Robust Distributed Computing and Sensing Algorithm

Richard R. Brooks
S. Sitharama Iyengar
Louisiana State University

Sensors that supply data to computer systems are inherently unreliable. When sensors are distributed, reliability is further compromised. How can a system tell good sensor data from faulty? A hybrid algorithm combines proposed solutions to address the problem.

Our modern world contains many automated systems that must interact with changing environments. Because these environments cannot be predetermined, the systems rely on sensors to provide them with the information they need to perform their tasks. Sensors providing data for control systems are the unenviable interface between computer systems and the real world. Programming automated control systems is difficult because sensors have limited accuracy, and the readings they return are frequently corrupted by noise.

To avoid systems being vulnerable to a single component failure, it is reasonable to use several sensors redundantly. For example, an automatic tracking system could use different kinds of sensors (radar, infrared, microwave) that are not vulnerable to the same kinds of interference. Redundancy presents a new problem to system designers because the system will receive several readings that are either partially or entirely in error. It must decide which components are faulty, as well as how to interpret at least partially contradictory readings.

To improve sensor-system reliability, researchers have actively studied the practical problem of combining, or *fusing*, the data from many independent sensors into one reliable sensor reading. When integrating sensor readings, robustness and reliability are crucial properties. It is increasingly obvious that sensor integration, which must include some type of fusion, is necessary to automate numerous critical systems.[1]

Redundant sensors in an automated control system form one type of distributed system. A key advantage of distributed computing is that it adds a new dimension of integrity to computing. Computations made by a network of independent processors are insensitive to a single hardware failure. Instead, the concerns in a distributed system are

- determining how many component failures a network can tolerate and still be reliable and
- how the network separates the output from correctly functioning machines from that of defective machines.

The central question is, how can an automated system be certain to make the correct decision in the presence of faulty data? Much depends on the system's *accuracy*—the distance between its results and the desired results—and on the system's *precision*—the size of the value range it returns.

To solve the problem algorithmically, we basically have *sensor fusion* and *Byzantine agreement*. Danny Dolev[2] presented one Byzantine agreement algorithm to solve the Byzantine generals problem posed by Leslie Lamport and colleagues.[3] The Byzantine generals problem presupposes a distributed decision-making process in which some participants not only

0018-9162/96/$5.00 © 1996 IEEE

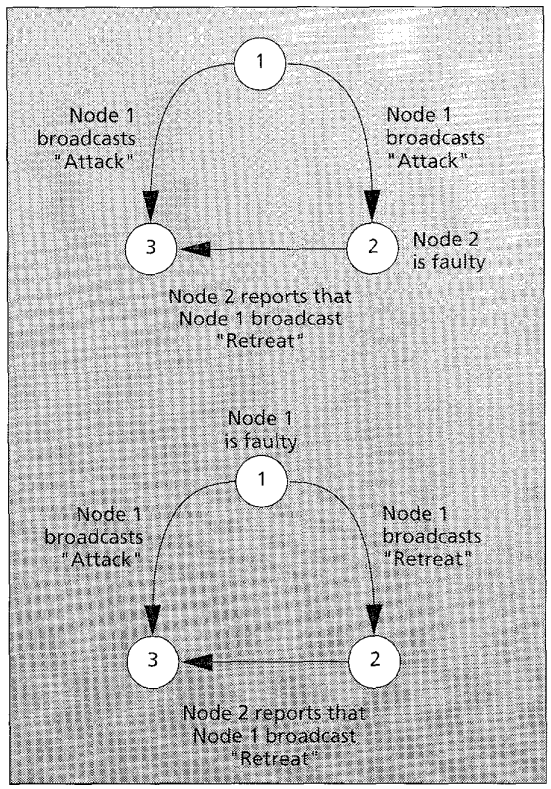

Figure 1. Node 3 cannot distinguish between the two scenarios shown. It is impossible for it to determine if node 1 or node 2 is the faulty node.

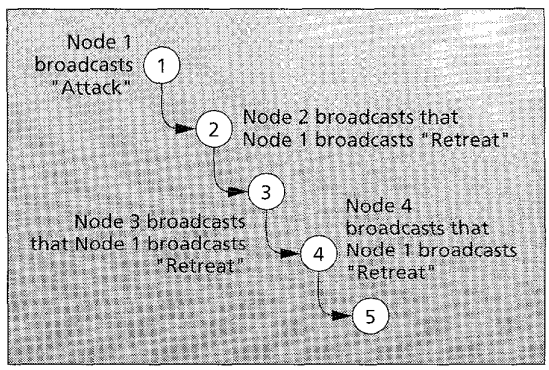

Figure 2. A trivial example of how a faulty node, 2, can defeat Byzantine agreement when the network has connectivity less than $2\tau + 1$. The broadcast from node 1 is modified by node 2 so that nodes 3, 4, and 5 receive the wrong message. One faulty node corrupts the majority of the network.

make the wrong decision but maliciously attempt to force disagreement within the group. An algorithm that solves this problem can reliably be used in distributed computing, because even the failure of a limited number of machines in a network cannot cause the network to malfunction.

In this article, we describe a hybrid algorithm we developed that satisfies both the precision and accuracy requirements of distributed systems. We used established methods for distributed agreement based on data of limited accuracy. The inexact-agreement and approximate-agreement algorithms have been successfully used for implementing clock synchronization protocols and proposed as models for sensor averaging.[4,5] The sensor-fusion algorithm is well established as a method for accurately averaging sensor readings.[6,7] Our hybrid algorithm is suitable for use in both environments and manages to provide increased precision for distributed decision-making without adversely affecting system accuracy.

BYZANTINE GENERALS PROBLEM VARIANTS

A discussion of existing BGP research is beyond the scope of this article. (For an excellent survey on the subject, see Michael Barborak.[8])

Byzantine generals problem

The Byzantine generals problem concerns a mythical siege of a city by the Byzantine army. The army's commander-in-chief has several troops positioned around the city. Each position is under a general's command. The commander-in-chief knows that many of his generals and messengers are traitors who are loyal to the opposing army. He must tell all his generals to either attack or retreat.

The generals can discuss the decision among themselves via messengers. If all loyal armies follow the orders of a loyal commander-in-chief, they stand a good chance of success. If one part of the army attacks while another part retreats, they face certain defeat. How can the loyal generals guarantee, by exchanging messages among themselves, that all generals make the same decision, and that this decision is given by a loyal commander-in-chief?[3]

This problem is directly applicable to distributed computing. It can be rephrased as a system of N independent processing elements (PEs), up to τ of which may be faulty. We must develop a protocol that guarantees for all messages broadcast by any processor X:

- The nonfaulty processors agree among each other on the contents of the data received from X.
- If X is nonfaulty, the agreement should equal the contents of the message sent from X.

This is also called *general interactive consistency*.[9]

This problem has some interesting characteristics.[3] It can be solved only if τ, the number of traitors, is less than one third of N, the total number of PEs. The proof in Figure 1 is done by showing that, in a graph of only three nodes with one faulty node, it is impossible for a correct PE to determine which of the other two nodes is faulty.

Dolev showed that τ must be less than half the connectivity of the graph.[2] This is intuitively evident, as Figure 2 shows: Because a node can change messages passing through it, any part of the graph that receives a majority of messages potentially modified by traitors will be deceived. In other words, to tolerate τ faults, the system must have at least $3\tau + 1$ PEs, and every PE must be connected directly to at least $2\tau + 1$ other PEs.[2] It has been proven as well that an algorithm that solves the Byzantine generals problem must execute at least $\tau + 1$ rounds of broadcasts between nodes.[10]

Many algorithms have been found to solve the Byzantine generals problem. Dolev presented a typical one.[2] We won't present the details of these algorithms here, but it's worth noting that they require each node to rebroadcast all the information it has received. A lower bound of $O(N\tau)$ messages must be broadcast to ensure agreement, and the message size grows exponentially with each round, finishing with size $O(N\tau + 1)$.[8]

Approximate agreement

The approximate-agreement problem posed by Dolev assumes N independent PEs; each one starting with its own real value and all looking for a value within distance ε from the values held by the other PEs. An approximate-agreement algorithm must fulfill the following two requirements:[4]

1. *Agreement.* All nonfaulty PEs must halt with output values within ε of each other.
2. *Validity.* These values must all be contained in the range of initial values held by nonfaulty PEs.

As we mentioned earlier, a system's *accuracy* is the distance between its results and the desired results. A dart player's accuracy is the distance from the bull's-eye to the dart farthest from the bull's-eye. A system's *precision* is the size of the value range it returns. A dart player's precision is the radius of the smallest circle that encloses all darts thrown. Figure 3 illustrates the difference.

The *agreement requirement* corresponds to the system's precision. The algorithm will terminate with each PE possessing a value within the given value ε of all values held by the other PEs. (Note that ε can be arbitrarily small.)

The *validity requirement* corresponds to the system's accuracy. Let's call δ the range of values returned by correctly functioning PEs. As long as the answer returned is within range δ, it is at least as accurate as some correctly functioning PE.

Dolev thus aimed to increase the system's precision. This gain in precision cannot cause the system to become less accurate than the starting value of the least accurate correctly functioning PE.

Dolev[4] took a rigorous, set-theoretical approach to solving the problem, but we can also present the approach intuitively. From the definition of system accuracy, a PE's value is invalid only when outside the allowed accuracy, δ. Since at most τ PEs are faulty, τ or fewer PEs have values that are less than the lower bound of δ. Similarly, there are at most τ PEs whose values are greater than the upper bound of δ. The number of PEs is at least $3\tau + 1$, so discarding the τ smallest and the τ largest values leaves at least $\tau + 1$ values within the range δ. Taking the mean of a subset of these values gives a value that must also fall within δ.

The algorithm[4] for the synchronous approximate-agreement problem is thus:

Algorithm: Approximate-agreement
Input: A set of PEs, each with a value.
Output: A set of PEs, each with a new value converging toward a common value.

Step 1. Each PE broadcasts its value.

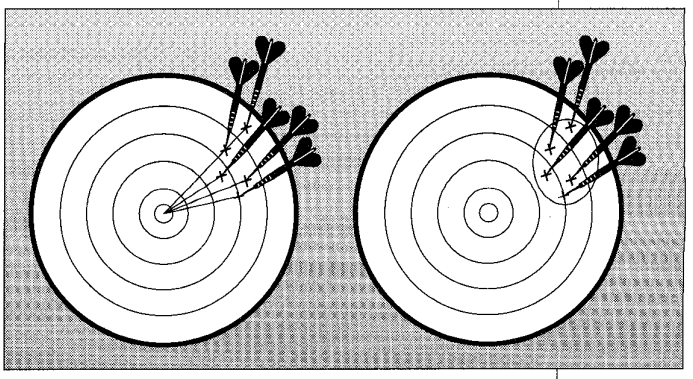

Figure 3. The difference between accuracy and precision. Accuracy is the distance of an answer from the correct answer; precision is the distance between a number of results of the same type.

Step 2. Each PE receives the values from the other PEs and sorts the values into vector $\mathbf{v}$.

Step 3. The lowest τ values and the highest τ values are discarded from $\mathbf{v}$ at each PE.

Step 4. Each PE forms new vector $\mathbf{v}'$ by taking the remaining values $\mathbf{v}[i * \tau]$ where $i = 0,1,...$ (that is, the smallest remaining value and every remaining τ'th value in order.)

Step 5. The new value is the mean of the values in $\mathbf{v}'$.

Dolev also presented an algorithm for the equivalent asynchronous problem.[4] It is virtually identical except that:

- N must be greater than 5τ
- Step 3 discards the 2τ lowest and 2τ highest values
- Step 4 forms $\mathbf{v}'$ by taking the values from $\mathbf{v}[2^* i * \tau]$ where $i = 0,1,...$

Dolev proceeded to show that both algorithms converge and can be forced to converge within any arbitrarily small range ε by performing several iterations.[4]

Alan Fekete improved upon these results with algorithms for three different failure scenarios: crash-fail, failure by omission, and Byzantine failure.[11] The algorithm for Byzantine failure is similar to Dolev's algorithm except that it eliminates some faulty values by verifying that all PEs broadcast the same value to all correctly functioning PEs. This is done by using a Byzantine agreement algorithm like Dolev's.[2] Fekete filtered out the remaining extreme values and calculates the mean of the remaining values. The Fekete algorithm requires $N > 4\tau$ and has a convergence rate superior to the algorithm given in Dolev.[4]

Unfortunately, this improvement requires increased network traffic, so it is therefore doubtful that it can be exploited.[11] Reducing the number of iterations needed for convergence is impractical when the elapsed time needed to perform each iteration increases due to network bandwidth limitations. This is especially true when dealing with sensor data that must be processed in a timely manner.

Inexact agreement

Stephen Mahaney and Fred Schneider have studied the inexact-agreement variant of the Byzantine generals problem.[5] In posing their problem, Mahaney and Schneider explicitly considered both system accuracy and precision. In their context, *precision* is the maximum distance between the values of any two correctly functioning PEs, and *accuracy* is the maximum distance that a PE's value can be from the real value and still be considered correct. (Instead of the symbol δ that Mahaney and Schneider used, we use ε here to represent precision for consistency with the above discussion. Similarly, Mahaney and Schneider used the symbol κ for accuracy, but we use δ.)

An inexact-agreement algorithm will take values from N PEs, up to τ of which may be arbitrarily faulty ($N > 3\tau$) and return a reading on each PE which is:

- within δ of the actual value (accurate)
- within ε of the values on all the other PEs (precise)

Like approximate agreement, these algorithms will converge and can be used iteratively to provide any arbitrary level of precision. This gain in precision could unfortunately result in a loss of accuracy for some PEs.[5]

Note the difference between Mahaney and Schneider's definition of accuracy—*distance from the abstract value being estimated*[5]—and Dolev's—*maximum distance between any two accurate readings at the start of the algorithm*.[4] Dolev's definition is more restrictive than Mahaney and Schneider's because it does not account for skewed readings. As long as acceptable values exist on both sides of the physical value, the two definitions are functionally equivalent. Unfortunately, this is not always the case. Mahaney and Schneider's definition is closer to the physical scenario being modeled and often more relevant, since the discrepancy between data and the value to be measured is of primary interest.

Marzullo pointed out that readings are useful only if they are correct and if the range is small enough.

Mahaney and Schneider presented two algorithms that perform inexact agreement. Both algorithms use sets of "acceptable" values. A value is acceptable if it is within distance δ of $N - \tau$ other values. The problem definition states that we have at most τ faulty PEs, and it is thus evident that any value which is not acceptable cannot be correct.

FAST CONVERGENCE. The *fast convergence algorithm* (FCA) presented by Mahaney and Schneider is performed on each PE:

Algorithm:	Fast convergence algorithm
Input:	A set of PEs, each with a value.
Output:	A set of PEs, each with a new value converging toward a common value.
Step 1.	Each PE receives the values from all other PEs and forms a set **V**.
Step 2.	Acceptable values are put into a set **A**.
Step 3.	e(**A**) is computed.
Step 4.	Any unacceptable values are replaced in **V** by e(**A**).
Step 5.	The new PE value is the average of the values in **V**.

The value e(**A**) can be any of a number of functions on the values stored in **A**. Mahaney and Schneider suggested average, median, or midpoint of the value range as possible choices for e(**A**) that may be appropriate for different applications.

CRUSADERS' CONVERGENCE. Mahaney and Schneider also proposed the *crusaders' convergence algorithm* (CCA). CCA is the same as FCA except that it performs a Byzantine agreement algorithm on set **V** between steps 1 and 2. This extra step increases the algorithm's convergence ratio, similar to Fekete's results. Mahaney and Schneider reported that adding one round of Byzantine agreement increases the convergence ratio enough that one iteration of CCA provides more precision than two iterations of FCA.[5]

Both FCA and CCA are guaranteed to converge if $\tau < N/3$. It is also proven that performance for these algorithms degrades gracefully as long as $\tau < 2*N/3$. In this case *graceful degradation* means the algorithm will either

- Be unable to form an acceptable set and stop.
- Provide answers that are not as good as when $\tau < N/3$ but are within reasonable precision and accuracy bounds.

SENSOR FUSION

Sensors, as we have discussed, are integral to systems that rely on them to perform tasks in changing environments. Unfortunately, sensors are subject to errors, uncertainties, and mechanical failures. To avoid a system's being vulnerable to a single component failure, several sensors are used redundantly, as in our earlier example of an automatic tracking system. However, redundancy poses a new problem because of the inevitable erroneous readings that will occur.

Keith Marzullo[6] proposed a model with *abstract* sensors, which consist of a range of values returned by a sensor, and *concrete* sensors, which are the physical devices that return the value. Because all sensors have limited accuracy, the value returned by an abstract sensor consists of a lower bound and an upper bound. In this way, sensor inaccuracies can be dealt with explicitly.

The sensor-fusion problem we address is: Given a set of N sensors, all with a limited accuracy and at most τ of the sensors being faulty, what is the smallest value range where we can be certain to find the correct value?

As with the approximate-matching algorithms, we have a set of N real values whose correctness depends on their being within a certain distance of the unique correct value, as well as up to τ values that may be arbitrarily in error. Each sensor is represented by the upper and lower bound it gives for the value it measures. The size of this range is the accuracy, δ, for that sensor. It is often advantageous to use sensors of different types; therefore, it is not assumed that δ is uniform for the sensors, and the algorithm makes no precision restraints.

Marzullo pointed out that readings are useful only if they are correct and if the range is small enough. This is why a sensor is considered correct only when the range it

returns is of limited size and contains the physical value being sought. On the basis of these definitions and restrictions, Marzullo has proven that it is possible to find the smallest region containing the physical variable. This region's size will be less than or equal to the largest accuracy of any component sensor.

Finding all regions in which $N - \tau$ sensor readings intersect provides a correct solution to the sensor-fusion problem. It's easy to see that the correct value must be in one of these intervals. The correct range is thus defined by the value of the smallest lower bound and the largest upper bound of these intersections.[6]

Our optimal-region algorithm modifies Brooks' and Iyengar's[12] multidimensional algorithm and is roughly equivalent to Marzullo's[6] algorithm:

Algorithm: Optimal region
Input: A set of sensor readings S.
Output: A region describing the region that must be correct.

Step 1. Initialize a list of regions, which we will call C, to NULL.
Step 2. Sort all points in S into ascending order.
Step 3. A reading is considered active if its lower bound has been traversed and its upper bound has yet to be traversed. Work through the list in order, keeping track of active readings. Whenever a region is reached where $N - \tau$ or more readings are active, add the region to C.
Step 4. All points have been processed. List C now contains all intersections of $(N - \tau)$ or more sensor readings. Sort the intersections in C.
Step 5. Output the region defined by the lowest lower bound and the largest upper bound in C.

This algorithm can be implemented for a distributed sensor network, where all sensors (PEs) fuse information in parallel. In a distributed environment, the algorithm's precision is bounded by the accuracy.

Precision and accuracy

Approximate-agreement and sensor-fusion algorithms both accept data from many independent sources, where a minority of the data is arbitrarily incorrect, and the correct data does not agree perfectly. Both try to find a result that accurately represents the data from the correct PEs, while minimizing the influence of data from faulty PEs.

Approximate agreement emphasizes precision, even when this conflicts with system accuracy. For both approximate and inexact agreement, the overall accuracy of the readings remains constant and can decrease for some PEs. This approach is justified when processor coordination is more important than the quality of the results. For example, these algorithms are useful for clock synchronization among PEs.

Sensor fusion, on the other hand, is concerned solely with the accuracy of the readings, which is appropriate for sensor applications. This is true despite the fact that increased precision within known accuracy bounds would be beneficial for coordinating distributed automation systems.

Any desired level of precision can be attained by iterating the approximate-agreement algorithm. The accuracy requirements of inexact agreement could eventually make it impossible to perform further iterations, putting a lower limit on the precision attainable. It's impossible to achieve a higher precision or accuracy by iterating the sensor-fusion algorithms.

Any desired level of precision can be attained by iterating the approximate-agreement algorithm.

By allowing each PE to have its own accuracy, the sensor-fusion algorithm allows the use of heterogeneous sensors. The bandwidth needed for one iteration of all algorithms not containing Byzantine agreement is linear in the number of processors. If a Byzantine preprocessing step is added,[5,11] the number of bits to be broadcast will be exponential in the number of rounds R needed for Byzantine agreement on the values, $O(N^R + 1)$ where R will be greater than $\tau + 1$.[11] Some algorithms perform agreement with a polynomial number of bits but a larger number of rounds, making performance less than asymptotically optimal.[11] For this reason, algorithms with a Byzantine agreement step are inappropriate for real-time systems.

The complexity of the sensor-fusion and FCA algorithms is $O(N\log N)$. This comes from sorting the N values into increasing order. It is interesting that Mahaney and Schneider's "acceptability" criteria correspond to PEs whose values intersect with the values of at least $N - \tau$ other PEs. Marzullo's algorithm and the sensor-fusion algorithm can be trivially modified to determine if a PE is "acceptable." This is the key idea behind the hybrid algorithm we present.

One iteration of Dolev's approximate-agreement algorithm requires finding the τ largest and τ smallest of N values. This can be done by finding and removing the maximum and minimum of the N values τ times, or by sorting the set in increasing order. The algorithm can therefore be implemented in $\min[O(N\tau), O(N\log N)]$ time, depending on whether $\tau < \log N$.

BROOKS-IYENGAR HYBRID ALGORITHM

To satisfy the requirements of both the inexact-agreement problem and the sensor-fusion problem, we merged the optimal region algorithm with FCA to produce an algorithm that provides the best accuracy possible and increases the precision of distributed decision-making.

The data used by this algorithm can take any of the following three forms:

- Real values, all with the same implicit accuracy.
- Real values, along with an explicit accuracy value transmitted along with the value.
- A range consisting of an upper and lower bound.

The algorithm is:

Algorithm: Brooks-Iyengar hybrid
Input: A set of data S.

Table 1. S5 values broadcast to other sites.

	S1	S2	S3	S4
Value	4.7 ± 2.0	1.6 ± 1.6	3.0 ± 1.5	1.8 ± 1.0
S5 Value	3.0 ± 1.6	1.0 ± 1.6	2.5 ± 1.6	0.9 ± 1.6

Output: A real number giving the precise answer and a range giving its explicit accuracy bounds.

Step 1. Each PE receives the values from all other PEs and forms a set **V**.

Step 2. Perform the optimal region algorithm on **V** and return a set **A** consisting of the ranges of values where at least $N-\tau$ PEs intersect.

Step 3. Output the range defined by the lowest lower bound and the largest upper bound in **A**. These are the accuracy bounds of the answer.

Step 4. Sum the midpoints of each range in **A** multiplied by the number of sensors whose readings intersect in that range, and divide by the number of factors. This is the answer.

The asymptotic complexity of this algorithm, like FCA and the sensor-fusion algorithm, is $O(N\log N)$. This is due to step 2, which requires sorting the input by its lower bounds. Steps 3 and 4 work on ranges where $N-\tau$ or more values intersect. Since there are at most $2\tau + 1$ such ranges,[12] step 3 has complexity $O(\tau\log\tau)$, and step 4 has $O(\tau)$, both of which are less than $O(N\log N)$.

As with Marzullo's sensor-fusion approach, the hybrid algorithm treats all data as a range of possible values instead of as a point value. Since the problem concerns data with accuracy and precision limitations, computations based on point values can be misleading. This difference is useful conceptually and can also be advantageous in increasing the precision and accuracy of the results.

Our algorithm's accuracy is identical to that of the optimal region algorithm. Since the real value computed in step 4 must be within this range, it is within the accuracy bounds defined by Mahaney and Schneider[5] and by Marzullo.[6] The accuracy bound returned by the algorithm will necessarily contain Dolev's accuracy constraint as a subset.

The precision analysis Mahaney and Schneider performed is valid for the hybrid algorithm when uniform accuracy is assumed. The values will converge, despite arbitrary errors, at a rate of at least $2\tau/N$ per iteration.[5] This convergence must be kept as a lower bound, since the worst-case scenario of this algorithm is identical to Mahaney and Schneider's [5] FCA algorithm. If uniform accuracy is not assumed, it is impossible to define bounds on the convergence rate, short of it being less than 1, since we are dealing with a set of arbitrarily distributed numbers.

In many cases, however, the hybrid algorithm provides superior results to FCA. FCA computes its value based on a simple function of all values that fulfill the acceptability requirement that the value be within distance δ of $N-\tau-1$ other readings. Note that a system with four PEs can tolerate one faulty PE; the values the faulty PE sends to other nodes could differ by up to 4δ and still be acceptable. The different values are then entered directly into the function $e(\mathbf{A})$, which computes the result at each node.

In contrast to FCA, the hybrid algorithm uses the midpoint of the region where the faulty readings intersect with at least $N-\tau-1$ correct readings. Using the example of a system with four PEs, the value transmitted to two different nodes by the faulty PE can still vary by up to 4δ and be acceptable. The value used in calculations by the hybrid algorithm, however, differ by at most 2δ.

The hybrid algorithm also weights each region by the number of PEs that agree on the region. This increases precision by increasing the influence of PEs that intersect more than one set of PEs. A faulty answer is most disruptive when it barely fulfills the acceptability requirement and forms one intersection with $N-\tau-1$ other PEs. When a PE's value contributes to many acceptable regions, it's closer to the value that the hybrid algorithm will return. Weighting the region midpoints exploits this fact to increase the algorithm's convergence.

Note that it would also be possible to insert another step between steps 1 and 2 that performs Byzantine agreement among the PEs. This would correspond to Mahaney and Schneider's CCA algorithm. While this would increase the rate of convergence of the answer, it is unlikely that this increase would justify the network bandwidth needed.

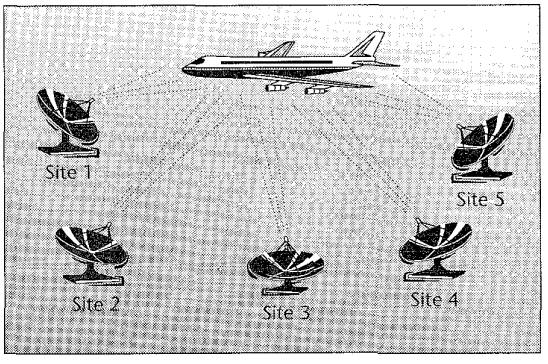

Figure 4. Multiple ground radar stations are detecting a target's position. To coordinate actions, you need a single accurate reading.

COMPARING ALGORITHM PERFORMANCE

Figure 4 depicts a scenario we can use to show the differences among algorithms. Multiple ground radar stations are detecting a target's position. To coordinate actions, we want to have a single accurate reading.

We have $N = 5$, and therefore τ cannot be greater than 1. Four sites, S1, S2, S3, and S4, function correctly. The fifth site, S5, is faulty; it broadcasts different values to each of the other four sites. Table 1 shows the values.

The four correct sites agree on the range [2.7...2.8], so the actual value must lie within this range.

Table 2. Differences between algorithms.

Algorithm	Dolev	Fekete	FCA	CCA	Sensor Fusion	Brooks-Iyengar Hybrid
Faulty PEs tolerated	$< N/3$	$< N/4$	$< N/3$	$< N/3$	$< N/2$	$< N/3$
Maximum faulty PEs	$< N/3$	$< N/4$	$< 2N/3$	$< 2N/3$	$< N/2$	$< 2N/3$
Complexity	$N\tau$	N/A	O (N log N)	N/A	O (N log N)	O (N log N)
Order of network bandwidth	$O(N)$	$O(N^{**}(\tau+1))$	$O(N)$	$O(N^{**}(\tau+1))$	$O(N)$	$O(N)$
Convergence rate	$1/\lfloor N-2\tau-1 \rfloor+1)$	$1/\lceil (N-2\tau)/\tau \rceil$	$2\tau/N$	τ/N	2*accuracy	$2\tau/N$
Accuracy	δ	δ	$\delta + \varepsilon\tau/N$	$\delta + \varepsilon\tau/N$	limited by input	limited by input
Iterate for precision	yes	yes	often	often	no	often
Precision over accuracy	yes	yes	yes	yes	no	no
Accuracy over precision	no	no	no	no	yes	no

Dolev's algorithm

Since τ is 1, Dolev's approximate-agreement algorithm discards the highest and lowest value at each PE and averages the remaining values.

- **S1:** Discard 1.6 and 4.7. Averaging the remaining values gives an answer of 2.6.
- **S2:** Discard 1.0 and 4.7. Averaging the remaining values gives 2.13.
- **S3:** Discard 1.6 and 4.7. Averaging the remaining values gives 2.43.
- **S4:** Discard 0.9 and 4.7. The problem becomes identical to the one for S2, and the answer is again 2.13.

Mahaney and Schneider's FCA algorithm

All values sent by S5 are within their accuracy range of an intersection with at least three other sites. The values from S5 will not be disqualified for use by any site. The answer for each site is a simple average of all five values present at that site.

- **PE S1:** Calculates a value of 2.82.
- **PE S2:** Calculates 2.42.
- **PE S3:** Calculates 2.72.
- **PE S4:** Calculates 2.4.

Optimal region sensor-fusion algorithm

This algorithm uses the ranges defined by the uncertainties of each site, including the faulty S5.

- **S1:** Four PEs intersect in range [1.5..2.7]. Five PEs intersect in [2.7..2.8]. Four PEs intersect in [2.8..3.2]. The correct answer must therefore lie in [1.5..3.2].
- **S2:** Four PEs intersect in range [1.5..2.6], and four PEs intersect in [2.7..2.8]. The correct answer must be in [1.5..2.8].
- **S3:** Four PEs intersect in range [1.5..2.7]. Five PEs intersect in [2.7..2.8]. Four PEs intersect in [2.8..3.2]. The correct answer must therefore lie in [1.5..3.2].
- **S4:** Four PEs intersect in range [1.5..2.5], and four PEs intersect in [2.7..2.8]. The correct answer must be in [1.5..2.8].

Brooks-Iyengar hybrid algorithm

The ranges the sensor-fusion algorithm finds are the accuracy limits the hybrid algorithm returns for each site. To find the answer for a site, the hybrid algorithm makes a weighted average of the midpoints of the regions found by the sensor-fusion algorithm.

- **S1:** The weighted average is: (4 * 2.1 + 5 * 2.75 + 4 * 3.0) / 13 = 2.625.
- **S2:** The weighted average is: (4 * 2.05 + 4 * 2.75) / 8 = 2.4.
- **S3:** The weighted average is: (4 * 2.1 + 5 * 2.75 + 4 * 3.0) / 13 = 2.625.
- **S4:** The weighted average is: (4 * 2.0 + 4 * 2.75)/ 8 = 2.375.

Results

All four algorithms solve the problem they are designed for within the accuracy and precision bounds the algorithm designers used. Note that the accuracy bounds the sensor-fusion and hybrid algorithms returned all contained the range [2.7..2.8], where the actual value is located.

All three algorithms designed for improving precision—Dolev's, Mahaney and Schneider's, and our hybrid—returned answers within a narrower range than the input data. For this example, the hybrid algorithm's output was the most precise, with all answers lying within a range of length 0.25. Table 2 summarizes the differences between the algorithms.

THE HYBRID ALGORITHM WE'VE DEVELOPED effectively solves the problem of making the correct decision in the presence of faulty data. This is important for several reasons:

- Because the same algorithm can enhance both accuracy and precision, many real-world distributed applications can use one unified general algorithm.
- The derivations of sensor fusion and approximate agreement are independent. One is based on set theory; the other, on geometry. It thus produces two explanations of the same problem, which makes the solution more easily understood.

- The combined algorithm can be extended to solve problems in many application areas.

Use of this algorithm is not limited to the applications we've cited. All floating-point computations have limited accuracy that differs from machine to machine. The hybrid algorithm affords scientific computing increased reliability by letting calculations be performed on a distributed system comprising heterogeneous components. This, then, provides a method more resistant to round-off and to skewing errors resulting from hardware limitations.

Software reliability is a growing concern. Programs written independently for different hardware platforms that produce equivalent output create a situation like the tracking system example with multiple sensor input. It's safe to assume that many independent programming teams would not make exactly the same programming error, and that of N such programs fewer than one third or one half would be incorrect for a given instance.

This is analogous to using many sensors based on different technologies—thus sensitive to different types of noise and interference—to measure the same physical entity. The failings of one hardware component will be compensated for by the use of another component made in a complementary manner. These assumptions, used to justify N-modular redundancy for hardware systems, could be valid for software.

Critical modules of important systems require extra effort to ensure their reliability. Reliability requires both precision and accuracy. Unfortunately, there is often a trade-off to be made between the two. The hybrid algorithm we've described allows for increased precision, without sacrificing accuracy in the process. The algorithm lets distributed systems converge toward an answer that lies within precisely defined accuracy bounds. With this algorithm, truly robust distributed computing applications can be developed. ■

Acknowledgments

We thank the anonymous reviewers whose comments greatly improved the presentation. This work was supported in part by the Office of Naval Research, grant N00014-94-1-0343.

References

1. S.S. Iyengar, L. Prasad, and H. Min, *Advances in Distributed Sensor Technology*, Prentice-Hall, Englewood Cliffs, N.J., 1995.
2. D. Dolev, "The Byzantine Generals Strike Again," *J. Algorithms*, 1982, pp. 14-30.
3. L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, July 1982, pp. 382-401.
4. D. Dolev et al., "Reaching Approximate Agreement in the Presence of Faults," *J. ACM*, July 1986, pp. 499-516.
5. S. Mahaney and F. Schneider, "Inexact Agreement: Accuracy, Precision, and Graceful Degradation," *Proc. Fourth ACM Symp. Principles of Distributed Computing*, ACM Press, New York, 1985, pp. 237-249.
6. K. Marzullo, "Tolerating Failures of Continuous-Valued Sensors," *ACM Trans. Computer Systems*, Nov. 1990, pp. 284-304.
7. P. Chew and K. Marzullo, "Masking Failures of Multidimensional Sensors," *Proc. 10th Symp. Reliable Distributed Systems*, IEEE CS Press, Los Alamitos, Calif., 1991, pp. 32-41.
8. M. Barborak, M. Malek, and A. Dahbura, "The Consensus Problem in Fault Tolerant Computing," *ACM Computing Surveys*, June 1993, pp. 171-220.
9. T. Krol, "(N, K) Concept Fault Tolerance," *IEEE Trans. Computers*, Apr. 1986, pp. 339-349.
10. M. Fisher and N. Lynch, "A Lower Bound for the Time to Insure Interactive Consistency," *Information Processing Letters*, June 1982, pp. 183-186.
11. A. Fekete, "Asymptotically Optimal Algorithms for Approximate Agreement," in *Distributed Computing*, Springer-Verlag, Berlin, 1991, pp. 9-29.
12. R. Brooks and S.S. Iyengar, "Optimal Matching Algorithm for Multi-Dimensional Sensor Readings," *SPIE Proc. Sensor Fusion and Advanced Robotics*, SPIE, Bellingham, Wash., 1995, pp. 91-99.

Richard Brooks *is a PhD candidate in computer science at Louisiana State University. He has worked on projects with Goddard Space Flight Center, Radio Free Europe/Radio Liberty Munich, and the French stock exchange authority. As a consultant for the World Bank, he helped implement its network in Africa, Eastern Europe, and Central Asia.*

Brooks earned a BA in mathematical sciences from Johns Hopkins University and studied operations research and computer science at the Conservatoire National des Arts et Metiers, Paris. He is a member of ACM and the Institute for Operations Research and Management Science.

S. Sitharama Iyengar *is chair of the Computer Science Department and professor of computer science at Louisiana State University, where he directs the Robotics Research Laboratory. He has been actively involved in research in high-performance algorithms and data structures for more than 20 years. He has served as principal investigator on research projects supported by the Office of Naval Research, NASA, the National Science Foundation, the Jet Propulsion Laboratory, the Department of the Navy, the Department of Energy, the Louisiana Education Quality Support Fund Board of Regents, and Apple Computer.*

Iyengar received a PhD in engineering from Mississippi State University and an MS from the Indian Institute of Science. He has served as guest editor for numerous publications and is currently a series editor for Neuro Computing of Complex Systems *and area editor for* Journal of Computer Science and Information. *Iyengar is an IEEE fellow and a Distinguished Visitor of the IEEE Computer Society (1995-1998).*

Readers can contact Brooks at Louisiana State University, Dept. of Computer Science, Baton Rouge, LA 70803-4020; rrb@bit.csc.lsu.edu.