Abstract

The protocol for synchronous approximate agreement presented by Dolev et al.[7] exhibits the undesirable property that a faulty processor, by the dissemination of a value far removed from the values held by good processors, may delay the termination of the protocol by an arbitrary amount of time. Such behavior is clearly undesirable in a fault tolerant control system subject to hard real-time constraints. This work presents a mechanism by which discarding data suspected of being from failed processors can lead to quicker, predictable, convergence upon an agreement value. Under specific assumptions about the nature of values transmitted by failed processors relative to those transmitted by good processors, we present Monte Carlo simulations whose quantitative results illuminate the trade-off between accelerated convergence and the accuracy of the value agreed upon.

1. Introduction

One approach to the construction of a fault tolerant computing system is through redundancy in the system hardware and/or software. An architecture for a redundant avionics subsystem, resilient to some number of hardware and software failures, is shown in Figure 1. Four processors, P1 through P4, execute individual replicates of a particular software function. That function causes each processor to map digital sensor input into an output which is intended to drive some physical actuator or display. Each processor has access to its own sensor. The output of each processor is a vote in an election to determine the sole output of the (sub)system. The redundancy in this subsystem is intended to tolerate the failure of any single processor, sensor, or function (software failure).

The assumption that the function replicates fail independently (an important assumption for the efficacy of N-version programming[3,5]) has been empirically shown to be invalid in practice[12]. Also, analytical work suggests that the possibility of coincident errors in the function replicates may require a higher degree of redundancy than would be intuitively expected and that, in some situations, the reliability of the N-version system would be less than that of a 1-version system[8]. These problems, however, are irrelevant to the structure of a modularly redundant system. Their impact is felt in the degree of fault resilience, and it has been argued that careful design of the replicates and their integration into the system can lead to an acceptable level of resilience[1].

A key assumption in the preceding argument that redundancy provides resilience is that of input congruence. If the software functions are truly replicates of one another, then they are intended to compute the same output if they are presented with the same set of inputs. In the system of Figure 1, several intuitive alternatives for providing function input are available:

(a) Each function uses the data presented by its attached sensor.

(b) A single sensor provides input for the system. That sensor's output is cross-
strapped to the input of all function replicates.

(c) A single sensor value is reliably broadcast to all replicates.

(d) The values of all sensors are reliably disseminated to all processors. Each processor then chooses some representative data value (presumably derived from the ensemble of sensor readings) as input for its function replicate.

The first alternative (a) is clearly not conducive to input congruence in that there is no coordination of sensors—each sensor delivers a value independent of the other sensors, and those values may be arbitrarily far apart. The second alternative (b) appears to gain input congruence at the expense of providing less fault tolerance—the chosen sensor becomes a critical single point of failure. The cross-strapping approach, however, generally offers no advantage for input congruence—wires break and output drivers go out of tolerance (an intuitively appealing presentation of this scenario may be found in [13]).

The last two alternatives, (c) and (d), are, in fact, valid means of assuring input congruence. Their validity relies upon the term "reliable" as it applies to the dissemination of data in a network of processors connected by some communications medium. Of course, approach (c) is still vulnerable to the failure of the single sensor.

1.1. Reliable Dissemination Through Agreement

The mechanism by which the reliable dissemination of data is achieved is an agreement protocol. The transmitting process disseminates a value to all of the receiving processes. The receivers must then engage in several rounds of message-exchange. At the end of those rounds the receivers may infer the value transmitted; if, indeed, the transmitter is non-faulty. Otherwise, the receivers may agree upon a null value which indicates a faulty transmission. The agreement protocol requires that two conditions be met[14]:

Agreement 1: All non-faulty recipient processes agree on the same value.

Agreement 2: If the transmitting process is non-faulty, all non-faulty recipients agree on the value transmitted.

In the absence of faults, exact agreement is trivially achieved by the transmitter's broadcast of the same data value to all receiving processes. In the presence of faults, the possibility of faulty behavior by any process, or possible corruption of any data message by the communications network, transforms a trivially solved problem into one which is decidedly non-trivial, both conceptually and in terms of the resources required to achieve a solution. Bit-by-bit agreement, with no assumptions about failure modes, may be achieved by a Byzantine agreement protocol [14]. Such protocols require substantial connectivity of the communications network (to avoid the forwarding of data through faulty processors), many rounds of message-passing (to assure that faulty processors cannot contribute to the agreement value), or message authentication through cryptographic means or through error-detection coding techniques (to limit the degree of faulty behavior). We note several salient facts about exact Byzantine agreement:

1. The system must be synchronous[10]. Intuitively, a synchronous protocol permits the determination of whether or not a remote process has crashed or is simply slow.

2. A system of n processes (not utilizing authenticated messages) can be made resilient to the failure of t processes, provided that t < \frac{n}{3} [14].

3. At least t + 1 rounds of message passing are required for exact Byzantine agreement under the most general assumptions[9]. The original Byzantine agreement algorithm[14] required a total of \(O(n^{t+1})\) messages; the best algorithm[6], in the worst case, requires \(O(nt + t^3)\) messages.

A scan of these facts leads one to the conclusion that a Byzantine agreement protocol is typically extremely expensive in terms of replicated system components and communication bandwidth.

In the context of the system architecture of Figure 1, agreement would be utilized to disseminate sensor values in alternatives (c) and (d). In alternative (c), however, even with the reliable data dissemination provided by an agreement protocol, the entire system is vulnerable to inaccuracy of the sole sensor—erroneous sensor output could be reliably broadcast to all processors, resulting in unanimous consent on an output which is erroneous. In contrast, consider alternative (d) in which each processor could serve as the transmitter in a Byzantine agreement protocol (conceptually, one would envision four protocols, each with a different process as transmitter, concurrently in operation). At the conclusion of these (expensive) protocols, each non-faulty processor would
know the values of all four sensors, and could apply some (application specific) selection or averaging algorithm to choose a single "sensor value" as input to its function replicate. The validity of this scheme rests upon the facts that all non-faulty processors must have the same set of sensor data values as a consequence of the agreement-based dissemination and that the selection or averaging algorithm is the same at all processors.

1.2. Approximate Agreement

An alternative approach to providing data to the function replicates of Figure 1 is through approximate agreement. That is, we require that all non-faulty processors derive an input value which must be "close to" the value derived by any other non-faulty processor, and that value must be within the range of the set of non-faulty sensor values. Informally speaking, these requirements are met by the approximate agreement protocol developed by Dolev et al. [7]. That protocol is intended to serve a network of \( n \) processors, at most \( t < \frac{n}{2} \) of which may be faulty. The maximum difference between any two final agreement values must be \( n \) larger than some tolerance, \( \epsilon \), a fundamental parameter of the protocol. The approximate agreement protocol operates as shown in Figure 2. This example depicts a four processor system \( (n = 4 \text{ and } t = 1) \) in which the first processor is assumed to be faulty. During each step of the protocol, each processor maintains its current estimate of its agreement value; this is represented by the horizontal array at the top of the figure. The first action of a step of the the protocol is each processor’s broadcast of its value to all processors (including itself). In Figure 2 (a), the \((i, j)\) entry of the matrix in the center represents the value received by \( P_i \) from \( P_j \) at the end of this round of message passing. \( P_1 \), the faulty processor, is shown sending different values to different processors, and, as illustrated, its multiset of received values is irrelevant—a faulty processor will not, in general, adhere to the approximate agreement protocol. Each processor then applies the same value update algorithm to its multiset of received values (i.e., its row of the matrix) as shown in Figure 2(b). For the case of a synchronous system, that update procedure involves sorting the multiset, discarding \( t \) values on each end of the range of the multiset, retaining the first and every \( t \)-th successive elements thereafter, and taking the arithmetic mean of the values so retained. It can be shown that a single application of the update function at all good processors will effectively reduce the diameter\(^1\) of the multiset of values held by good processors by a factor of 

\[
d = \left\lceil \frac{n - 2t - 1}{t} \right\rceil + 1.
\]

Since \( n > 3t \), this factor must be at least 2.

We note several fundamental differences between approximate agreement and (exact) Byzantine agreement: In Byzantine agreement, the agreement value is identical, on a bit-by-bit basis, for all non-faulty processors. In approximate agreement, the agreement value derived by any non-faulty processor must

\(^{1}\)The diameter of a multiset of real numbers is the absolute value of the difference between the largest and smallest elements of that multiset.
1.3. Halting the Approximate Agreement Protocol

Processors which are participating in a synchronous approximate agreement protocol each determine a halting round independently. This follows from the fact that a non-faulty processor, \( P_i \), starting with an initial multiset of values \( V_i^1 \), can only pessimistically assume that the diameter of its multiset will be decreased by a factor of \( d \) on each round. Thus, the processor anticipates executing

\[
H_i^1 = \left\lceil \log_d \left( \frac{\text{diameter}(V_i^1)}{\epsilon} \right) \right\rceil
\]

rounds of the protocol in order to ensure that the multiset of values held by any good processor has a diameter no larger than \( \epsilon \). Two additional points concerning this halting technique are relevant: First, on the round after which a non-faulty processor halts, it broadcasts a halt tag with its agreement value on the next round of the protocol; after receipt of a tagged value from a halted processor, another processor will continue to use that tagged value as the value transmitted (without receipt of an actual message). Secondly, a processor whose value is not received within a timeout interval (presumably representative of the maximum message transmission time), will be assumed to have send some arbitrary default value.

The fundamental difficulty with this halting technique defined by \( H_i^1 \) is that \( H_i^1 \) can be arbitrarily large; i.e., faulty processors, by the initial broadcast of values which are arbitrarily far removed from actual values, can arbitrarily enlarge the diameter of the initial multisets of good processors, and consequently, delay convergence arbitrarily.

2. Accelerated Convergence

In this section we will develop and empirically evaluate two approaches to faster convergence: adaptive halting and initial multiset editing. The degree to which each approach succeeds depends on knowledge of the expected behavior of the system in which the protocol operates. To that end we define a model of how good processors and faulty processors will behave:

---

### Two-Faced Independent Fault Model

**Initialization:** Each good processor independently draws an initial value (i.e., will read its sensor) from a distribution of "good values". For the purposes of our simulations, we define this distribution to be \( N(0, 1) \), the standard normal distribution.

**Broadcast:** Whenever a faulty processor transmits a message, it sends a value drawn independently from a distribution of "bad values". We view a broadcast to \( n \) recipients as \( n \) separate message transmission. We utilize \( N(0, \alpha) : \alpha \geq 1.0 \) as the family of bad value distributions.

---

We make no distinction between the failure of a processor and the failure of a sensor; the two constitute a single entity as far as fault behavior is concerned. In the following discussion, when we refer to a "failed processor", we actually refer to the processor/sensor pair. All faults are assumed to be independent of previous fault behavior and the current state of the system. Our model, therefore, does not address the notions of correlated faults or of processors failing in coordinated fashion. We term this model the two-faced independent fault model, and we utilize this model in the remainder of this paper.

Note that values transmitted by faulty processors share the same mean with the values chosen by good processors. This facet of the model was intended to represent worst-case behavior—faulty values chosen from a distribution whose mean is substantially different from the mean of the distribution of good values are easily discarded by the approximate agreement update algorithm. A model based on consistent failure behavior ("one-faced lying") is also easily handled by the approximate agreement updating procedure. One-faced lying would have a faulty processor disseminate the same bad value to all recipients during the broadcast phase. This implies that every good processor would have the same multiset on every round of the protocol. Further, the value computed by a good processor in the first round can be shown to be the agreement value (and that value will be the same for all good processors in the system). That is, under the assumption of one-faced lying, the approximate agreement protocol should halt in one round.
2.1. Adaptive Halting

The adaptive halting procedure is outlined below:

Adaptive Halting Protocol

- Let $V^r_i$ be the multiset of (good and bad) values held by processor $P_i$ at the conclusion of the $r$-th round of the protocol. Let $H^r_i$ be $P_i$'s computed halt round at the end of the $r$-th round of the protocol. $H^r_i$ is the previously defined static halt round for $P_i$ as defined in the original protocol.

- After accumulating $V^r_{i+1}$ in the broadcast phase of round $r + 1$, compute

$$\text{temp}_i = \max\left(0, \left\lceil \log_d \left( \frac{\text{diameter}(V^r_{i+1})}{\epsilon} \right) \right\rceil \right).$$

- Set $H^r_{i+1}$ to $\min(H^r_i, r + 1 + \text{temp}_i)$.

The obvious goal of this approach is to exploit the round-by-round variation in $V^r_i$ to achieve much faster convergence than that guaranteed by the pessimistic convergence factor $d$. For example, suppose that a single faulty processor ($P_f$), in a 4-processor system, {$P_1, P_2, P_3, P_4$}, consistently broadcasts a common value which is significantly different from the valid sensor values on the first round. The original protocol will then compute the same large halt rounds for all good processors based on their common inflated initial multisets. Moreover, the agreement values computed by the good processors on the first round will agree exactly, and thus all rounds after the first will be unnecessary overhead. If $P_f$, at some later round, consistently broadcasts a common value relatively close to the agreement values, the adaptive halting procedure will then provide quicker termination. Adaptive halting can be shown not to invalidate any of the functional correctness properties of the original approximate agreement protocol.

Figure 3 presents the results of adaptive halting applied to a four processor system (one of those processors is faulty). As in all experiments presented in this paper, $\epsilon = 0.1$. Smaller values of $\epsilon$ (all other system parameters remaining constant) merely shift the plots upward, and larger values shift the plots downward. All of our results are derived for four processor systems (one processor assumed to be faulty). Larger systems tend to shift the plots upwards. The vertical axis of all the plots is the mean halting round for a Monte Carlo simulation (1000 replications) of the protocol in operation for a specific value of $\alpha$, shown on the horizontal axis. The halting round on any replication is defined to be the largest halt round for all non-faulty processors.

The results are disappointing but intuitive. Adaptive halting provides negligible gains over the traditional pessimistic approach. In effect, the probability that a faulty process broadcasts a common "nearly good" value is so small that the adaptive halting round only rarely becomes smaller than $H^1_i$ for all non-faulty processors. In a seven processor system, the two performance curves are even closer together than in Figure 3. The trend towards less gain in larger systems is also intuitive: adaptive halting succeeds only when bad values are somewhat consistently disseminated to good processors, and the greater the number of processors the more difficult it is to lie consistently under the two-faced independent failure model. We would expect that adaptive halting would be valuable only in a failure regime characterized by processors which operate correctly most of the time but which fail wildly for only brief intervals. Thus we reject adaptive halting as a general approach for accelerated convergence and turn our attention to the possibility of editing sensor data to achieve faster convergence without introducing error into the agreement protocol.

2.2. Editing the Initial Multiset

If we consider the application of approximate agreement in a real-time fault-tolerant control system, we
gain some leverage by which the halting difficulty may be overcome. We assume that the control loop (i.e., read sensors, disseminate values, compute control function) is frequently executed—a 30Hz loop would not be uncommon. A consequence of this frequency is that the sensor readings will presumably not change “too much” on successive iterations. For example, if the sensor is an accelerometer in an avionics guidance system, the dynamics of the aircraft and the integrity of the airframe itself would impose bounds on the rate at which acceleration can change. Therefore, we assume that certain application-specific criteria can be used to rationally edit bad data values from the initial multisets accumulated in the first round of the approximate agreement protocol.

In order to verify that data editing can provide substantial performance benefits, we consider two theoretical bounds on the best possible halting behavior of an approximate agreement protocol.

**Absolute Optimal** The earliest possible halting would be achieved by globally monitoring, on each round, the agreement values computed by all good processors. As soon as the diameter of that multiset becomes less than $\epsilon$, halt. Of course, there is no practical way to discriminate between faulty and non-faulty processors or to globally monitor agreement values (although global snapshots[4,15] could be used, at great expense, to monitor the values).

**Practical Optimal** If we compute halting rounds based on the initial multisets of values from only non-faulty processors, we achieve the best possible halting performance for the original approach to halting. Note that we still cannot discriminate absolutely between good processors and bad, and thus, this bound is also not directly implementable.

Figure 4 demonstrates that there is an appreciable gap between the stopping behavior of the original approximate agreement protocol (the upper line) and both idealized techniques. Note specifically that the separation between the original algorithm’s performance and that of the practical optimal technique is five or six rounds of message passing (and updating) for relatively large values of $\alpha$. That gap translates into the potential for substantial real-time savings via editing the initial multiset. If we could omnisciently edit, i.e., remove only values from faulty processors from the initial multisets, the halting behavior would be exactly that of the practical optimal bound. The separation between the practical optimal bound and the absolute optimal indicates the price we pay for not being able to identify failed processors as the algorithm executes.

![Figure 4: Bounds on Performance](image)

The initial multiset editing approach is based on wild-point editing as outlined below:

**Initial Multiset Editing Protocol**

- Let $\mu_G$ be the mean of the distribution of good sensor values, and let $\sigma_G$ be its standard deviation. Let $V_i = \{v_1, \ldots, v_n\}$ be the multiset of values obtained by processor $P_i$ in the broadcast phase of the initial round of the approximate agreement protocol.
- For all $v_j \in V_i$, if $|v_j - \mu_G| > \beta\sigma_G$, then replace $v_j$ with $\mu_G$.
- Calculate $H_i^1$, based on the edited initial multiset, as specified in the original approximate agreement protocol.

In an operational system, historical data could be used to provide statistical estimates of $\mu_G, \sigma_G$ would most likely be a static property of the sensor technology (a better sensor being represented as a smaller $\sigma_G$, and the threshold parameter $\beta$ would be chosen according to the needs of the application. There is an important trade-off here. The larger the value of $\beta$, the less likely valid data is to be discarded and the slower the rate of convergence. Conversely,
smaller $\beta$ implies faster convergence at the possible expense of an error. Here an error actually means non-convergence; i.e., either the diameter of the final agreement values of non-faulty processors is greater than $\epsilon$ or an agreement value falls outside of the initial range of sensor values read by non-faulty processors.

Figure 5 shows the result of initial multiset editing applied to a four processor system. The threshold parameter, $\beta$, is varied between 1.0 and 3.0. The heavy line is the practical optimal for this system.

![Figure 5: Halting with Edited Initial Multisets](image)

Results of Figure 5 must be evaluated in the light of the possibility of errors which may be produced by the rejection of valid data. Unlike the ineffective adaptive halting procedure, data editing may result in non-convergence—clearly, the possibility of discarding data from a good sensor/processor prevents any formal correctness argument developed for standard approximate agreement. Table 1 shows the worst-case performance for a given choice of $\beta$ in four and seven processor systems. Worst-case performance here is simply the highest percentage of errors for 1000 replications for any $\alpha > 0$.

We make the following two inferences from Figure 5 and Table 1:

- Halting performance which is significantly better than practical optimal ($\beta \leq 1.0$) can be achieved at the expense of a non-negligible error rate.
- With a reasonably large rejection threshold parameter ($\beta = 2$ or $3$), halting performance asymptotically equal to the practical optimal is achieved under the two-faced independent failure model. An experiment in which $\beta = 2.0$ and $\alpha = 0.5$ executed for over three million replications with only two errors.

One point about our editing experiments should be emphasized—the mean of the good value distribution was assumed to be constant. If one intends to utilize wild-point editing in an environment characterized by a time-varying mean of the good value distribution, some care must be taken. If the change in mean is "relatively gradual", then the editing procedure we have presented in this paper will perform satisfactorily. If the mean changes "too much", our procedure will not adapt to those changes (editing will then treat all points as wild points and restore the previous mean). A more conservative approach to editing, but one which seems to be necessary to adapt to quickly changing sensor values, is to allow editing of the data values furthest from the current mean. Quantitative evaluation of this scheme is currently in progress[11].

### Table 1: Errors Induced By Editing

<table>
<thead>
<tr>
<th>System</th>
<th>4 Processor</th>
<th>7 Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta$</td>
<td>Error Rate</td>
<td>Error Rate</td>
</tr>
<tr>
<td>0.5</td>
<td>27.2</td>
<td>11.2</td>
</tr>
<tr>
<td>1.0</td>
<td>4.8</td>
<td>0.6</td>
</tr>
<tr>
<td>1.5</td>
<td>0.7</td>
<td>0.1</td>
</tr>
<tr>
<td>2.0</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>2.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

3. Concluding Remarks

We have presented a model of fault behavior under which data editing can yield substantial performance benefits without introducing appreciable error into the approximate agreement protocol. Under our model, a rejection threshold of two or three standard deviations from the mean of the distribution of good values leads to near optimal performance without introducing appreciable error into the system. We postulate that historical data, accumulated during the operation of a control loop, may be used as a basis for this editing. Clearly, the editing tends to damp the control system somewhat, but a gradual change in sensor value will be accommodated immediately.

We are in the process of the analytic modeling of the editing procedure. This work will allow validation of the simulation results and permit rapid extrapolation into new system contexts. The is-
sue of how one chooses $\epsilon$ and $\beta$ is clearly application specific. We are currently beginning to examine real sensor (accelerometer) data to better understand how that choice impacts upon the end-to-end performance of the control system (e.g., if the loop iteration frequency is sufficiently high, perhaps an occasional agreement error would have no appreciable effect on the performance of the system). The issue of the degree of damping induced by the data editing will also be examined in the light of real sensor data.

We postulate that approximate agreement can be a useful construct in the design and implementation of real-time reliable systems. Further, adapting the basic protocol in response to knowledge about expected input values and the demands of the application is likely to be essential in order to meet performance specifications.

References


