round executions with f failures and a length k lean prefix.

Numerical evaluations of the above expression reveal that under realistic circumstances, consensus can be achieved with non-negligible probability even in executions where the number of failures is double the protocol resiliency [1]. Given the above two lemmas, we can easily obtain the following result.

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

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

We will use this result in Section 3.3 when we consider the optimal duration of a consensus protocol used to disseminate the input in a replicated system.

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 are 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 $n - \Psi$. We quantify the probability of these events in the following result.

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*

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

Figure 1 depicts 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 time unit 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 local area communication network speeds allow a round to be realized in under one second. Consequently, realistic normalized values for α are of the order 10^{-5}. Our results also include larger values for α simply to explore as large a design space as possible including long-haul communication networks and/or less reliable processors.

While the reliability of the nonreplicated system, denoted "single" in the figures, is always perfect (1.0) at the outset of the computation ($T = 0$), the state machine ensembles can have reliability less than one even at this time due to the input consensus having failed. Depending on the system parameters and the length of computation, there are regions where the nonreplicated system has greater reliability than the replicated design. Furthermore, after a sufficiently (unrealistically) long computation, the nonreplicated option has uniformly higher reliability than any replicated alternative.[4] As the replication level grows larger,

[4] Recall that the time axis has been scaled by the mean-time-to-failure value of a single processor. Thus, in a system where, on the average, a processor fails after 100 hours of operation, $T = 0.5$ corresponds to a computation that lasts 50 hours.

the computation time that delineates highly reliable systems from highly unreliable systems becomes more sharply defined. The behavior that is observed in Figure 1 is not unique to the system design we are investigating. In particular, it is independent of the properties of the consensus phase for input dissemination or the Byzantine failure assumption. Similar observations have been made for other replicated systems (such as N-modular-redundancy) that rely on an output threshold for correctness [26]. Formally, we can attribute the above observations to the following asymptotic results.

Let $\bar{T}$ denote a positive random variable such that $\bar{T} = t$ 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

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

In other words, in a system where the input consensus phase takes an insignificant amount of time, the expected time for the replicated ensemble to become incorrect approaches a constant as the replication level increases. In a system where the correctness threshold is majority, the above limit becomes $\ln(2) = 0.693$ as $n \to \infty$. Interpreting this result in a system where $1/\lambda = 100$ hours, we expect the system output to have become incorrect after a computation of 69.3 hours. We note that some of these observations were also made by Tay [31].

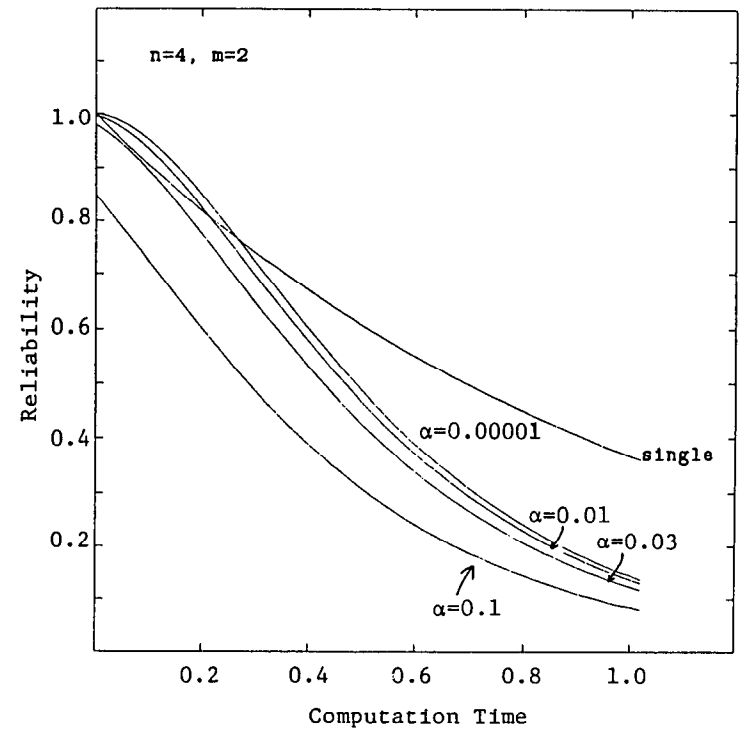
As the replication level of the system increases, not only does the expected time for the system to become incorrect approach a constant, the transition time from a correct system to an incorrect one becomes more deterministic. Intuitively, as the replication level increases, there are more processors that can fail. Furthermore, the number of failures that occur in different intervals becomes more uniform and the interval between successive failures shrinks. Consequently, the time at which the single additional failure occurs that is sufficient to violate the correctness threshold becomes more sharply defined. We formalize this intuition in the following proposition.

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*

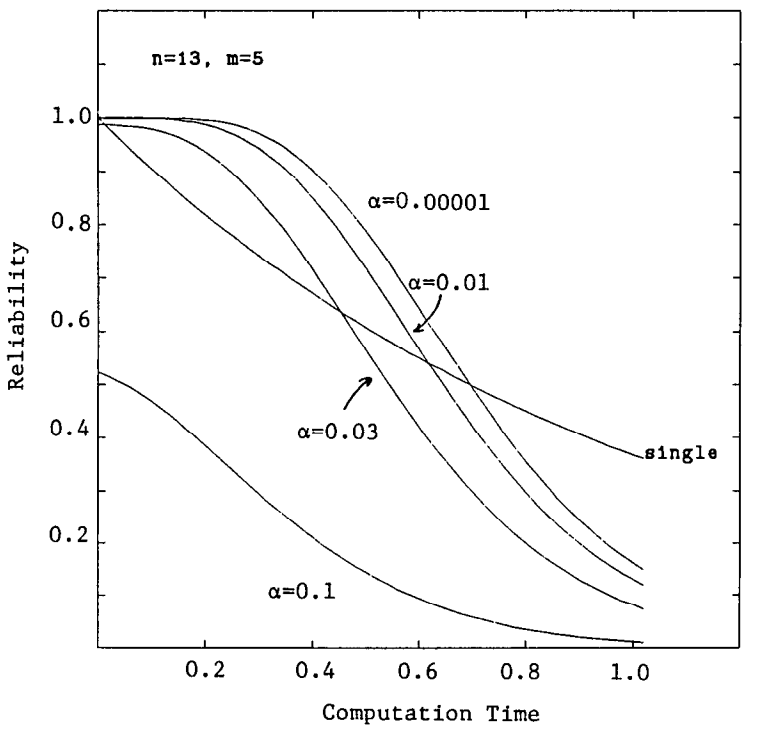
$$\mathrm{Var}(\bar{T}) = \sum_{j=0}^{n-\Psi} \frac{1}{(n-j)^2}.$$

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

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



(a)



(b)

Fig. 1. Reliability of state machine ensembles.

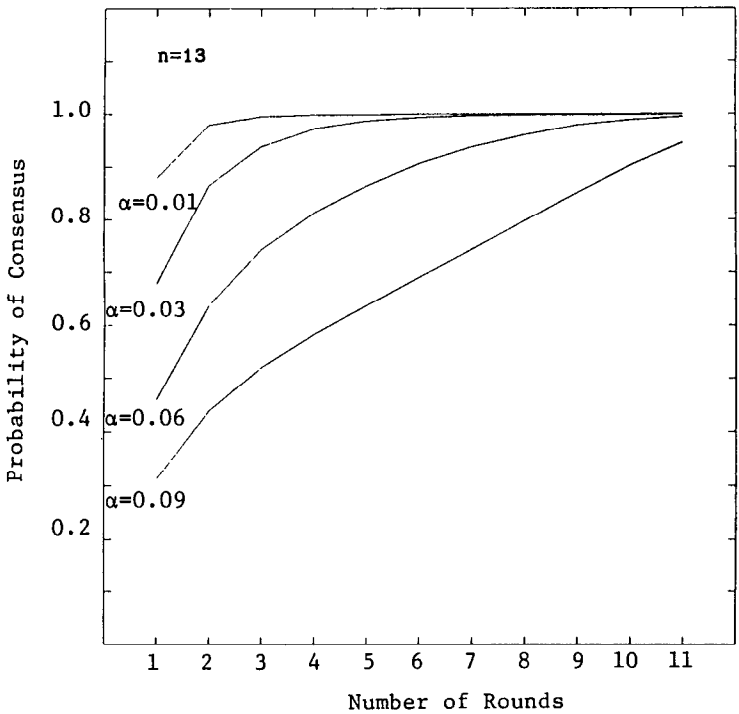


Fig. 2. Probability of consensus versus protocol duration.

Note that in systems where the correctness threshold is $\Psi = \psi n$, the above limit is zero. For example, consider the curve in Figure 1(b) corresponding to $\alpha = 0.00001$. Even for this moderate level of replication ($n = 13$), the curve begins to take on a step function shape—the system makes a rapid transition from being highly reliable to being highly unreliable around a computation given by Proposition 1 (approximately 0.6 units). 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 apply the previous results to explore the trade-offs 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 rounds of execution. 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

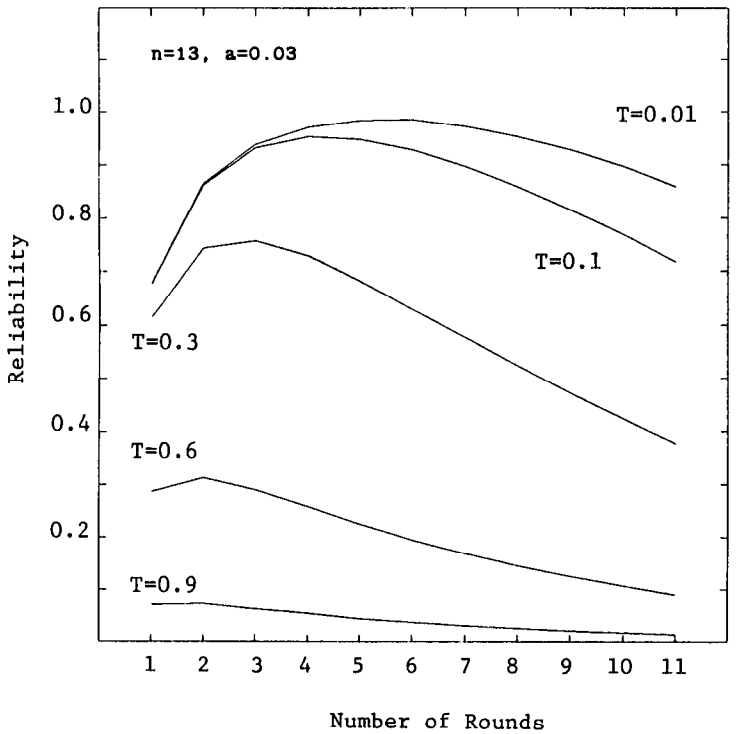


Fig. 3. System reliability versus protocol duration.

in Figure 2. Note that for small (realistic) values of α, a 2- to 3-round execution defines the point of diminishing returns—the system achieves consensus with high probability for this protocol duration and any additional rounds of execution results in negligible gain. For large values of α, however, each additional round of execution increases the probability of consensus by a significant amount.

Next we investigate the effect of input consensus protocol duration 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 observe the reliability of the state-machine ensemble, 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 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 [3, 4, 5, 23]. While they are the only class of protocols that permit consensus to be achieved in asynchronous systems, randomized protocols are often touted as fast alternatives to deterministic protocols also in synchronous systems.

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

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

for some positive constant $p \leq 1$. In other words, provided that the resiliency requirement has not been violated (at most ρ faulty processors), the protocol has a fixed probability p of achieving agreement in the current round. Note that such a protocol can achieve consensus with probability p even after the first round. Analogous to Theorem 1, the following result gives 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*

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

As a concrete example, consider the randomized protocol presented in [5]. 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) $\frac{1}{3}$. 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 processors. Note that this protocol has substantially different behavior than the class of deterministic, authenticated protocols studied earlier. First, the probability of consensus starts off small ($\frac{1}{3}$) 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 our 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 correct. Such a characterization (analogous to Lemma 1) is beyond the scope of this paper.

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. As discussed in Section 3.2, there comes a time such that it takes only one additional processor failure (the $(n - \Psi + 1)st$) to make 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

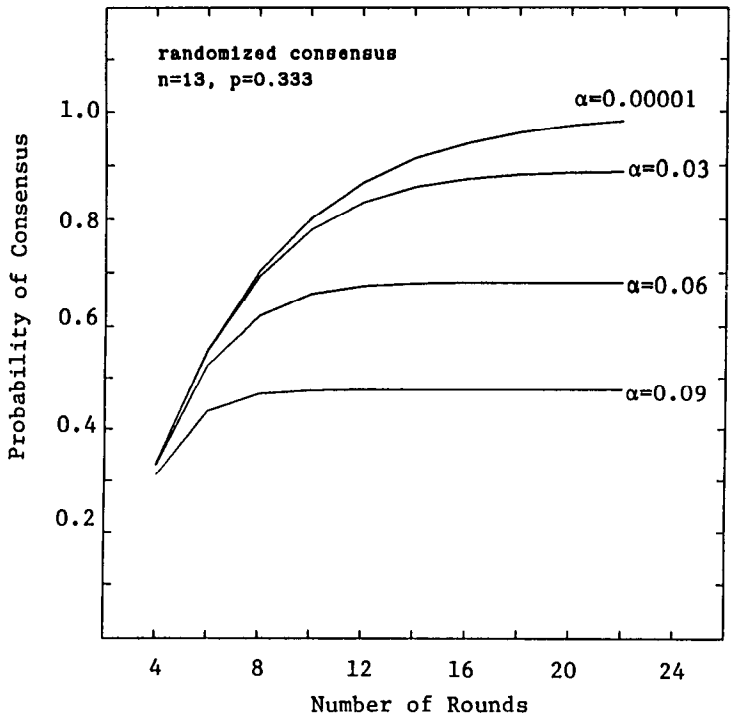


Fig. 4. Probability that randomized protocol achieves consensus.

faulty. It will be convenient for us to redefine 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 overt faulty behavior 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 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 lengths where the expected number of processors to fail during the consensus phase is larger. Obviously, for consensus protocols that can detect only some fraction of the

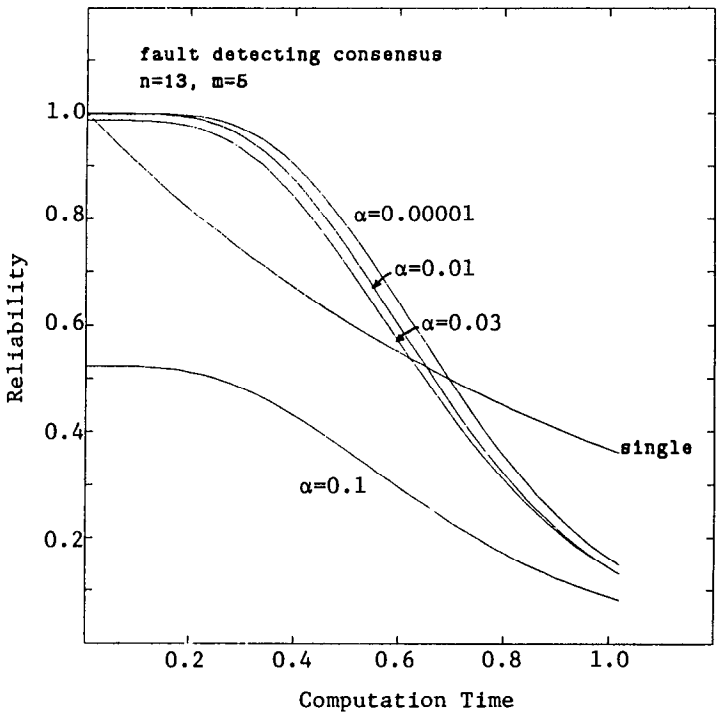


Fig. 5. Fault-detecting consensus protocol.

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 all processors that have failed by some time θ after the beginning of the application computation. Given such a mechanism, the expression for system reliability becomes:

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*

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

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 results 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 (address,

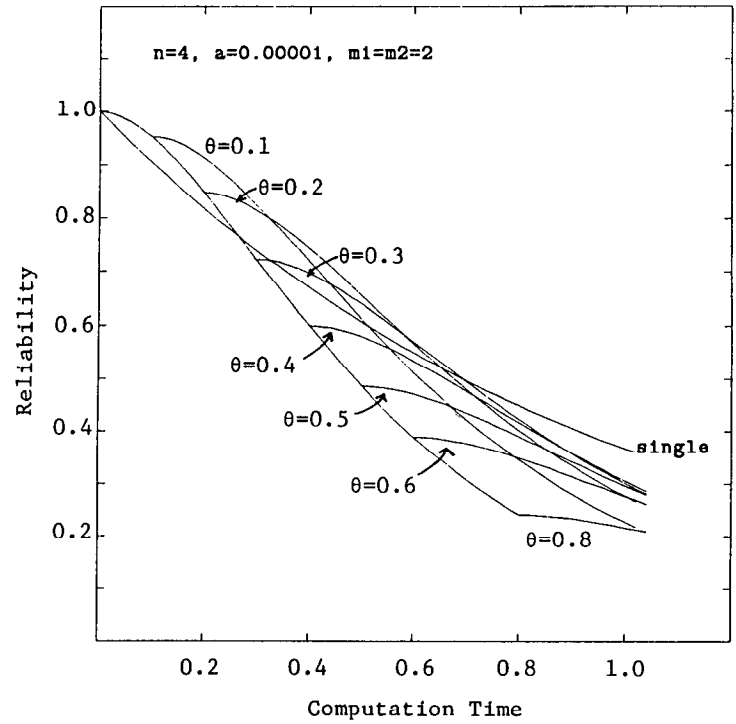
value) pair for each memory update operation as the consensus value [24]. To keep the overhead associated with this technique low in the presence of frequent memory updates, processors could compute a hash function on their state only after a fixed number of iterations of a loop or after the passage of some length of time to generate a suitably compact consensus value [15]. 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.

Let $m1$ and $m2$ denote the number of rounds that the input-consensus protocol and the number of rounds that the fault-detector-consensus protocol execute, respectively. Figure 6 illustrates the effectiveness of this mechanism on the reliability of systems with 4 and 13 processors. Note that these are the same 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 (i.e., $m1 = m2$). We notice that at the time fault detection takes place, the system ceases to suffer rapid loss of reliability. Instead, the current reliability is maintained for some time and then the system continues 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 duration fixed at 5 rounds. Intuitively, the expected number of faulty processors in the system by the time the fault detector executes increases with increasing θ. Consequently, the probability of the fault-detector consensus succeeding decreases. Clearly, $m2$ should be increased as θ increases.

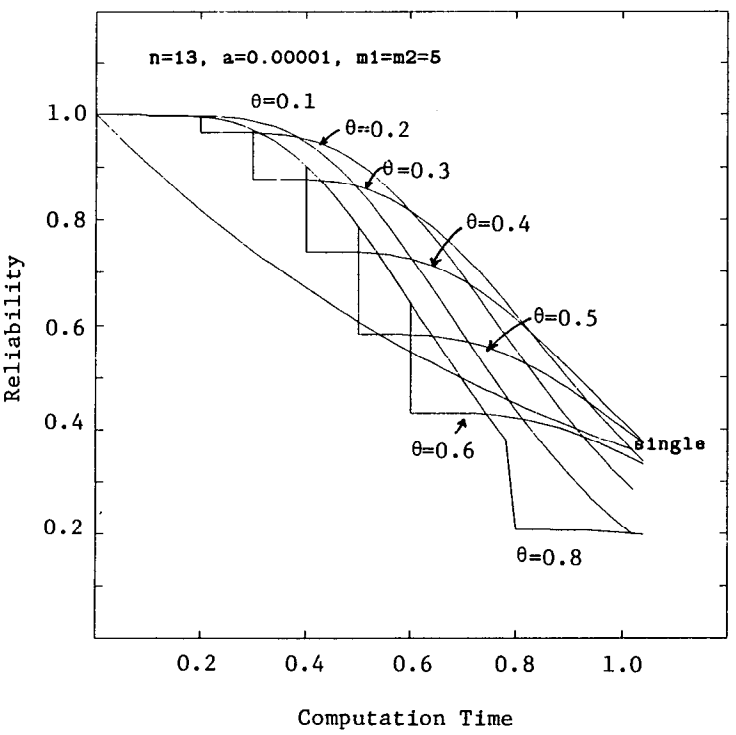
Given n processors that fail according to our stochastic model, the expected number of faulty processors by time θ is given by $n(1 - e^{-\theta})$. In Figure 7 we recompute the reliability of the system depicted in Figure 6(b) where the fault detector consensus duration is made to increase according to the relation $m2 = m1 + \lceil n(1 - e^{-\theta}) \rceil$. In other words, the number of rounds that the fault detector executes in addition to $m1$ corresponds to the expected number of faulty processors in the system at that time. The effect of this "adaptive parameter" fault detector is much more positive in that the system remains uniformly more reliable with fault detection than without.

Consider the decision to select the time of fault detection (θ). If it occurs early in the computation, few faulty processors will be detected. Although more faulty processors may be detected if it occurs late in the computation, the system reliability may already have become unacceptably low. The results of Section 3.1 give us a formal interpretation of the "early" and "late" characterization above. More precisely, we can divide the computation phase into two regions with respect to $\mathbf{E}(\bar{T})$, the expected time for system failure. Intuitively, to gain the maximum benefit from fault detection, it should occur just before $\mathbf{E}(\bar{T})$. With this choice of θ, the number of faulty processors detected will be maximized while the system continues to be highly reliable.

While the above argument is valid in the limit as the system-replication level grows unbounded, it has to be modified slightly for finite n. As we saw in Figure 1(b), the transition of a system with 13 processors from levels of high reliability



(a)



(b)

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

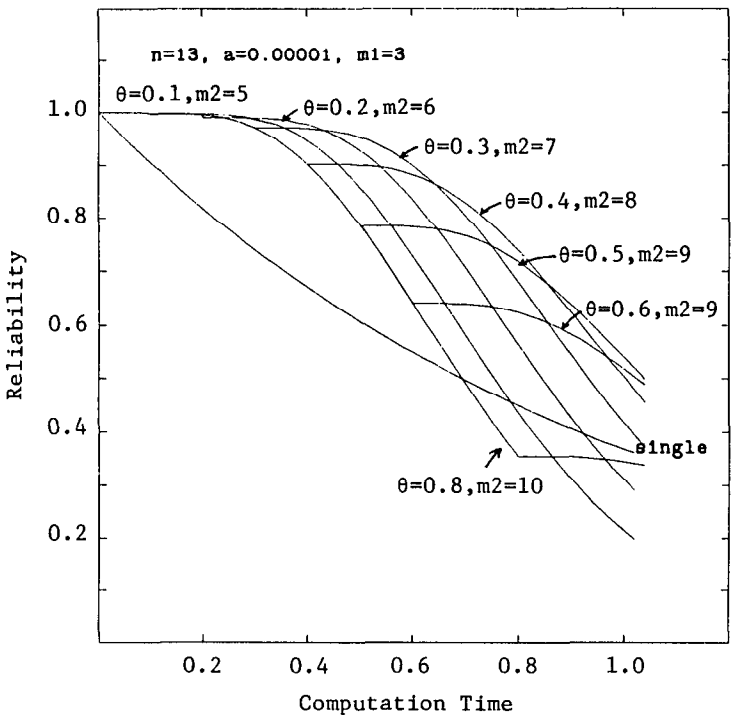


Fig. 7. The effect of variable execution length fault detection.

to those of low reliability is not instantaneous at $\mathbf{E}(\bar{T})$. While it is centered about $\mathbf{E}(\bar{T})$ ($T = 0.693$), it spans approximately 0.4 time units. Consequently, a reasonable value for θ for this system might be 0.4, which is considerably smaller than $\mathbf{E}(\bar{T})$.

Our analysis for the computation augmented with a single fault detection phase can be extended in an obvious way to consider multiple phases. Given that fault detection incurs a certain cost, the question of determining the optimal number of fault detection phases to occur during a computation is closely related to that of inserting checkpoints in a sequence of tasks [32].

4. LACK OF AUTHENTICATION

Consider a fault-tolerant system design constrained not to 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 [22]. 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 [29].

We can reason about the unauthenticated system by considering an authenticated system. This follows from the manner in which the unauthenticated protocol of [29] 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 it has a lean prefix of length at most $n/3$ phases, then the execution will achieve consensus. If, on the other hand, the lean prefix is longer than $n/3$ phases, there must have been $n/3$ or more failures during the execution. Without a finer characterization of the failure sequence, no guarantee can be made that the simulation of authentication remains valid. Consequently, consensus cannot be guaranteed either. Which failure sequences involving greater than $n/3$ processors permit consensus in this environment 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 assumption that there exists some time when we have high confidence that the system is fault-free (or that there are at most a small number of potentially faulty components). 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 replication level, 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 results are a first step towards a methodology for designing fault-tolerant distributed system that satisfy formal reliability specifications. For example, a specification of the form "$R(n, m, \theta, T) \geq P_{\min}$ for all $T \leq T_{\max}$" sets a lower bound on the system reliability during a mission that lasts $T_{\max}$ units of time. Contrast this with a resiliency specification which states that correct operation is to be guaranteed as long as the number of failures is bounded. In environments where such a bound is not known or does not exist, our reliability specification is the only viable alternative. By generating results similar to Figure 7, a designer can easily evaluate various system-design choices with respect to the reliability specification.

APPENDIX

PROOF OF LEMMA 1. 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. Recall that our failure model permits Byzantine behavior for those processors that have failed. 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 reliable broadcast protocols with authentication [10]. Therefore, since reliable broadcast cannot be achieved in such an execution, consensus cannot be achieved either.

Now we consider the "if" case for the lemma. We will exhibit a consensus protocol that consists of n concurrent "early stopping" (eventual) protocols that solve the reliable broadcast problem [9]. In round one, each processor initiates an instance of the reliable-broadcast protocol (acting as the transmitter) in order to disseminate its local input value. From then on, it also participates in the $n - 1$ reliable-broadcast protocols initiated by the other processors. Given this construction, the success of a protocol to achieve consensus in m rounds depends on the length of the longest reliable broadcast protocol—if all n protocols terminate by round m then consensus is guaranteed. Note that a processor may decide on values corresponding to each of the concurrent reliable broadcasts in different rounds. Once a processor decides on a value corresponding to a particular broadcast, it stops participating in that protocol by not sending any messages and ignoring all incoming messages. However, it continues to participate in the protocols corresponding to undecided values.

By the hypothesis, the m-round composite execution has a lean prefix. Let f be the length of the lean prefix for this execution and let h denote the total number of processors that fail during the m rounds. By definition, $N_{f-1} = f - 1$, round f is failure free, and $h - f + 1$ additional failures occur in rounds $f + 1$ through m. Consider the behavior of the N_{f-1} faulty processors that are present by the end of round f. Regardless of when these processors failed or how they exhibited their faultiness, each of the n early stopping reliable-broadcast protocols encounters an f-round execution where at most $f - 1$ processors fail and the last round is failure-free. This is precisely the characterization for the termination of an early-stopping protocol[5] [9, 16]. Of course, processors may have decided on some of the values at rounds earlier than f; round f represents the latest time by which any of the n reliable broadcasts can terminate.

By the end of round f, all correct processors are able to construct the consensus vector $\mathbf{W}$ from the n decision 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 until round m serves to synchronize the correct processors with respect to the termination of the consensus protocol. Finally, note that the vector $\mathbf{W}$ constructed by the correct processors at round $f + 1$ continues to satisfy the requirements for consensus at round m. The $h - f + 1$ entries corresponding to the processors that fail during rounds $f + 1$ through m are identical in all of the consensus vectors of correct processors. Thus they continue to satisfy the requirements for consensus by the end of round m.

PROOF OF LEMMA 2. Since processors fail independently, we can interpret the failure of each of them during t time units as a "success" outcome in n independent

[5] 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 being slightly optimistic in our analysis.

trials. The probability of a success is simply the probability that an exponential random variable is smaller than t. Furthermore, the number of processors that fail during disjoint intervals have a common binomial distribution [12]. The common-distribution result is a consequence of the "memoryless" property of the exponential distribution.

By definition, an m-round execution $(Y_1, Y_2, \ldots, Y_m)$ where f processors fail has a lean prefix of length k if and only if

$$N_{k-1} = k - 1,$$
$$Y_k = 0, \quad \text{and}$$
$$N_m - N_{k-1} = f - k + 1.$$

The joint distribution of these events can be obtained easily by conditioning on the number of failures in each round of execution (starting with $Y_1 = y_1$) and repeatedly applying Definition 2. Given the probability of the execution having a lean prefix of length k, the result follows by summing over all $k \leq m$.

PROOF OF THEOREM 1. 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. □

PROOF OF THEOREM 2. The desired expression is obtained by first conditioning on the number of failures during consensus, N_m, 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. □

PROOF OF PROPOSITION 1. 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 [12]. 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. □

PROOF OF PROPOSITION 2. This 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 [12]. □

PROOF OF THEOREM 4. 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. □

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, Joe Halpern, and the referees.

REFERENCES

1. BABAOĞLU, Ö. Stopping times of consensus protocols: a probabilistic analysis. *Inf. Process. Lett. 25*, 1 (May 1987), 163–169.
2. BABAOĞLU, Ö. Distributed consensus in the presence of probabilistic failures. Dept. of Computer Science, Cornell Univ., Ithaca, N. Y. In preparation.
3. BEN-OR, M. Fast synchronous Byzantine Agreement. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing* (Minaki, Ont., Canada, Aug. 1985), ACM, New York, 1985, 149–151.
4. BRACHA, G. An $O(\lg n)$ expected rounds randomized Byzantine Generals protocol. In *Proceedings of the 17th ACM Symposium on Theory of Computing* (Providence, R. I., May 1985), ACM, New York, 1985, 316–326.
5. CHOR, B., AND COAN, B. A. A simple and efficient randomized Byzantine Agreement algorithm. *IEEE Trans. Softw. Eng. SE-11* (June 1985), 531–538.
6. COHN, S. N. Assessment of the new Arpanet routing. Presentation at the Workshop on Fault-Tolerant Distributed Computing, (Pacific Grove, Calif., Mar. 17, 1986).
7. CRISTIAN, F., AGHILI, H., STRONG, R., AND DOLEV, D. Atomic broadcasts: from simple message diffusion to Byzantine Agreement. In *Proceedings of the 15th Symposium on Fault Tolerant Computing* (Ann Arbor, Mich., June 1985), IEEE, New York, 1985, 200–206.
8. DIFFIE, W., AND HELLMAN, M. New directions in cryptography. *IEEE Trans. Inf. Theor. IT-22*, 6 (Nov. 1976), 644–654.
9. DOLEV, D., REISCHUCK, R., AND STRONG, H. R. 'Eventual' is earlier than 'immediate.' In *Proceedings of the 23rd Symposium on Foundations of Computer Science* (Chicago, Ill., Nov. 1982), IEEE, New York, 1982, 196–203.
10. DOLEV, D., AND STRONG, H. R. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput. 12*, 4 (Nov. 1983), 656–666.
11. DWORK, C., AND MOSES, Y. Knowledge and common knowledge in a Byzantine environment I: Crash failures. Private communication, Jan. 1986.
12. FELLER, W. *An Introduction to Probability Theory and Its Applications*. Wiley, New York, 1971.
13. FISCHER, M. J. The consensus problem in unreliable distributed systems (a brief survey). Tech. Rep. YALEU-DCS-RR-273, Dept. of Computer Science, Yale Univ., New Haven, Conn., June 1983.
14. FISCHER, M. J., AND LYNCH, N. A lower bound for the time to assure interactive consistency. *Inf. Process Lett. 14*, 4 (Apr. 1982), 183–186.
15. GARCIA-MOLINA, H., PITTELLI, F., AND DAVIDSON, S. Applications of Byzantine agreement in database systems. Tech. Rep. TR 316, Princeton Univ., Princeton, N. J., June 1984.
16. HADZILACOS, V. Issues of fault tolerance in concurrent computations. Ph.D dissertation, Tech. Rep. TR-11-84, Aiken Computation Lab., Harvard Univ., Cambridge, Mass., June 1984.
17. LAMPORT, L. Time, clocks and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (July 1978), 558–565.

18. LAMPORT, L. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. Program. Lang. Syst. 6*, 2 (April 1984), 254–280.
19. LAMPORT, L., AND MELLIAR-SMITH, P. M. Synchronizing clocks in the presence of faults, *J. ACM 32*, 1 (Jan. 1985), 52–78.
20. LAMPORT, L., SHOSTAK, R., AND PEASE, M. The Byzantine Generals problem. *ACM Trans. Program. Lang. Syst. 4*, 3 (July 1982), 382–401.
21. MAHANEY, S. R., AND SCHNEIDER, F. B. Inexact agreement: accuracy, precision, and graceful degradation. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing* (Minaki, Ont., Canada, Aug. 1985), ACM, New York, 1985, 237–249.
22. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (Apr. 1980), 228–234.
23. RABIN, M. Randomized Byzantine Generals. In *Proceedings of the 24th Symposium on Foundations of Computer Science* (Tucson, Ariz., Nov. 1983), IEEE, New York, 1983, 403–409.
24. SCHNEIDER, F. B. Byzantine generals in action. *ACM Trans. Comput. Syst. 2*, 2 (May 1984), 145–154.
25. SCHNEIDER, F. B., AND LAMPORT, L. Paradigms for distributed programs. In *Distributed Systems: Methods and Tools for Specification. Lecture Notes in Computer Science, vol. 190.* M. Paul, and H. J. Siegert, Eds., Springer-Verlag, New York, 1985, 431–480.
26. SIEWIOREK, D. P., AND SWARZ, R. S. *The Theory and Practice of Reliable System Design.* Digital Press, Belford, Mass., 1982.
27. SPECTOR, A. Z., AND GIFFORD, D. Case study: the space shuttle primary computer system. *Commun. ACM 27*, 9 (Sept. 1984), 874–900.
28. STANKOVIC, J. A. A perspective on distributed computer systems. *IEEE Trans. Comput. C-33*, 12 (Dec. 1984), 1102–1115.
29. SRIKANTH, T. K., AND TOUEG, S. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Tech. Rep. 84-623, Dept. of Computer Science, Cornell Univ., Ithaca, N. Y., July 1984.
30. STRONG, H. R., AND DOLEV, D. Byzantine agreement. In *Digest of Papers, Spring Compcon 83* (San Francisco, Mar. 1983), 77–81.
31. TAY, Y. C. The reliability of (k, n)-resilient distributed systems. In *Proceedings of the 4th Symposium on Reliability in Distributed Software and Database Systems* (Silver Spring, Md., Oct. 1984), IEEE, New York, 1984, 119–122.
32. TOUEG, S., AND BABAOĞLU, Ö. On the optimum checkpoint selection problem. *SIAM J. Comput. 13*, 3 (Aug. 1984), 630–649.

Received May 1986; revised April 1987; accepted May 1987