CHAPTER 4

Voting: A Paradigm for Adjudication and Data Fusion in Dependable Systems

BEHROOZ PARHAMI

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560 (E-mail: parhami@ece.ucsb.edu).

Voting is used in realizing ultrareliable systems based on multichannel computation. In practical applications to date, the multiple computation channels have consisted of identical hardware circuits, which are expected to produce the same outputs, or diverse software modules, which may produce inexact or incomplete results. The challenge in hardware voting is the choice of adjudication points where voters must be inserted, and the associated synchronization and data throughput requirements. In software voting, algorithms for reconciling multiple inexact or incomplete results, and their time and code complexities, are important aspects of the problem. Chapter 4 deals with weighted voting algorithms and their hardware and software realizations, as a unifying methodology for a variety of voting schemes that have been found to be useful in practice. As in all algorithm design problems, correctness and performance of voting algorithms are key properties to evaluate and ascertain. We also show, through examples, that there is more to voting than the simple majority or plurality viewpoint heretofore dominating the dependable computing literature. It is argued that considering voting as a form of data fusion, and drawing from both mathematical studies of the anomalies in voting as well as sensor fusion work in signal processing, can be beneficial to further developments in the field of dependable computing.

4.1 INTRODUCTION

Voting is important in many different contexts. It is used, explicitly or implicitly, in the fusion of data originating from multiple sources [19], for realizing ultrareliable

Dependable Computing Systems. Edited by Hassan B. Diab and Albert Y. Zomaya
Copyright © 2005 John Wiley & Sons, Inc.

systems based on the multichannel computation paradigm [41], as a component of consistency and agreement algorithms in distributed systems [16, 56], in distributed self-diagnosis [27], for improved learning [14], and as a classification tool with neural networks [7]. The use of voting to obtain highly reliable data from multiple unreliable versions was suggested by John von Neumann half a century ago [60]. Since then, the concept has been practically utilized in fault-tolerant computer systems and has been extended and refined in many different ways. Reliability modeling of voting schemes by considering compensating errors [55], handling of imprecise or approximate data [3], combination with standby redundancy [35], voting on digital *signatures* obtained from computation states [28] so as to reduce the amount of information to be voted on, and adapting vote weights based on a priori reliability data [50] constitute some of these extensions and refinements in the history of voting.

The bulk of the literature on voting in dependable systems pertains to the derivation of an integer or real numerical result based on values offered by several computation channels. With integers belonging to a small set, this is much like voting in a political election. For example, possible conclusions in evaluating the radar image of an aircraft may be "civilian" (0), "fighter" (1), or "bomber" (2). If three evaluation units independently process the available data and arrive at the conclusions $\langle 1, 1, 2 \rangle$, then the presence of a fighter plane may be assumed based on a majority rule; that is, candidate "1" wins a majority of the vote. Voting with real data, or, more generally, with a large output space, is more complicated and has no direct counterpart in political elections. For example, there is no strict majority if three measurements of the distance to an approaching aircraft yield $\langle 12.5, 12.6, 14.0 \rangle$ in kilometers, even though based on the fairly good agreement between the first two numbers, an estimate of 12.55 km may be deemed as the "voting" result.

To facilitate and systematize the study of voting in diverse areas, a unified high-level view is essential. We categorize voting schemes according to implementation in hardware or software (*voting networks* or *voting routines*) and based on the size and structure of their input/output spaces. A *voting algorithm* [42] specifies how the voting result is obtained from the input data and may be the basis for implementing a voting network or a voting routine. We present an overview of voting networks and algorithms for dependable systems in Section 4.2. The choice of *voting scheme* (algorithm, plus implementation strategy) has important implications for the cost, speed, and reliability of the voting process and, ultimately, affects the respective parameters of the system into which voting is embedded. Understanding the complexity and performance of voting schemes is important in choosing the least costly or fastest algorithm that satisfies the specified correctness and reliability requirements of the system being designed.

4.2 VOTING IN DEPENDABLE SYSTEMS

Use of voting in dependable systems has a long history, and von Neumann's work [60] is generally considered the beginning of research efforts in hardware voting.

Subsequently, voting was applied as a mechanism for reconciling inaccurate or conflicting results produced by multiple diverse versions of a software module [3]. In this section, we review issues in hardware and software voting and provide a unified framework that can be applied to both.

4.2.1 Hardware Voting

The simplest voter produces an output from n bit-inputs, where all inputs are treated as equals. Greater in sophistication are word voters in which input words may be associated with weights reflecting their varying reliabilities. The general structure of a hardware voter with n inputs and one output, along with their associated votes, is shown in Fig. 4.1. In this subsection, we review some issues for hardware voters, without repeating the extensive background information, justifications, and design alternatives provided in Ref. [38].

A bit-voter is quite easily implemented using a variety of design strategies. We first focus on the design of w-out-of-n bit-voters whose output is 1 when w or more of the inputs are 1s. We have $w = \lfloor (n+1)/2 \rfloor$ for *majority* bit-voters and $w > \lfloor (n+1)/2 \rfloor$ for *supermajority* bit-voters. The case $w \leq n/2$ makes sense in a fail-safe environment where the probability of producing an incorrect output of 1 by a module is significantly smaller than the probability of an incorrect 0 output. The design of a w-out-of-n bit-voter is essentially a parallel counting problem, which has been extensively studied in computer arithmetic due to applications in fast multiplier designs [46]. An n-input parallel counter supplies the $\lceil \log_2(n+1) \rceil$-bit sum of the n input bits as an unsigned binary number. Comparing this sum to the threshold w and deciding whether the threshold has been matched or exceeded is an easy matter.

In fact, with a fixed threshold w, the two-stage design outlined in the preceding paragraph can be simplified through *merged arithmetic* design [46] for detecting the sign of $w - \sum_{i=1}^{n} x_i$. For example, designing a 4-out-7 voter requires only that the most-significant bit of the 3-bit sum be produced, and this bit serves as the output without a need for comparison. More generally, a *saturating counter* [26], which need not be capable of producing exact counts exceeding w, may be employed. Gate networks, multiplexer-based design, and construction based on decomposition

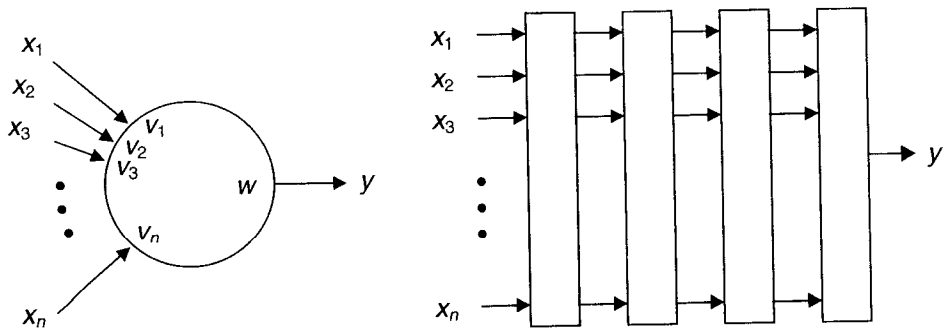

Fig. 4.1 Schematic diagram of a hardware voter and its pipelined implementation.

are some of the other possible strategies for implementing w-out-of-n bit-voters [38]. All such designs are readily extended to the case of weighted voting. For example, the parallel counting method will be based on adding the votes associated with 1 inputs rather than the inputs themselves. The resulting multioperand addition scheme [46] will be based on input operands that are formed by fanning out the input bits (fixed votes) or ANDing the input bits with arbitrary votes supplied as inputs.

Hardware word-voters are represented by the same diagrams as in Fig. 4.1, except that each signal line now represents k bits, where k represents width. The simplest word-voters take k separate votes on the bits within the words, either in a parallel structure or sequentially in time through a single bit-voter. This, however, produces the correct voting output only under certain conditions. For example, with 3-input majority voting, such a voter would produce the incorrect output of 11 when presented with the 2-bit input words 01, 10, and 11. However, if a majority does exist, then independent bit voting does in fact produce the correct majority result for word inputs. Note that, in word-voting, the k bits of each input are presumed to represent, collectively, an encoding of a single value. If the k-bit words are obtained by simply juxtaposing k independent true/false or yes/no data elements (such as binary properties of a target being tracked), then using k bit-voting operations may in fact be the right thing to do.

Note that with bit inputs, a majority always exists. Therefore, the only problem in a majority bit-voter is determining whether more than half of the inputs are 1s; if not, then 0 is output. For a word voter, on the other hand, none of the input values may be in the majority; in such cases, we may use *plurality voting* to determine which input value appears more often than all of the others. Thus, word-voting is more complicated than bit-voting. It is known that, with a totally ordered input space, voting has the same time and circuit complexity as sorting and that the requisite circuit can be implemented by augmenting a sorting network with vote-combining and max-selection stages, with the time and cost complexity of the sorting stage being dominant [38]. Just as was the case for the weighted version of bit-voting, the weighted version of word-voting is not fundamentally more difficult than nonweighted word-voting. It is, however, the case that threshold voting is fundamentally simpler than plurality voting [40]; thus, the former should be chosen over the latter whenever the application allows it.

4.2.2. Software Voting

Although it is quite possible to implement the voting schemes of Section 4.2.1 in hardware or software, the latter is often used when the process of adjudication for obtaining an output from multiple inputs is more complicated than that in simple majority or plurality voting. Despite the added complexity, we still call the process "voting" under our extended definition of the term. We will see later that voting, as studied by mathematicians and social scientists, is in fact much more than the majority or plurality schemes commonly associated with political elections. It is in this spirit that we use the term "voting" (see Sections 4.2.3 and 4.5.1).

One hallmark of software voting is that there are often indicators of result reliabilities that can be taken into account in obtaining the adjudged result [13]. Therefore, the output of the voting process is not only a function of input values but also depends on ancillary or peripheral information. Such information may be derived from past histories of modules providing the inputs, results of prescreening or acceptance tests, special flags supplied to indicate self-evaluation by modules, fail-safe or *exception* outputs, and knowledge of correlations between the results under special circumstances. Using these indicators and the input values (or indicators of agreement and disagreement between various inputs), a decision process is used to derive the output result. The said result may be one of the input values or a compromise value, such as the mean or median of a selected subset of input values [32, 42]. The goal of adjudication in this context is often to maximize the probability of producing the correct output or to minimize the risk [8, 13], the latter being of interest when various incorrect output values are associated with different costs or penalties.

As far as run-time resource utilization is concerned, table-based adjudication is the most efficient scheme; it is also the most general. The table lists, for each possible combination of values for the various indicators, the appropriate output or the rule to be used for computing the output. The table may be represented in alternate forms, such as coteries [15] or Boolean expressions [25]. Consider for example three sensors, each of which detects the occurrence of an event with one of three degrees of confidence. Using a, b, and c for the three sensors and subscripts 1, 2, and 3 for degrees of confidence (3 being the highest), the adjudged detection output f may be specified by the following Boolean expression:

$$f = a_1b_1c_1 \vee a_2b_2 \vee b_2c_2 \vee a_3c_3$$

This Boolean expression essentially specifies that the event's detection is signaled if all three sensors agree with low degree of confidence, if a and c agree with the highest degree of confidence, or if b agrees with a or c with medium degree of confidence. The greater confidence level required of a and c, when b disagrees with them, may be due to our knowledge that a and c have a common weakness that makes them prone to correlated failures, leading to false alarms. Figure 4.2 depicts the same decision process in the form of *quorums*, or subsets

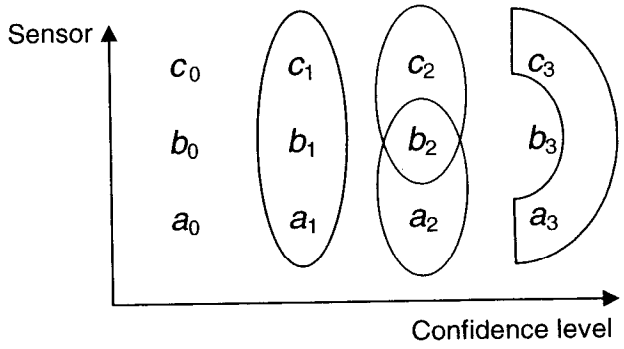

Fig. 4.2 Quorums for the three-sensor reliable detection problem.

needed for a positive decision. The confidence level 0 has been added to Fig. 4.2 to denote nondetection.

From the viewpoint of the relationship of the output to inputs, the voting process can be categorized as selection or compromise. Selection voting implies that one of the finite or infinite set of possible output values is selected as the voter output. This is based on the notion of a majority or plurality of inputs "supporting" the chosen output (the notion of support will be discussed later; often, it means exact or approximate matching of values). Compromise voting, on the other hand, allows flexibility in computing an output based on all, or a selected subset of, inputs. Compromise voting is exemplified by the (generalized) median or mean selection rule from a set of candidate inputs. Generalized median rule refers to the process of eliminating pairs of data points that are furthest apart (using an arbitrary distance metric), until only one or two data points remain. Figure 4.3 illustrates the process on a Euclidian plane, where the set of candidates is initially selected to be a maximal set of proximate data points.

4.2.3 A General Framework

To allow systematic study of the voting methods discussed in Sections 4.2.1 and 4.2.2, as well as many other schemes, we present the following definition of weighted voting. Given n input data objects $x_1, x_2, \ldots, x_n$, and associated nonnegative real votes (weights) $v_1, v_2, \ldots, v_n$, with $\sum_{i=1}^{n} v_i = V$, (generalized) *weighted voting* aims to compute the output y and its vote w such that y is "supported by" a number of input data objects with votes totaling w, where w satisfies a condition associated with the voting subscheme; e.g., $w > V/2$ for majority voting, $w \geq t$ for t-out-of-V (generalized w-out-of-n) voting, and w corresponding to maximal support among all possible outputs in the case of plurality voting. Figure 4.1 depicts the elements of this definition.

A number of features in the definition above are noteworthy. First, we speak of input objects, not input values. This is because it is possible, and indeed quite useful, to deal with voting on composite or nonnumeric data. Second, we introduce the

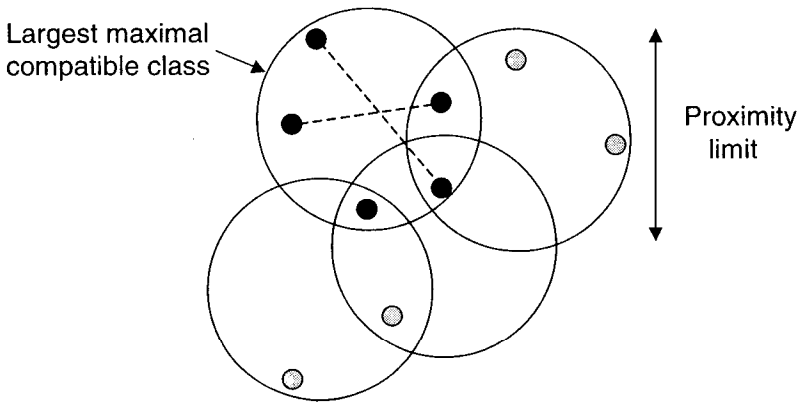

Fig. 4.3 Generalized median voting on a subset of nine data points. Once the furthest pairs of points are removed from the subset, the sole remaining point is the output.

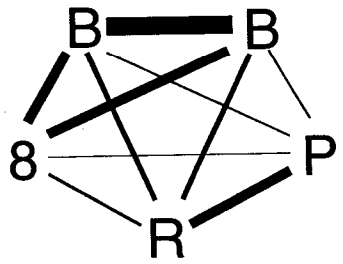

Fig. 4.4 Results from five character recognizer and mutual support among their conclusions (heavier lines indicate greater levels of support).

notion of input objects "supporting" the output y. In the simplest case, an input supports a chosen output if the two are equal. However, approximate equality, as well as more complex notions of support, may be envisaged. An example is depicted in Fig. 4.4 where five character recognizers draw independent conclusions with varying degrees of mutual support. For each possible output, such as "B," the total level of support can be derived and the results used in the final selection.

Despite its generality, as illustrated in Fig. 4.4, our weighted voting model does not cover all possible adjudication schemes. As an example, consider a system with four sensors placed at the corners of a square area. The system should signal an event that can only occur outside the square area, if and only if any two adjacent sensors indicate the event but not if two diagonally opposite sensors do so. It is easy to see that no assignment of votes allows us to emulate this decision process through weighted voting. When weighted voting is a feasible implementation alternative, the assignment of weights to the various inputs can be nontrivial [6].

It is our contention, however, that weighted voting, as defined above, is general enough to provide a useful unified framework for the study of voting in data fusion and dependable computation. Such a unified framework allows the use of a common set of algorithms that have been analyzed and optimized for correctness and efficiency. As an example, we show how the adjudication scheme of Fig. 4.2 can be formulated in this manner. Figure 4.5 depicts a possible assignments of votes and a threshold to convert the problem into weighted threshold voting.

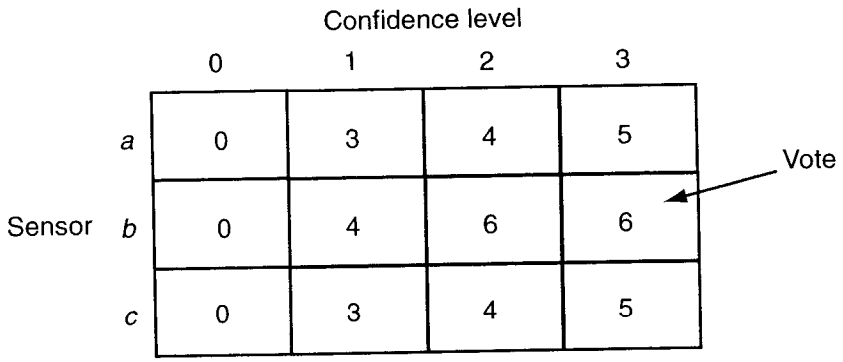

Fig. 4.5 Possible vote assignments to implement the outcome $f = a_1b_1c_1 \vee a_2c_2 \vee b_2c_2 \vee a_3b_3$ as weighted threshold voting with a threshold of 10.

A relevant idea in discussing what can and cannot be implemented as weighted voting is that of coteries [15]. A coterie is simply a collection of subsets (quorum groups) of the units that participate in a decision process such that no subset is a superset of another and any two subsets intersect. In the context of distributed systems, mutual exclusion can be guaranteed if critical operations require permission from all members in any subset within a coterie. Because any two subsets intersect, there is no way that another permission can be granted. The notion has been generalized to read and write coteries so as to ensure high-performance and reliable read/write operations with replicated data. For example, with 9-way replication, one can view the replicas as being logically arranged in a 3×3 array. If each write operation updates at least three replicas that are in the same row and each read accesses at least three replicas located in the same column, then the replicas read are always guaranteed to contain the latest information.

At this point, two items of bad news can be revealed. First, coteries are more general than weighted voting in the sense that there exist coteries that one cannot implement as weighted voting. Second, adjudication processes of interest in dependable computing and data fusion are even more general than coteries, in the sense that the requirement for intersection of the subsets may be dropped: if an alarm is to be activated due to unsafe conditions, we do not really care whether several subsets independently arrive at this conclusion. Figure 4.2 supplies an example. When the decision process has more than two possible outcomes, an extra layer of complexity may result from separate decision processes for each of the possible outputs. For example, in the graduated response system for nuclear reactor safety (blinking light, siren, automatic shutdown), the more drastic decisions may be associated with larger subsets.

There is also some good news. First, any coterie can be replaced by a multilevel weighted voting system [57]. Providing the details is beyond the scope of this chapter, but intuitively, some decisions are made early and others are postponed to later rounds of voting where different vote values are used with candidate sets that have been identified in earlier rounds. Second, even more general decision schemes than coteries can often be converted to weighted voting, as illustrated by the example in Fig. 4.5. Third, any quorum set (set of subsets of inputs to the voter) can be realized as multidimensional voting [1], a generalization of weighted voting.

4.3 VOTING SCHEMES AND PROBLEMS

Voting schemes can be classified in many different ways. In this section, we present a classification of voting schemes in the framework of our generalized weighted voting paradigm and discuss the salient features of, as well as potential problems in applying, some of the more important categories.

4.3.1 A Taxonomy of Voting

The four main components of a voting algorithm, namely input data (the x_is), output data (y), input votes (the v_is), and output vote (w), can be used to impose a binary

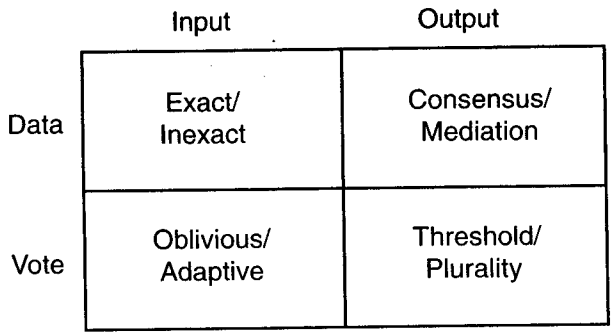

Fig. 4.6 Variations in voting based on input/output data and input/output votes.

4-cube classification scheme, leading to 16 classes [44]. Briefly, the four dichotomies used in the classification are defined as follows (see Fig. 4.6). The exact/inexact dichotomy has to do with whether input objects are viewed as having inflexible values or as representing flexible "neighborhoods." For example, bit-voting algorithms are exact, whereas word-voting on floating-point data may be inexact. Consensus voting involves agreement or quorum, in the sense of a subset of inputs "agreeing with" or "supporting" y, as we have chosen in our definition of weighted voting. With mediation voting, the output y is chosen to minimize or maximize an objective function of all inputs. This is excluded from our definition of weighted voting because in this case "support" has levels, perhaps related to the notion of distance between input and chosen output values. For example, a least-squares fit may be used to derive y. The oblivious/adaptive dichotomy corresponds to the v_is being set at design time or allowed to change dynamically (be *adjustable* or *variable*). Finally, threshold voting requires that w exceed a given threshold, whereas plurality voting identifies an output y with greatest support from the inputs.

Within each of the four boxes in Fig. 4.6, the first option is simpler (in terms of time complexity and implementation cost) than the second one and is taken to be the default option. The 16 classes of voting schemes obtained from the four dichotomies may be labeled by four-letter acronyms, beginning with the simplest scheme ECOT and ending with the most complex IMAP. When we talk about *threshold voting*, we are really considering the 8 schemes XXXT out of 16. Similarly, *adaptive voting* refers to any of the eight schemes XXAX and *inexact adaptive voting* encompasses the four schemes IXAX. When default settings are used for the unspecified features, we characterize the voting algorithm as simple: e.g., simple voting (ECOT), simple plurality voting (ECOP), and simple inexact adaptive voting (ICAT).

In the remaining subsections of Section 4.3, we consider some of the more common voting schemes and point to all possible variations of the basic variants of each scheme in relation to the categorization of Fig. 4.6.

4.3.2 Threshold Voting

Threshold voting is the simplest and oldest voting method. In consensus versions of this scheme, an a priori threshold value is specified (e.g., three inputs, or inputs with

votes totaling 10), and any output that has this level of support among the inputs is considered a valid output. Thus, if candidate outputs y_1 and y_2 are both supported by three inputs in a 3-out-of-5 majority voting scheme, then either output can be produced by the voting scheme. In this example, if the notion of support is defined to require exact equality, then *standard majority voting* with at most one valid output will result. Note that threshold voting is quite general in that it allows schemes such as 2-out-of-n (any output that is proposed and "seconded") or even 1-out-of-n (when inputs represent threats that are too serious to ignore, even if only one computation channel or a single sensor signals the threat). At the other extreme, *unanimity voting* is also a special case of threshold voting. In all these cases, the output vote w is implicit and built into the algorithm.

Threshold voting can be implemented without actual vote tallying, which is a requirement for plurality voting (see Section 4.3.3). Threshold voting can thus be significantly more efficient in cost and time complexity. For example, when the output must equal one of the inputs and "support" is transitive (x_i supports x_j and x_j supports x_k implies that x_i supports x_k), t-out-of-V weighted threshold voting requires $O(np)$ time and working storage space for $O(p)$ input objects, where $p = \lfloor V/t \rfloor$ and the unit of time is the latency to establish whether x_i supports x_j [40]; contrast this with the $\Omega(n^2)$ time complexity and $\Omega(n)$ space complexity of weighted or unweighted plurality voting when the input objects cannot be ordered and the $\Omega(n \log n)$ time complexity if they can. Easily derived corollaries of this result establish the relative simplicity of unweighted w-out-of-n and majority voting, the latter requiring $O(n)$ time and working storage space for a single input object. The algorithm just mentioned is readily parallelized for even greater performance [45].

Threshold voting may be used in a two-stage process. First, mutual support among the inputs is used to establish compatibility classes in which every member supports every other member. A compatibility class with total vote exceeding the threshold t is then chosen for the second stage where one of the members of the class (consensus voting) or a compromise output (mediation voting) is chosen as the output. Examples of the latter include calculating the mean within the chosen class or generalized median selection rule, as in Fig. 4.3.

4.3.3 Plurality Voting

In plurality voting, some form of vote tallying is required to determine which output has the greatest support among the inputs. If the output must equal one of the inputs, then vote tallying amounts to pairwise comparison of all inputs in order to establish which input supports which other inputs. This is required even if "support" is transitive because, in the worst case, only one pair of inputs support each other and only that particular comparison can establish this fact; all other comparisons indicate lack of support, thus not contributing toward deducing mutual support between x_i and x_j [40]. If input objects are ordered, however, sorting can be used before vote tallying to reduce the time complexity from $\Omega(n^2)$ to $\Omega(n \log n)$ in software. Fairly efficient sorting-based voting networks can be similarly designed in this latter case.

The case of arbitrary output (not necessarily equal to one of the inputs) is much more complicated, especially when the output space is large (the usual case for this type of voting). In such a case, it is clearly impractical to tally the votes for all possible outputs to select an output with maximal support. Practically, however, this is not a problem because "support" is not arbitrary in the sense, for example, of an input value 3.10 supporting 3.12 but not 3.11 or 3.09. In other words, support or lack thereof is a simple function of a suitably defined distance metric rather than an arbitrary relation. For example, if a numerical input x_i is taken to support all real values in the interval $[x_i - \varepsilon, x_i + \varepsilon]$, plurality voting is converted to interval voting (see Section 4.3.4).

Like threshold voting, plurality voting may be used in a two-stage process. First, mutual support among the inputs is used to establish compatibility classes. A largest, or highest weighted, compatibility class is then chosen for the second stage where one of the members of the class (consensus voting) or a compromise output (mediation voting) is chosen as the output.

4.3.4 Approval Voting

In the sociopolitical context, approval voting is a scheme in which each participant votes for a subset of candidates who meet his or her criteria for the position rather than just picking a "best" candidate. Thus, approval voting is not at odds with threshold or plurality voting but rather complements them; it is simply a specific mechanism for one input to support multiple outputs. Among advantages of approval voting, which may carry over to its use in dependable computing, is the property that the splitting of votes among several qualified candidates does not cause a less qualified candidate to win a plurality of the votes (run-off elections are intended to prevent this from happening, but they do not solve the problem in all cases). Multiple approved outcomes may be due to nonunique answer to a problem or uncertainties in the solution process. As an example of where approval voting might be useful, consider a process control system where safe settings for a particular system variable are proposed by multiple redundant versions of the control program and the final setting is derived from among those that enjoy greatest approval.

Approval voting forces a voter to divide all possible candidates into two classes of acceptable (supported) and unacceptable (not supported). This is inherently limited as there may be different shades of approval (some candidates may be deemed more qualified than others). More generally, we can allow voters to provide a list of candidates with their associated support levels. Interval voting, as a special case of approval voting, where the approved outputs by each voter constitute an interval of values along the real line, faces similar drawbacks in that all values in the interval offered as an input to the voting process are deemed equally valid outputs from a voter's viewpoint. It is quite natural to assume that values near the midpoint of an interval are somehow preferable to those closer to the boundaries, but this is not taken into account in simple interval voting (more on this later). Without this latter assumption, the output value must be chosen from a subinterval that is the intersection of the largest number of the input intervals.

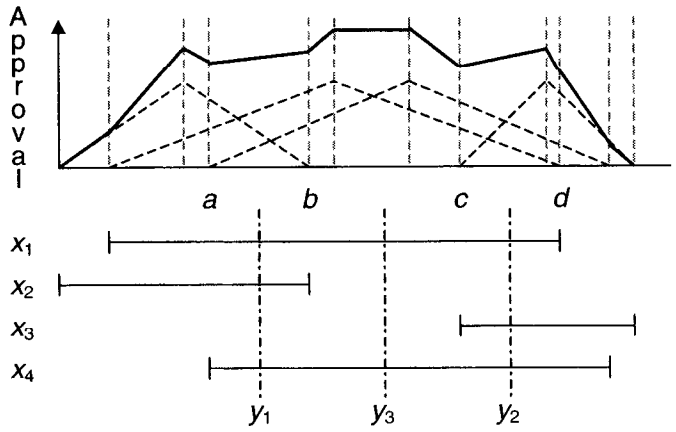

Fig. 4.7 Example of interval voting when there are multiple, disjoint, maximally approved subintervals.

Although the process described in the preceding paragraph is usually straightforward, there are some complications that must be dealt with. Consider, for example, nonweighted voting with the four intervals shown in Fig. 4.7. Two disjoint subintervals $[a, b]$ and $[c, d]$ enjoy a support level of 3 among the voters. So, it appears that output values y_1 and y_2 have equal support and either value could be chosen as output. However, what would be the basis of choosing one value over the other? Why would x_2 be given preference over x_3 or vice versa? In fact, should we not discount the votes of x_2 and x_3, given that they essentially disagree with each other, choosing the output from the intersection of x_1 and x_4? But then, would not an output y_3, say, that is outside both x_2 and x_3 be somehow of lower quality than y_1 or y_2?

Answers to the questions posed in the preceding paragraph depend on the context and application. For example, it may make sense to attach to the points in each interval varying approval levels, ranging from 0 at the boundaries to 1 at the midpoint (see the dotted triangular shapes in Fig. 4.7). Adding up the approval levels, as done at the top of Fig. 4.7, leads to total approval levels and one or more best choices. Other combining rules are also possible. However, in the interest of fairness, the triangular support areas for each interval may be normalized to have unit area, thus preventing a voter from exerting undue influence by presenting a very wide interval as its input. It may also be desirable to associate the outside of a proposed interval with some form of negative approval, hence penalizing values that are too far outside an interval.

4.4 VOTING FOR DATA FUSION

Data fusion refers to integrating data for the purpose of drawing correct conclusions from imprecise, incomplete, or incompatible raw data [22, 33, 51]. The data are often provided by sensors, with or without some kind of preprocessing (raw data from simple sensors, or processed data from intelligent sensors). For an overview of the field of data fusion, see Ref. [18]; it not only contains a compendium of

the most important ideas and techniques for data fusion but also provides an extensive list of Web sites, news groups, and other Internet resources.

4.4.1 Sensor Processing and Fusion

Sensor processing is needed in environmental monitoring, intelligent manufacturing, process control, military surveillance, medical imaging, robotics (e.g., handling of hazardous material), and remote sensing [21]. Sensors can range from rudimentary (metal or smoke detector, barometer) to highly complex (identifying obstacles in a robot's path). The goal of data fusion is to overcome one or more of the following:

1. inherent sensor limitations,
2. permanent or intermittent malfunctions,
3. communication-related errors or delays, and
4. errors in storage and processing.

A good example of *aspect 1*, overcoming sensor limitations, is when one uses two sensors to obtain target position data: one with poor elevation and azimuth precision but with relatively accurate range (radar) and another with complementary characteristics (forward-looking infrared or FLIR). Figure 4.8 depicts the greater precision of fused data. Another example is when the coverage of a sensor (e.g., area sensed) is limited and multiple sensors are needed to provide the desired coverage. Limitations are assessed with respect to the required certainty level. A particular sensor may be adequate for one application (uncritical situation or unsophisticated adversary) but deemed to be limited for another (battlefield, advanced jamming methods). As for *aspect 2*, sensor unavailability may result from physical faults, jamming, or overload in the case of non-dedicated sensors. Both *aspects 1* and *2* are usually handled through sensor replication, whereas *aspects 3* and *4* are dealt with via coding and redundant computations.

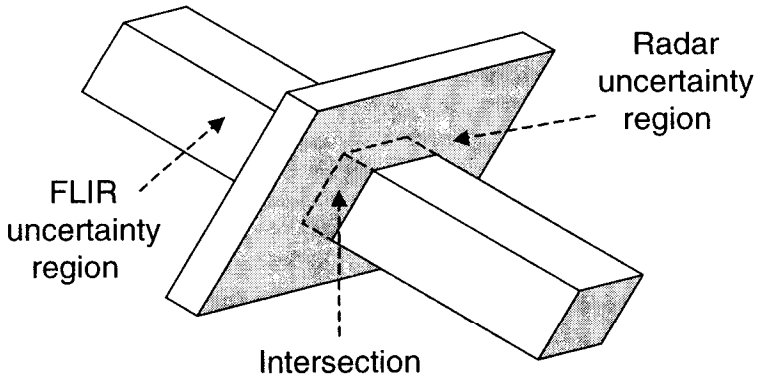

Fig. 4.8 Pinpointing the location of a moving object via two different sensors with complementary characteristics.

4.4.2 Components for Data Fusion

The sensors used in a multisensor data fusion system may be similar/competitive (e.g., the two eyes in a human) or diverse/complementary (e.g., visual, tactile, and auditory sensors). If the sensor data have no common feature, fusion is difficult, if not impossible (see the discussion of data diversity in Section 4.4.3). At the other extreme, if all features of the multiple sensors are common, then fusion reduces to filtering out the variations in measurement [52]. In general, multisensor data fusion may be done in a hierarchical manner, with fused data from one level forming the raw data for the next level.

As in other technical disciplines, sensor fusion systems and various states of processed data in them are described using special terminology [58]. At one system boundary, *sensors* collect *raw data*, which are then subjected to *preprocessing* to yield *filtered data* for each sensor. Sensors can be classified based on their decision strategies into *hard-* and *soft-decision* sensors. The former process their incident signal data and use decision rules to declare outcomes such as target identity, whereas the latter may provide partial identity evidence upon signal detection. Soft-decision sensors "accumulate and integrate evidence, reporting partial evidence and associated uncertainty (via probabilities, fuzzy membership functions, confidence factors, or evidential intervals)" [17]. Data provided by a sensor take varying forms (waveform, integer designating a class, real vector, image) and usually include information about the sensor itself: current state (e.g., pointing angle), configuration, health, and so on. The optional *preprocessing* or *filtering* may be needed to reduce the data volume to avoid overwhelming the processing part.

The data then enters the processing part, which is composed of *alignment* and *correlation* (low-level processing), yielding *calibrated data*, and *assessment* and *detection* (high-level processing), which results in the final *integrated data.* The low-level processing stage consolidates the data by using spatial and temporal references, unit conversions, pairwise association of observations (a quadratic-time process), and position/identity determination. The high-level processing stage interprets the output of the previous stage and makes inferences in the context of system structure and goals (e.g., threat assessment in a battlefield). At the other end of the fusion system, *decision* or *actuation* is based on the latter integrated data and may involve external guidance via a *human interface.* This final system output may be further processed external to the fusion system, used to directly control some system components (e.g., adjust or reorient sensors), or tied to other systems (e.g., warning or response units).

The sensor data fusion community, which has grown tremendously since the emergence of early rudimentary systems in the late 1970s, has long sought a unified approach [17]. This group is fragmented, with members interested in practical military applications isolated from the more theoretically inclined nonmilitary segment. Research in data fusion is multidisciplinary and uses techniques from signal processing, statistics, pattern recognition, and information theory, among others. A variety of results and techniques from decision or detection theory (e.g., Bayes' method), estimation theory (least-squares, maximum-likelihood), association or correlation, and uncertainty management (evidence/belief theory, Shafer-Dempster reasoning, fuzzy calculus) provide the theoretical bases for system implementations [61].

4.4.3 Data Fusion Examples

In this subsection, we take two example applications from the multisensor data fusion literature and formulate them in terms of generalized voting. Our goal is to show that some techniques from one area can be useful in the other. In addition, we discuss an example application context where both techniques are applied in a complementary manner.

The first example is from p. 93 of Ref. [61]. Two sensors produce ambiguity sets following their attempt to recognize the class of a target. Sensor 1 supplies the ambiguity set {4, 5, 12, 18}, whereas the second sensor provides {12, 21, 32, 33}. Combined, the two sensors unambiguously identify the target as being in class 12. This type of fusion is actually a special case of approval voting. Each of the channels provides as output a set of approved values (e.g., system states that are deemed safe following a detected fault). The approved values, with each item or each set perhaps having an associated weight or confidence level, are combined through an approval voting algorithm to identify the best value or set of values.

The second example is somewhat more complex and has been the subject of extensive research in multisensor data fusion [9, 20, 21, 34]. Assume that multiple sensors provide real-valued scalar data. A nonfaulty sensor S_i provides the real-valued output x_i. With knowledge of sensor accuracy, one can define an interval $[x_i - \Delta x_i, x_i + \Delta x_i]$ as containing the *correct* or intended sensor data. The objective is then to obtain a value or an interval of values that represents the best estimate of the sensed quantity. Again, the multiple intervals can be viewed as sets of values approved by the sensors, with simple or weighted voting used to fuse the data. Faulty sensors are properly handled as their intervals likely do not overlap with those of correct sensors (correct conclusion is reached with very high probability even if they do).

It is interesting to note that many of the results published in the references above can be obtained directly and simply from the approval voting interpretation. Conversely, some of the bounds derived in these references on the width of the fused interval can be applied to analyze the precision and fault diagnosability in other interval voting applications.

Consider now either of the problems above in a situation where the processing part of the sensor fusion system is also subject to unavailability or failure. This motivates distributed multichannel processing of sensor data, with the outcome being multiple fusion results for use or interpretation by humans or another reliable multichannel system. In this context, voting for fault tolerance and data fusion become indistinguishable, as it is difficult to differentiate between sensor failures or inaccuracies and computation errors.

4.4.4 Dealing with Data Diversity

In dependable computing, data diversity refers to slightly modified data that allow recomputation of desired results in such a way that it is unlikely for faults to affect the recomputed results in the same way as those obtained from the original

data, hence leading to fault tolerance or, at least, fault detection [2]. This is akin to the situation depicted in Fig. 4.8, where multiple sensor data are of the same nature, differing only in format and/or precision. Here, we take a more general view so as to cover many types of data fusion. For example, high temperature and high pressure may constitute diverse forms of data that can be used to deduce hazardous conditions in a nuclear reactor or chemical plant. In the latter case, the data are not slightly modified but rather take vastly different forms: 200°C and 1,000 kg/cm^2 are unrelated in form or meaning, yet both may support the conclusion that a certain hazardous condition exists.

Consider a voter with an output space of size 3: viz, no action (0), triggering an audible alarm (1), or emergency shutdown (2). Inputs to this voter need not all be of the same type; temperature, pressure, and other sensors can supply independent evidence of the system state for use in drawing an appropriate conclusion. What is important is to be able to translate such evidence to a level of support for each possible output. Once this has been done, the rest of the voting process can proceed without regard to the type of input that was used. Figure 4.9 is a graphical depiction of this idea with four inputs. Each input x_i supports an output y_j at a certain level, represented by the thickness of the line connecting them in Fig. 4.9. The total support for each output can then be derived in exactly the same way as for nondiverse data inputs.

Although the idea discussed above is intuitively appealing, its implementation requires extensive evaluation and great care. The key to success in this approach is a robust way of deducing support levels. This, along with judicious weight assignment, provides a unified way of treating confidence levels derived from the data and supplied externally.

4.5 IMPLEMENTATION ISSUES

Like all algorithm design problems, correctness and performance of voting schemes are key properties to evaluate and ascertain for practical applications. In Section 4.5,

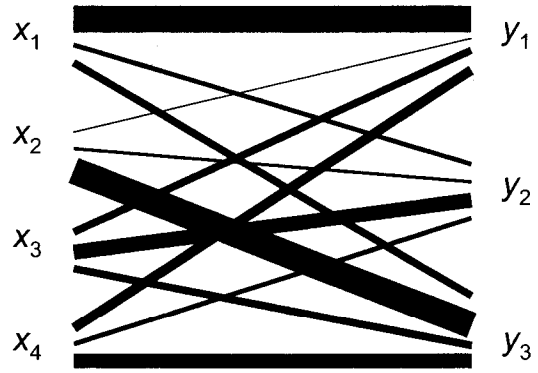

Fig. 4.9 Different levels of support for various outputs from diverse inputs (heavier lines indicate greater levels of support).

following a discussion of some inherent limitations of voting systems, we deal with a number of relevant issues in this regard, citing examples where appropriate.

4.5.1 Impossibility Results for Voting

The process of sociopolitical voting is often taken for granted, and its theoretical underpinning is not well appreciated. For example, it is not generally known that, once the number of candidates exceeds two, no "best" voting scheme exists. That is, worst-case scenarios can be constructed for any given voting scheme so as to yield an illogical or undesirable result. For an exposition of these problems, framed in the context of recent US and French presidential elections, see Ref. [12]. Another highly readable account of certain problems in voting can be found in Ref. [59].

An ideal voting scheme should reflect the will of the electorate. Suppose that there are three candidates (c_1, c_2, c_3) and that each voter freely and independently ranks them according to his or her preference (condition 1, no big brother). Further suppose that the relative ranking of a pair of candidates should be determined strictly by the voters' preferences regarding the same pair, without taking information about other candidates into account (condition 2, independence of irrelevant alternatives). A valid voting procedure should result in each of the outcomes "c_i preferred over c_j" and "c_j preferred over c_i" for some voting profiles (condition 3, involvement). Finally, it is reasonable to require that the final outcome not always agree with the preferences of a single voter or always be the opposite of someone's preferences (condition 4, no dictatorship or antidictatorship). Arrow's theorem, in somewhat generalized form, states that no voting procedure exists that satisfies conditions 1–4 above [10].

One implication of Arrow's impossibility or incompleteness result to voting for dependable computation and data fusion is that more sophisticated voting schemes are needed to ensure a smaller likelihood of illogical results. Consider for example a particular voter's preferences among pairs of candidates: $c_1 > c_2$, $c_2 > c_3$, $c_3 > c_1$, where $c_i > c_j$ means that the voter chooses c_i over c_j. This kind of voter, with circular preferences, is referred to as a "confused voter," because there is general expectation that preferences be transitive (a person who prefers a conservative to a moderate and a moderate to a liberal should not pick a liberal over a conservative). Even if confused voting is deemed illogical in the context of dependable systems (and this has not been proven yet), it is still the case that one of the more faulty hardware or software units may cast a confused vote and the voting process should be robust enough to derive a logical conclusion despite the presence of confused votes.

Assuming transitivity of preferences for now, a voting scheme known as *true majority* requires each voter to submit a rank-ordered list of the candidates, say in decreasing order of preference. The voting process then picks a candidate who beats each opponent in pairwise competitions based on the submitted rankings. To see that this outcome may not be the same as the outcome of ordinary majority voting, consider an election with 3 candidates, depicted geometrically [53] in Fig. 4.10. The integer given in each triangular region denotes the number of

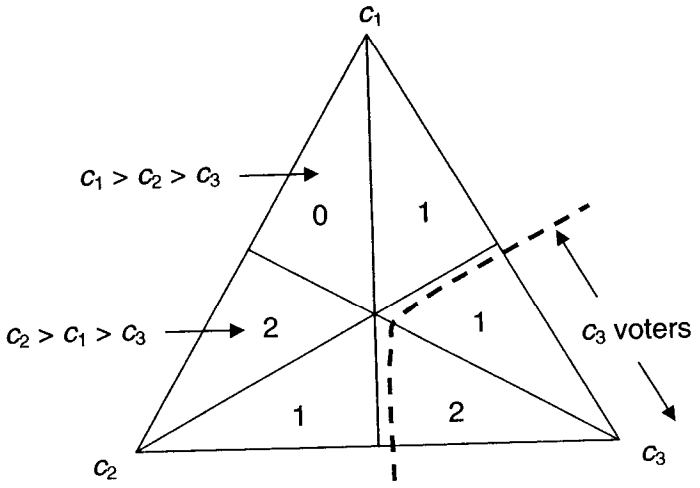

Fig. 4.10 Preferences of 7 voters in a 3-way election. A line bisecting the triangle, with c_i and c_j on its two sides, divides the voters according to their preference for c_i relative to c_j.

voters who rank the three candidates in the same way. For example, the leftmost number 2 indicates that two voters prefer c_1 to c_3, c_2 to c_1, and c_2 to c_3 (abbreviated $c_2 > c_1 > c_3$). True majority voting results in c_3 being elected, given its slim 4-to-3 edge over both c_1 and c_2. In ordinary voting when only the first-choice candidate is identified, the votes would be 1-3-3 for c_1, c_2, and c_3, respectively (no majority). Even the well-regarded Borda voting scheme, in which a first-choice candidate in an n-way election gets $n - 1$ points, a second-choice candidate $n - 2$ points (2 points and 1 point in our 3-way voting example), and so on, with the points tallied to determine the winner, leads to the unacceptable ordering $c_2 > c_3 > c_1$. The moral of our story is that no voting scheme works as expected in all cases. The best that we can do is to come up with a scheme that works well in the desired context and with the semantics attached to votes in a particular application.

To see the relevance of the example in Fig. 4.10 to voting in dependable computing or data fusion, consider the three candidates to be three different conclusions based on input data (e.g., that a radar object is a civilian airliner, a bomber, or a fighter jet) and the various orderings represent the conclusions of seven program versions or smart sensors about the target.

4.5.2 Correctness Concerns

There are two types of correctness concerns for voting schemes. The first one, which we will basically ignore here, is the issue of correctly implementing a chosen voting scheme in hardware or software. As voting schemes become more complicated to deal with more complex input objects or to achieve greater performance, it is entirely possible to have residual design errors in their implementations. We ignore this class of correctness concerns here because they are not unique to voting schemes. Application of sound hardware/software design methodologies is as important here as it is in the design of any complex system.

More relevant to our discussion is the notion of correct deduction of an output from the input information supplied to the voter. In other words, the output should be chosen so that it is the most likely correct output based on the input information and other auxiliary data. In the literature, this has been referred to as *maximum-likelihood voting*. Of course, there are situations where we do not choose the maximum-likelihood output due to safety concerns. For example, if out of many sensors measuring a critical parameter of a nuclear reactor only a couple indicate a dangerous condition, we may decide to override the output that is most likely to be correct [8]. Concerns for safety can be separately incorporated into the voting process, so let us focus on the maximum-likelihood voting strategy.

One may think of the inputs to a maximum-likelihood voter as comprising a *syndrome*, which includes the inputs originally presented by multiple data sources as well as other information such as outcomes of acceptance tests, results of pairwise comparisons, and the like [13]. Additionally, the voter may have information, in the form of probability distributions, about the likelihood of correct results or of various incorrect results from each source and the a priori distribution of valid results in the output space. The computation for a maximum-likelihood voter is generally quite complex as it entails determining the probabilities of correctness for each possible output. When the output space is small [36], this might be feasible, given the high computational power of modern microprocessors or the capabilities of hardware devices (custom ICs or FPGAs). For large output spaces, a brute-force approach is impractical, and theoretical results are needed to reduce the search space.

One such result pertains to the case when the inputs x_i to the voting process are integer valued and are thus considered exact. In other words, all correct data sources will present precisely the same value to the voter. In this case, under some fairly reasonable conditions, the search for an optimal (maximum-likelihood) result can be limited to at most $n + 1$ values: the n values from the data sources and one representative value from among those not offered by any source [8]. Consider for example an output space with five possible values and seven data sources producing two of these values, as depicted in Fig. 4.11. In many cases, y_1 will be chosen as the maximum-likelihood output. This is the case, for example, when data sources have similar error characteristics and the a priori output distribution is uniform. On the other hand, if output y_2 has a much higher a priori probability and/or data sources are more prone to producing an erroneous y_1 than an erroneous y_2, then y_2 might

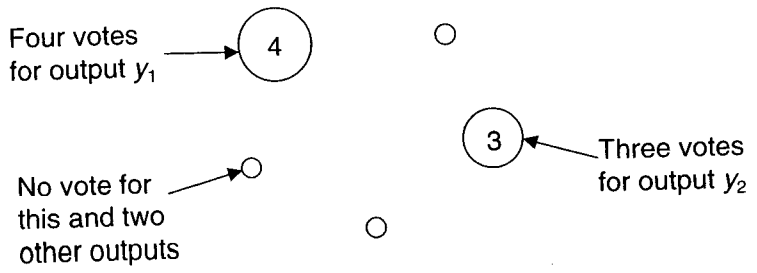

Fig. 4.11 Example output space in maximum-likelihood voting.

become the maximum-likelihood result. In this example, only one of the three possible outputs with no vote (the one with maximum a priori probability of being correct) need be considered, if at all, in the selection process.

Maximum-likelihood output selection is somewhat more complicated when correct values differ slightly from each other (inexact voting). The process to be used in this case depends on how far a "correct" result can stray from the ideal value. Furthermore, straying from the ideal value can be treated as uniform (much like intervals denoting a set of equally likely values in interval voting) or can be associated with a probability distribution that makes larger deviations less likely. Even in the simple uniform case, if a correct value can differ from the ideal value by l ticks, where one tick is the resolution of number representation, some $(2l+1)n+1$ different values may have to be considered to deduce the maximum-likelihood output. This is because each of the n inputs has l potential values for the output on each side and because one value not produced by any of the voters must also be considered [8].

4.5.3 Performance Considerations

Performance parameters of interest in assessing voting schemes include latency and throughput. Even though low latency generally implies high throughput, the converse is not true, especially for hardware implementation of voting, which can be easily pipelined (see Fig. 4.1). Latency is defined as the length of the time interval between the availability of the last input and the production of the voter output. As such, any preprocessing that can be performed on early inputs to speed up the process after the receipt of the final input is not included in the latency, provided that such is performed concurrently and does not slow down the process of input generation or reception. Throughput is defined as the number of adjudications per unit time and is dictated primarily by application requirements and the frequency of voting.

Besides the latency of the voting operation itself, voting introduces additional delays in the form of data communication and synchronization. The latency of the voter is measured from the time it receives its last input. However, if this input must be communicated to the voter from a distant node within a distributed system, the data communication latency must somehow be figured in. Additionally, if multiple data sources must await the voting outcome before they are allowed to continue, the corresponding barrier synchronization latency must also be included. Thus, the voting latency must be viewed as having three components: latency of the voting algorithm, added delays for data transmission to and from the voter, and synchronization overhead.

Voting latency depends on the algorithm utilized and on the particular inputs presented to the algorithm. For most voting algorithms of practical interest, the voting latency is not significant. Even when voting requires interactive convergence, as in Refs. [4] and [24], it is not the computational part of the algorithm but rather the multiple rounds of communication that is the dominant factor. Despite the observation above, it is still the case that some steps can be taken to minimize the computational complexity. Examples include decomposition of voting schemes

into a hierarchy of simpler schemes [31] and choice of integers and rational numbers for the input votes and the voting threshold, respectively [37].

The communication overhead of voting is a significant factor when software voting is used in a distributed environment. Here, some reduction in load is possible by optimizing the voting scheme for the most probable cases, thus paying a heavy communication penalty only if necessary in rare cases. For example, in n-way exact voting by n sites, each site can broadcast $(1/n)$th of the result bits to all sites that assemble the pieces, compare the assembled word with their own results, and broadcast agreement bits to all other sites. If majority agreement exists (the usual case), then the broadcast pieces represent the correct majority, assuming perfect communication. Only when majority agreement does not exist will each site have to broadcast its complete result in a second round. When the error probability is not very small, encoding the result in an error-correcting code and sending $(1/n)$th of the codeword reduces the chance of complete transmission, possibly leading to higher communication efficiency [62].

Hardware voters generally work in synchrony, expecting the inputs at exactly the same time, to avoid excessive circuit complexity due to the need for buffering and matching of inputs. For software voters, however, the simplicity advantage of synchronous operation is often overshadowed by its severe performance penalty. It is quite advantageous to allow speculative computation to proceed while a previous set of results is being voted on. The results of such speculative executions are committed to permanent memory only upon the completion of voting and dissemination of its relevant outcomes. In some cases, it might be possible to strike a balance between the overhead of tight synchronization and the algorithmic complexity of fully asynchronous operation via an intermediate approach [54].

4.6 UNIFYING CONCEPTS

In Section 4.6, we point to similarities between the concepts applied to data fusion and those used in dependable computation, propose a unified terminology, and show how both fields can benefit from more interaction with each other [43]. Formulation of these approaches to data quality enhancement as generalized voting schemes constitutes a step in this direction.

4.6.1 Toward a Common Terminology

Traditionally, researchers in multisensor data fusion have assumed that the communication, storage, and processing subsystems are highly reliable and have focused only on algorithms for integrating data from homogeneous or heterogeneous collections of potentially faulty/inaccurate sensors. On the other hand, researchers dealing with redundant or replicated computations have, for the most part, assumed that the input data is *perfect* and that the only sources of errors or inaccuracies are faults in the data communication, storage, and processing subsystems and perhaps the numerical characteristic of the algorithms being used (e.g., with regard to the

accumulation of round-off errors). Despite this separation, the two fields have many problems and techniques in common. The main goal in each is to proceed from raw, suspect, or low-quality data to integrated, trustworthy, or high-quality data on which important decisions can be based.

Figure 4.12 highlights the similarities between multichannel computation and sensor fusion, proposing a common terminology applicable to both fields. Some of the correspondences between sensors and computation channels (raw and suspect data, preprocessing and acceptance testing, filtering and reasonableness checks, integrated and trustworthy data, decision and output) are self-explanatory. The human interface, which is absent from the dependable computing track, has been added to account for the fact that advanced voting techniques often involve adjustable parameters, perhaps necessitating human intervention.

From the viewpoint of unifying the two areas, the most important system components in Fig. 4.12 are the boxes labeled "Correlator" and "Voter" in the unified terminology. The concept of correlation comes from the data fusion side, but it has been used in limited forms by the dependable computing community when results of computation channels are cross-compared as a way of weeding out obviously wrong data or for producing a syndrome to help with the subsequent decision process. The term "adjudication" was introduced by fault-tolerant computing researchers to avoid the restrictive meaning often associated with "voting." Others prefer to use "generalized voting." We simply use "voter" because our view of voting, as introduced in Section 4.2.3, is indeed much more general than commonly considered. Because the correlator box assists the voter in deriving its conclusions, it is quite possible to merge the two into a still more general voting unit, as depicted in Fig. 4.12 in the form of a dotted circle. This reduces the number of different blocks to three: data modules, evaluators or testers, and the voter.

Researchers in both multisensor data fusion and reliable multichannel computation are becoming increasingly aware of the need for unifying theories.

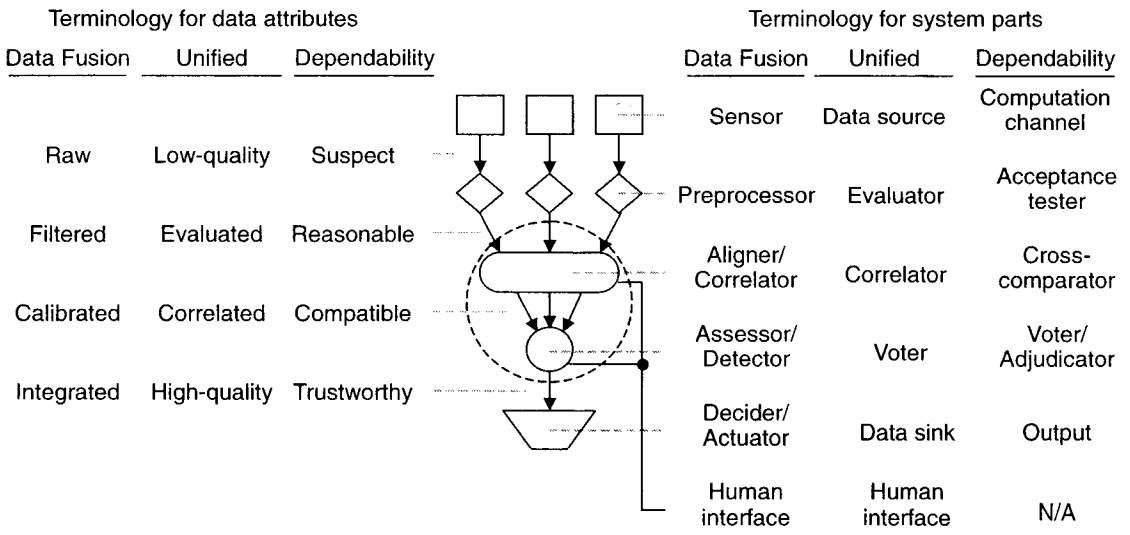

Fig. 4.12 Unified view of voting for data fusion and dependable computation.

For example, it has been observed that "while the term data fusion is widely used, its meaning is subject to varying interpretations," but "data fusion has a common basis in theory, which is independent of application" [61]. Similarly, the need for a generalized formulation of voting to cover the wide variety of methods in current use was noted long ago in the reliable computing community [32].

4.6.2 A Data-Centered Methodology

In Section 4.6.2, we briefly discuss a data-centered or data-driven methodology that can be used as a framework for dealing with both multisensor data fusion and reliable multichannel computation in forms that are more general than Fig. 4.12. The methodology, which uses both voting and acceptance testing (evaluation), fosters a unified treatment of data errors, inaccuracies, and tardiness regardless of causes (data generation, collection, transmission, storage, manipulation, interpretation) and explicates the optimal allocation of resources for dealing with various error sources. It also allows the designer to view varied redundancy features or techniques, from data and design diversity to retry and replication, as instances of a general data quality enhancement process, thus facilitating comparisons and design tradeoffs. The desirability of associating a confidence level with each data item has been noted in both the dependable computing and data fusion communities [39, 49]. Although obtaining or assigning confidence levels is nontrivial, this difficulty should not discourage us from seeking appropriate methodologies.

The aforementioned data-driven methodology, originally proposed for redundancy evaluation and optimization in software-based multichannel computations [47], focuses on data correctness and accuracy rather than on the reliable operation of modules producing or handling the data. Note that the symmetric way in which data sources are depicted in Fig. 4.12 is neither necessary nor even desirable in all cases, neither is the restriction that a single voter be used on the path from data inputs to output. In fact, varying characteristics of data sources and the amenability of some types of results to fast and efficient acceptance testing, makes a hierarchical or clustered approach much more efficient. In ongoing research, we are using DD-MTV (data-driven module/test/voter) graphs as tools for describing and modeling various software fault tolerance architectures. Figure 4.13 depicts an architecture with two voting levels as an alternative to straight n-version programming whereby n independently developed software modules provide results to an n-way majority voter (one version has been removed and replaced by the test T). Combining n-version programming with acceptance testing has been proposed by several researchers, but the study of asymmetric configurations such as the one shown in Fig. 4.13 is fairly recent [47].

Because inputs to the second voter V_2 in Fig. 4.13 have different reliabilities even when all data sources are identical, weighted voting must be used to optimize the overall reliability. For the same reason, the weight attached to the leftmost input of V_2, coming from the test module T, must depend on the test outcome and on the level of disagreement present among the inputs to V_1. In the simplest case, outputs of data sources may begin with dependability-tags (d-tags) of 1. Voter V_1

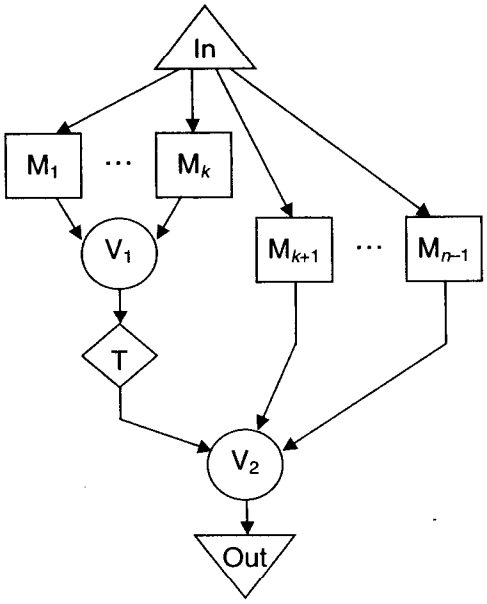

Fig. 4.13 DD-MTV graph showing an asymmetric treatment of $n - 1$ data sources.

attaches a d-tag equal to the count of supporting inputs to its output which is then forwarded to T. Finally, T adjusts the d-tag depending on the test's outcome. Such adjustments are done via d-raising and d-lowering functions that can be variously defined.

Variations of the simple scheme described above can be used to optimize the overall system reliability. Examples include different d-tag assignment schemes for weighted voters and alternate vote augmentation or reduction policies associated with pass or fail (or graduated) test outcomes. However, even the simple strategy reflected in the DD-MTV graph of the example above has produced interesting results regarding general combinations of data sources and evaluative testing that are quite counterintuitive. This has led to a better understanding of such architectures and facilitated the search for optimal configurations.

4.7 CONCLUSION

We have reviewed multisensor data fusion and reliable multichannel computation, pointing out their similarities as well as the benefits of a unified approach to their treatment. The unified approach involves a common terminology for referring to system components and data states or attributes, along with a methodology that allows data of varying qualities and formats to be integrated. Through several examples from data fusion and dependable computing applications, we have shown that (generalized) weighted voting, with an appropriate formulation of "support" and suitable assignment and dynamic manipulation of weights, can serve as a powerful tool in both domains. Threshold, plurality, and approval voting each has a place in a system designer's toolbox. The choice of a voting method is dictated by

(probabilistic) correctness attributes, as well as cost-performance tradeoffs. We have also shown that there is more to voting than the simple majority/plurality viewpoint heretofore dominating the literature in dependable computing and that voting methods are inherently limited.

One of our important conclusions is that considering voting as data fusion, and drawing from both mathematical studies of the anomalies in voting as well as sensor fusion work in signal processing, can be beneficial to further development in the field of dependable computing. A key to continued progress in this direction is integration of research results from many different disciplines into the unified model presented here. Aspects of the integration include drawing from well-established and emerging formalisms such as metrology [30], rough set theory [48], and fuzzy control and arithmetic [23, 63]. These, and other disciplines, that involve decision processes with uncertainties in data, must be carefully studied in search of insights and techniques that can be applied to data fusion and dependable computation. In parallel and distributed systems, voting decisions must be distributed so as to provide greater robustness and performance. With distributed computation, sensing, and voting, the possibility of malicious or Byzantine faults, in the form of malfunctioning sensors or node that send conflicting information to other nodes, must be given serious consideration [11].

REFERENCES

1. Ahamad, M., M.H. Ammar, and S.Y. Cheung. Multidimensional voting. *ACM Trans. Computer Systems*, vol. 9, no. 4, pp. 399–431, November 1991.
2. Ammann, P.E. and J.C. Knight. Data diversity: an approach to software fault tolerance. *IEEE Trans. Computers*, vol. 37, no. 4, pp. 418–425, April 1988.
3. Avizienis, A. The *N*-version approach to fault-tolerant software. *IEEE Trans. Software Engineering*, vol. SE-11, no. 12, pp. 1491–1501, December 1985.
4. Azadmanesh, M.H. and A.W. Krings. Egocentric voting algorithms. *IEEE Trans. Reliability*, vol. 46, no. 4, pp. 494–502, December 1997.
5. Ballou, D.P. and G. Kumar-Tayi. Methodology for allocating resources for data quality enhancement. *Communications of the ACM*, vol. 32, pp. 320–329, March 1989.
6. Barbara, D. and H. Garcia-Molina. The reliability of voting mechanisms. *IEEE Trans. Computers*, vol. 36, no. 10, pp. 1197–1208, October 1987.
7. Battiti, R. and A.M. Colla. Democracy in neural nets: voting schemes for classification. *Neural Networks*, vol. 7, no. 4, pp. 691–707, 1994.
8. Blough, D.M. and G.F. Sullivan. Voting using predispositions. *IEEE Trans. Reliability*, vol. 43, no. 4, pp. 604–616, December 1994.
9. Brooks, R.R. and S.S. Iyengar. Robust distributed computing and sensing algorithm. *IEEE Computer*, vol. 29, no. 6, pp. 53–60, June 1996.
10. Campbell, R.B. Rational election procedures. Chapter 3 in *Applications of Discrete Mathematics* (edited by J.G. Michaels and K.H. Rosen), McGraw-Hill, pp. 40–56, 1991.

11. Clouqueur, T., K.K. Saluja, and P. Ramanathan. Fault tolerance in collaborative sensor networks for target detection. *IEEE Trans. Computers*, vol. 53, no. 3, pp. 320–333, March 2004.
12. Dasgupta, P. and E. Maskin. The Fairest vote of all. *Scientific American*, vol. 290, no. 3, pp. 92–97, March 2004.
13. Di Giandomenico, F. and L. Strigini. Adjudicators for diverse-redundant components. *Proc. 9th Symp. Reliable Distributed Systems*, pp. 114–123, 1990.
14. Freund, Y. Boosting a weak learning algorithm by majority. *Information & Computation*, vol. 121, pp. 256–285, September 1995.
15. Garcia-Molina, H. and D. Barbara. How to assign votes in a distributed system, *J. ACM*, vol. 32, no. 4, pp. 841–860, October 1985.
16. Gifford, D.K. Weighted voting for replicated data. *Proc. 7th ACM SIGOPS Symp. Operating System Principles*, pp. 150–159, December 1979.
17. Hall, D.L. *Mathematical Techniques in Multi-Sensor Data Fusion*, Artech House, 1992.
18. Hall, D.L. and J. Llinas, *Handbook of Multisensor Data Fusion*, CRC Press, 2001.
19. Iyengar, S.S., R.L. Kashyap, and R.N. Madan (Guest Editors), Special Section on Distributed Sensor Networks. *IEEE Trans. Systems, Man, and Cybernetics*, vol. 21, no. 5, pp. 1027–1031, September/October 1991.
20. Iyengar, S.S., D.N. Jayasimha, and D. Nadig, A versatile architecture for the distributed sensor integration problem, *IEEE Trans. Computers*, vol. 43, pp. 175–185, February 1994.
21. Iyengar, S.S., L. Prasad, and H. Min. *Advances in Distributed Sensor Integration*, Prentice-Hall, 1995.
22. Jayasimha, D.N. Fault tolerance in multisensor networks. *IEEE Trans. Reliability*, vol. 45, no. 2, pp. 308–315, June 1996.
23. Kaufmann, A. and M.M. Gupta, *Introduction to Fuzzy Arithmetic: Theory and Applications*, Van Nostrand, 1991.
24. Kieckhafer, R.M. and M.H. Azadmanesh. Reaching approximate agreement with mixed-mode faults. *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 1, pp. 53–63, January 1994.
25. Klein, L.A. A Boolean algebra approach to multiple sensor voting fusion. *IEEE Trans. Aerospace and Electronic Systems*, vol. 29, no. 2, pp. 317–327, April 1993.
26. Koren, I., Y. Koren, and B.G. Oomman. Saturating counters: application and design alternatives. *Proc. 16th IEEE Symp. Computer Arithmetic*, pp. 228–235, June 2003.
27. Lee, J.Y., H.Y. Yoon, and A.D. Singh. Adaptive unanimous voting (UV) scheme for distributed self-diagnosis. *IEEE Trans. Computers*, vol. 44, no. 5, pp. 730–735, May 1995.
28. Levitt, K.N. et al. Beyond FTMP and SIFT—advanced fault-tolerant computers as successors to FTMP and SIFT. In *The Fault-Tolerant Multiprocessor Computer* (edited by T.B. Smith and J.H. Lala) Noyes Publications, pp. 733–782, 1986.
29. Libby, V. (Editor), *Data Structures and Target Classification*, SPIE Proc., vol. 1470, April 1991.
30. Lira, I. *Evaluating the Measurement Uncertainty: Fundamentals and Practical Guidance*, Institute of Physics Publishing, 2002.
31. Loeb, D.E. The fundamental theorem of voting schemes, *J. Combinatorial Theory*, series A, vol. 73, pp. 120–129, 1996.

32. Lorczak, P.R., A.K. Caglayan, and D.E. Eckhardt. A theoretical investigation of generalized voters for redundant systems. *Proc. Int'l Symp. Fault-Tolerant Computing*, pp. 444–451, 1989.

33. Luo, R.C. and M.G. Kay. Multisensor integration and fusion in intelligent systems. *IEEE Trans. Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 901–927, September 1989.

34. Marzullo, K. Tolerating failures of continuous-valued sensors. *ACM Trans. Computer Systems*, vol. 8, pp. 284–304, November 1990.

35. Mathur, F.P. On reliability modeling and analysis of ultra-reliable fault-tolerant digital systems. *IEEE Trans. Computers*, vol. 20, pp. 1376–1382, November 1971.

36. McAllister, D.F., C.-E. Sun, and M.A. Vouk. Reliability of voting in fault-tolerant software systems for small output spaces. *IEEE Trans. Reliability*, vol. 39, no. 5, pp. 524–534, December 1990.

37. Nordmann, L. and H. Pham. Weighted voting systems. *IEEE Trans. Reliability*, vol. 48, no. 1, pp. 42–49, March 1999.

38. Parhami, B. Voting networks. *IEEE Trans. Reliability*, vol. 40, no. 3, pp. 380–394, August 1991.

39. Parhami, B. A data-driven dependability assurance scheme with applications to data and design diversity. In *Dependable Computing for Critical Applications*, pp. 257–282, 1991.

40. Parhami, B. Threshold voting is fundamentally simpler than plurality voting. *Int'l J. Reliability, Quality, and Safety Engineering*, vol. 1, no. 1, pp. 95–102, March 1994.

41. Parhami, B. A multi-level view of dependable computing. *Computers & Electrical Engineering*, vol. 20, no. 4, pp. 347–368, 1994.

42. Parhami, B. Voting algorithms. *IEEE Trans. Reliability*, vol. 43, no. 4, pp. 617–629, December 1994.

43. Parhami, B. Multi-sensor data fusion and reliable multi-channel computation: unifying concepts and techniques. *Proc. 29th Asilomar Conf. Signals, Systems, and Computers*, pp. 745–749, October 1995.

44. Parhami, B. A taxonomy of voting schemes for data fusion and dependable computation. *Reliability Engineering and System Safety*, vol. 52, no. 2, pp. 139–151, May 1996.

45. Parhami, B. Parallel threshold voting. *The Computer J.*, vol. 39, no. 8, pp. 692–700, 1996.

46. Parhami, B. *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford, 2000.

47. Parhami, B. Approach to component based synthesis of fault-tolerant software. *Informatica*, vol. 25, no. 4, pp. 533–543, November 2001.

48. Pawlak, Z., J.F. Peters, A. Skowron, Z. Suraj, S. Ramanna, and M. Bokowski. Rough measures, rough integrals and sensor fusion. In *Rough Set Theory and Granular Computing*. (edited by M. Inuiguchi, S. Hirano, and S. Tsumoto) pp. 263–272, Springer, 2003.

49. Parra-Loera, R., W.E. Thompson, and A.P. Salvi. Adaptive selection of sensors based on individual performances in a multisensor environment. In *Data Structures and Target Classification*, SPIE Proc., vol. 1470, pp. 30–36, April 1991.

50. Pierce, W.H. Adaptive vote-takers improve the use of redundancy. In *Redundancy Techniques for Computing Systems* (edited by R.H. Wilcox and W.C. Mann), Spartan, pp. 229–250, 1962.

51. Prasad, L., S.S. Iyengar, R.L. Kashyap, and R.N. Madan. Functional characterization of fault tolerant integration in distributed sensor networks, *IEEE Trans. Systems, Man, and Cybernetics*, vol. 21, pp. 1082–1087, September 1991.

52. Rothman, P.L. and R.W. Denton. Fusion or confusion: knowledge or nonsense? In *Data Structures and Target Classification*, SPIE Proc., vol. 1470, pp. 2–12, April 1991.

53. Saari, D. *The Geometry of Voting*, Springer, 1994.

54. Shin, K.G. and J.W. Dolter. Alternative majority-voting methods for real-time computing systems. *IEEE Trans. Reliability*, vol. 38, no. 1, pp. 58–64, April 1989.

55. Siewiorek, D.P. Reliability modeling of compensating module failures in majority voted redundancy. *IEEE Trans. Computers*, vol. 24, no. 5, pp. 525–533, May 1975.

56. Spasojevic, M. and P. Berman. Voting as the optimal static pessimistic scheme for managing replicated data. *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 1, pp. 64–73, January 1994.

57. Tang, J. On multilevel voting. *Distributed Computing*, vol. 8, pp. 39–58, 1994.

58. Tanner, R. and N.K. Loh. A taxonomy of multisensor fusion. *J. Manufacturing Systems*, vol. 11, no. 5, pp. 314–325, 1992.

59. Tolle, J. Power distribution in four-player weighted voting systems. *Mathematics Magazine*, vol. 76, no. 1, pp. 33–39, February 2003.

60. von Neumann, J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Annals of Math. Studies*, no. 34, Princeton Univ. Press, pp. 43–98, 1956.

61. Waltz, E. and J. Llinas. *Multi-Sensor Data Fusion*, Artech House, 1990.

62. Xu, L. and J. Bruck. Deterministic voting in distributed systems using error-correcting codes. *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 8, pp. 813–824, August 1998.

63. Zadeh, L.A., K.S. Fu, K. Tanaka, and M. Shimura (Editors). *Fuzzy Sets and Their Applications to Cognitive and Decision Processes*, Academic Press, 1974.