PREDICTABLY DEPENDABLE COMPUTING SYSTEMS
TECHNICAL REPORT

PROJECT 3092 PDCS

PDCS No. 24
Site ref.

6 August 1990

ADJUDICATORS FOR DIVERSE-REDUNDANT COMPONENTS

F. Di Giandomenico, L. Strigini
IEI - CNR, Pisa

Abstract

In fault-tolerant components realized using replication, several alternate results are produced, from which a single correct result must be derived. This is done by a mechanism, that we call an *adjudicator* (a concept more general than a voter), which uses as its inputs the individual results produced by the replicated subcomponents comprising the redundant component.

This paper gives a rigourous definition of the adjudication problem, summarizes the existing literature on the topic, and investigates the use of probabilistic knowledge about errors/faults in the subcomponents of a fault tolerant component to obtain good adjudication functions. The concept of an *optimal adjudicator* is introduced and discussed.

Release number	1
Release status	Final
Availability status	Public
Available from	Felicita Di Giandomenico, IEI-CNR, Via S. Maria 46, 56126 Pisa, Italy.

A shorter version of this report has been accepted for presentation at the Ninth Symposium on Reliable Distributed Systems, Huntsville, Alabama, October 9-11, 1990.

ADJUDICATORS FOR DIVERSE-REDUNDANT COMPONENTS

F. Di Giandomenico   L. Strigini

I.E.I.- CNR,
Via Santa Maria, 46 -56126 Pisa, Italy

Abstract

In fault-tolerant components realized using replication, several alternate results are produced, from which a single correct result must be derived. This is done by a mechanism, that we call an *adjudicator* (a concept more general than a voter), which uses as its inputs the individual results produced by the replicated subcomponents comprising the redundant component.

This paper gives a rigourous definition of the adjudication problem, summarizes the existing literature on the topic, and investigates the use of probabilistic knowledge about errors/faults in the subcomponents of a fault tolerant component to obtain good adjudication functions. The concept of an *optimal adjudicator* is introduced and discussed.

1 Introduction

A technique for making a system capable of tolerating faults in its hardware and/or software components is to use replicated hardware and/or software components in the system. A *modular-redundant component* (MRC) is substituted instead of of an ordinary component, and consists of:

1) a set of subcomponents (that we call *replicas*), each one implementing the same function as the whole component, plus

2) some mechanism that, from the set of results produced by the replicas, obtains a single result to be used as the output of the replicated component.

It is convenient to depict this mechanism as an additional component called an *adjudicator* [3]. In general, the adjudicator will receive a subset of the outputs due from the replicas, and from these it must either determine a single result or, if this is not possible, signal an exception. The result of the adjudicator becomes the result of the redundant component.

This paper considers the problem of designing adjudicators, so as to optimize the behaviour of redundant components, taking into account the fault hypotheses made about the system. Most of our considerations are independent of whether the replicas and the adjudicators are implemented by hardware or software.

The most obvious kind of adjudication function is the *m-out-of-n majority voting*, in which, if at least m replicas out of the n constituting the MRC (usually n=2m-1, m=2,3,..) produce the same result value, that value is taken as the correct result. This works well if (sufficient condition): i) no more than (n-m) replicas produce erroneous results; and ii) all correct results are identical.

Several causes have led designers to invent more complicated adjudication functions:

- distinct replicas may produce different correct results. This may be caused by design diversity among the replicas[1] (introduced to tolerate design faults [6, 19]), by random, expected errors (e.g. if the replicas are sensors measuring some physical parameter of the outside world), by foreseeable effects of slight differences among the inputs to the replicas (due for instance to their inputs being the results non simultaneous sampling of some physical variable);
- assuming restrictive hypotheses on errors just to prove the correctness of adjudication results is inconvenient;
- in most computer systems, more information is available about the trustworthiness of individual results than is given by their comparison: error detection mechanisms in the hardware, acceptance tests on results in the software, and such. Such additional information should ideally allow one to obtain correct results under less restrictive fault hypotheses.

The problem of evaluating adjudication functions has not been studied systematically. Studies of the dependability of diverse-redundant components do exist (most recently, [4]), but, if they model sometimes the effects of an unreliable implementation of the adjudicator, they do not model the effects of choosing different designs for the adjudicator.

Most authors only compare few similar adjudication functions, and have not attempted to use a uniform set of hypotheses allowing comparisons with other authors. In particular, most authors only point out individual error situations where some adjudication function can be shown to be better than some other, and none of those we surveyed uses probabilities.

The rest of this paper is organized as follows. In section 2 we give more rigourous definitions of modular-redundant components and adjudicators, and an explanatory example. In section 3, a number of adjudication functions from the literature are surveyed. In section 4, we discuss general criteria for the comparison and evaluation of adjudication functions, and prove that an *optimal* adjudication function can be defined. This result sets an upper bound on the reliability obtainable from an MRC, given the information input to the adjudicator and the a-priori probabilities of all error situations in the system; and can be used to orient design decisions even if direct implementation of the optimal function is unfeasible. Our conclusions are in Section 5.

2 Description of the problem.

2.1 Definitions.

2.1.1. Component specification. A software component is defined in terms of its external behaviour, i.e., the values of the outputs it produces as a consequence of receiving inputs.

The life of a software component is a series of one or more invocations. During each invocation the component interacts with the rest of the world via inputs and outputs. We shall consider all the information received by a component during an invocation, and all the information emitted by it, as *two* items of information, called the value of the input vector and

[1]One might contend that this will disappear with increasing usage of formal specification methods, which unambiguously define the desired results. We disagree, because, for many applications, a set or range of results are acceptable, and requiring that one specific result from this set be produced would be overspecification, artificially limiting the algorithmic diversity attainable. In particular, numerical problems are typically such that i) there is a range of acceptable results around the ideal correct result (often improducible with the finite representations used in computers), and ii) requiring that all results of diverse algorithms are equal would imply that either the algorithms must be identical or they must always produce very close approximations of the ideal result, which is an unjustified added cost.

the value of the output vector for that invocation. Their precise nature is not relevant for our purpose. It is sufficient to say that the *input* and the *output* (or *input vector* and *output vector*, for clarity) of a component are each a set of mathematical variables, obtainable by observing the system according to rules specific to each individual hardware or software component. For instance, the values of the output variables of a procedure may be obtained by observing the state of the stack when control returns from the procedure to the calling program. An *input* (*output*) *value* is a value of the *input* (*output*) *vector*, i.e. a set of one value for each input (output) variable(we shall often use the word "input (output)" instead of "input (output) value"). The *input space* and the *output space* are the sets of all the possible input and output values, respectively, and will be denoted by the names I and O.

2.1.2. Specification function of a component. The specification of a component includes a *specification function* F that maps elements of I into elements of the set of the subsets of O. For each $i \in I$, F(i) indicates a set of *correct* outputs, contained in O. For instance, in a Cartesian space, such sets of correct outputs for an input value could be shaped as 'hyperrectangles' (correct outputs lie within a multidimensional interval around an ideal output), or 'hyperplanes' (one component of the output is mandated, the others are 'don't care'), or such like.

It is considered good practice for F to be defined for every $i \in I$. We shall normally assume this to be the case. The usual real case where there is a subset I^u of I where F is not explicitly defined by the specifiers can be described by saying that over I^u any output is permitted, i.e. defining F(i)=O for any i in I^u.

Normally, the specification function defines not only the ideal result that a component should give if all goes well, but also the behaviour of the component in exceptional situations, as, for example, in presence of illegal inputs or of some internal cause, typically a fault. For these cases, *exceptional output values* (which are perfectly legal outputs) are specified, with a more or less detailed description of which conditions (not limited to the inputs, but including internal situations of the component) should produce each distinct exceptional output.

2.1.3. Modular-redundant component. A *modular-redundant component* (MRC) is a component that contains several *replicas* built to the same specification as itself, but is *in some sense better* than any of them. More precisely:

- the specification function of the MRC may differ from that of the replicas in the requirement of diagnostic outputs (as notification of some exceptional situation), and in the existence of diagnostic inputs (to be used by the adjudicator in making its decision);
- replicas may differ among themselves, provided that they are built to the same specification function; replicas that are identical copies of a same component are said to belong to the same *variant*;
- the "better behaviour" requested from the MRC consists in a better ability to behave "close" to its specification, even when something undesirable happens. We tentatively call this property *robustness* .

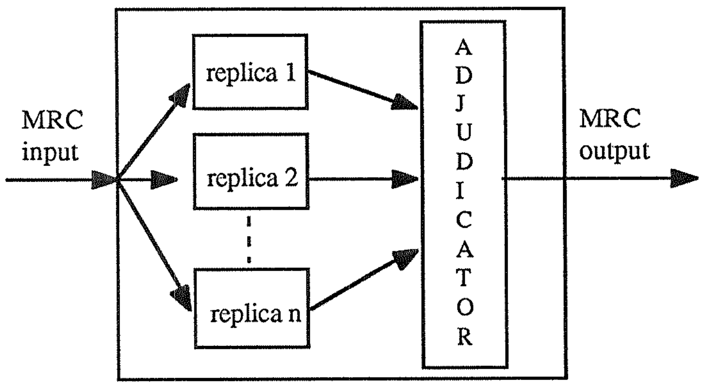
To better specify this concept, we can define a number of orderings (partial or total) among components with the same specification function, which could be called "A is more robust than B". "A is more robust than B" means "A produces correct results on a larger subset of the input space", where "larger" may mean "with higher cardinality", "with a greater probability mass", or "a proper superset of".

A more general definition, taking into account the existence of exceptional outputs, and of internal faults, and the actual consequences of erroneous outputs from the components, would start with defining, for each input value, a partition on the output space O and assigning a worth to each subset in the partition (or at least an ordering of relative worth). Once such an ordering is defined, "A is more robust than B" if the worth of A's output is greater than the worth of B's output in some statistical sense.

2.1.4. Stylized modular-redundant component. A simplified representation of a MRC is in the figure below:

- there is one *adjudicator* component, receiving the outputs of all the replicas;

- the inputs to the MRC are used as inputs by all the replicas; i.e., the underlying system guarantees that all the replicas normally receive identical inputs.



A stylized Modular-redundant Component (MRC).

This simplified scheme could represent an actual implementation. In practice, in all MRC schemes with an identifiable adjudication module, the adjudicator is assumed to run in a separate, protected hardware component, or to be replicated in several hardware components, so that it can be considered fault-free.

In the Delta-4 architecture, this protected hardware is called a fail-silent Network Attachment Controller. Throughout the rest of this paper, we shall call the output of the adjudicator the *adjudged output*, r*, and the outputs of the replicas the *replica outputs* (the set of the replica outputs will be indicated as $R=\{r_1, .., r_n\}$).

We ignore for the time being the details of how the system ensures that all the replicas receive consistent inputs, and that the adjudicator receives all the replica outputs. In particular, we ignore the problem of synchronization: how to make sure that the replica outputs from the same invocation of the replicas are collected, and how to deal with late or missing replica outputs. All these are general and difficult problems, but they are out of the scope of our immediate interest, which is the specification of adjudicator components. Once we have defined a range of possible specifications, and their respective merits, we shall consider their requirements for the services provided by the rest of the system, and the attending costs will contribute to determine the choice of a specification.For the time being, we shall assume that a stylized MRC can be built. In particular, with regard to the synchronization problem, we assume that the system will react to late delivery of results by replicas by supplying the adjudicator arbitrary (or exceptional) values instead of the corresponding replica outputs.

2.1.5. Specification of the adjudication function. The adjudication function is the function implemented in the adjudication mechanisms (or, the specification function of the adjudicator module when considering a stylized MRC as in the figure). It is defined on the input universe composed of the possible values of the input vector of the adjudicator. This input vector, or *syndrome*, consists of the replica outputs plus, possibly, additional information (such as the results of acceptance tests performed on the individual results), or some reduced set of information extracted from them, e.g., the set of results of their pairwise comparisons.

The purpose of the adjudication function is to select a correct output for the MRC, i.e., an output value that satisfies the specification function of the MRC. So, the specification for an adjudication function can be stated as follows.

The adjudicator, for a given syndrome s, must produce an output that the specification function of the MRC prescribes for that input i for which (given the values of the other factors -like faults - during the invocation) the replicas produce the syndrome s. It is evident that:

1. the preference between two adjudication functions depends mainly on their robustness (besides considerations of cost and such), i.e., on which approximates more closely this specification;
2. this specification, though it describes what we want from an adjudication function, gives very little help in designing one. An adjudication function is usually sought that is applicable to MRCs of different types. We cannot incorporate in it knowledge about the specification of the MRC, nor assume knowledge of the input to the MRC.

2.2 Example: a simple adjudication problem.

2.2.1. Description. We describe here, as an example, a simple practical adjudication problem taken from the Delta-4 distributed architecture [9, 17]. Very similar hypotheses are used in [9]: [9] does not consider software variants, but rather groups of replicas executed with slightly different inputs, due to different activation times; this is immaterial for our discussion. The hypotheses are as follows:

1. The replicas are software components, running on a set of host computers (one or more replicas per host).
2. The MRCs in the system are stylized MRCs.
3. The replicas are *deterministic*: when correct, the output of a replica is a deterministic function of its input and the variant to which it belongs[1]. In other words, the results of two replicas belonging to the same software variant, if correct, are identical. On the other hand, we make no hypothesis on how faulty replicas behave: for instance, a software fault may produce different results by two replicas of the same variant.
4. An erroneous output from a replica may be caused by a hardware fault in its host and/or by a software fault in its variant (a faulty variant is one that would produce an erroneous output, at least on some invocation, when running on a non-faulty host).
5. *Non-triviality*of the specification : programs that, according to their specifications, may produce the same result for all inputs are excluded.
6. Each replica in the MRC performs an acceptance test on its own results. The syndrome used by the adjudicator is composed of the set of the n results of the self-checks and the set of the results of pairwise comparisons (for bit-by-bit equality) among the output values of all the replicas. Both are sets of Boolean values, where "0" means "disagreement" and "1" means "agreement". Besides these, the adjudicator knows the host and the variant of each replica.
7. As a further constraint, we require that the adjudicator produce as a result one of the outputs of the replicas, or an exceptional output when it cannot extract an adjudged output with reasonable confidence.

---

[1]Real software components are often non-deterministic, due to oversights in their implementation or to unavoidable causes of non-determinism (typically hidden in the operating system). Even considering the latter, non-determinism could ideally be specified if all the parts and transitions of the machine status that may affect the behaviour of the replicas in the absence of (design or physical) faults are considered as inputs, and, as stated above, the system is so built as to provide identical inputs to all the replicas. (Hence, once the inputs are defined, any program that is non-deterministic contains, by definition, a design fault). This requirement is satisfied, for instance:

- with respect to variables used before initialization, if such use is considered as a design fault, or made impossible by the compiler, or if their values are preset to a default value when the program is loaded, or considered as part of the input;
- with respect to the ordering of messages sent to a replicated software component, if the system guarantees that they are delivered in the same order to all replicas, or if all variants are written in such a way that their outputs do not depend on the order of reception of the messages.

2.2.2. The adjudication function. An adjudication function maps syndromes into adjudged outputs: it can be described as a table, where rows are indexed with the possible values of the syndrome and lists the results that the adjudicator chooses.

In our example, assuming for instance that we have only three replicas, and their results are called r_i and the results of the acceptance tests are called Ai (i=1,2,3), the set of possible syndromes is represented by the following table, where $r_{i,j}$ is the result of comparison between the replica outputs r_i and r_j.

r1,2	r2,3	r1,3	A1	A2	A3
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
..	..	..	..	..	..
..	..	..	..	..	..
1	1	1	1	1	1

Table 1

Now, in choosing an adjudication function, some assignments would be obvious, as, for instance:

r1,2	r2,3	r3,1	A1	A2	A3	r*
1	1	1	1	1	1	r1
..	..	..	..	..	..	..
1	0	0	1	1	0	r1

Table 2

These are based on obvious probabilistic assumptions: if a majority exists, and the results in the majority pass their acceptance tests, then they are the most likely to be correct. But more ambiguous syndromes may occur, for instance:

r1,2	r2,3	r3,1	A1	A2	A3	r*
1	0	0	1	0	1	?

Table 3

In this syndrome, r_1 and r_2 form a majority (which is generally taken as a hint that they are correct), but r_2 does not pass the acceptance test. Several interpretations are possible:

- the acceptance test A_2 is erroneous, and r_1 and r_2 are correct. Then indifferently r_1 or r_2 can be taken (correctly) as r*;
- if r_1 and r_2 are erroneous and r_3 is correct, then A_1 is erroneous too, and the correct result for the adjudicator is r_3;
- if r_1, r_2 and r_3 are all erroneous, then A_1 and A_3 are erroneous too. In this case, the adjudged output should be an exceptional output.

To choose among these interpretations, we could use such probabilistic considerations as:

- many acceptance tests are more trustworthy when they diagnose a result as erroneous than when they pass it as correct. So, A_2 constitutes more important evidence than A_1 or A_3;
- if replicas 1 and 2 run on separate hardware modules and belong to diverse software variants, and there are many correct outputs for this value of the input, the probability of their failing in identical ways is lower;
- if we have previous evidence of malfunctioning from replica 3, then we should not accept r_3 lightly;

- if the cost of the adjudged output being an exceptional value is low, maybe that is the best solution.

These considerations can only be substantiated by more information about how the MRC is built and about its probabilistic behaviour. We shall now examine this issue in closer detail.

The information necessary to resolve ambiguities is, simply, which replica outputs are in error during the execution producing the ambiguous syndrome.

Let us define the error state E, for a given execution of the MRC, as a vector of n positions, with E_i equal 1 if the replica output r_i is correct, 0 otherwise. Our adjudication table could now be substituted with a table in which there is a row for each possible value of the error state and the syndrome. The problem is that all rows with the same syndrome must have the same adjudged result, while the correct results may be different, depending on the values of the error state. Table 4 shows those rows of this new table that correspond to the syndrome in Table 3. Note that only 3 errors states (out of 8) are compatible with the observed syndrome, and that the value of the error state, as defined, together with the syndrome, determines which acceptance tests are correct. For instance, E_2=1 and A_2=0 imply that the acceptance test for replica 1 is in error.

Error state			Syndrome							
E_1	E_2	E_3	$r_{1,2}$	$r_{2,3}$	$r_{3,1}$	A_1	A_2	A_3	correct r	r*
1	1	0	1	0	0	1	0	1	r_1 or r_2	?
0	0	1	1	0	0	1	0	1	r_3	?
0	0	0	1	0	0	1	0	1	exception	?

Table 4

Which is the correct value for the observed syndrome depends on the value of the error state during the execution producing this syndrome. This cannot be known with certainty. The best that can be done is to assume as the actual error state the one having the highest probability. So, we add probabilities to each row in the table of the adjudication. For our example, Table 4 could become, for instance:

	Error state			Syndrome							
P	E_1	E_2	E_3	$r_{1,2}$	$r_{2,3}$	$r_{3,1}$	A_1	A_2	A_3	correct r	r*
x>y+z	1	1	0	1	0	0	1	0	1	r_1 or r_2	r_1
y	0	0	1	1	0	0	1	0	1	r_3	or
z	0	0	0	1	0	0	1	0	1	exception	r_2

Table 5

The best choice for the adjudged output is a replica output produced by a replica that would be correct for the most probable error state. So, r_1 or r_2 are the most probably correct replica outputs.

The next section contains a survey of adjudication functions from the existing literature. , and their implicit or explicit assumptions about the distribution of error states. Authors do not generally provide a complete analysis as shown above, so that it is often necessary to guess the assumptions underlying their choices.In Section 4, we shall widen the scope of our analysis to obtain general evaluation criteria for adjudication functions.

3 Survey of adjudication functions.

In this section, we survey the literature about adjudication functions. Where possible, we provide observations about similarities, limits and other aspects of their behaviour. Most of these have been originally described referring to programs whose outputs are either numbers or vectors in a Cartesian space. Actually, many of them can be defined on more general spaces (typically, all spaces on which either an ordering or a distance function[1] is defined), preserving some or all of their properties. This analysis is omitted here for reasons of space. They also depend on assumptions about the specification function..

3.1 Adjudication functions using only the outputs of the replicas.

The first generation of adjudication functions worked only on the set R of replica outputs to select the adjudged output r*. In this class of adjudication functions are:

Exact (bit-by-bit) majority voting. This adjudication function selects r* as the majority value among the replica outputs (absolute majority, $\lceil (n+1)/2 \rceil$, or some other kind of majority), if such a value exists. r* is a correct value only if it is produced by a majority of correct replicas.

The limits of this adjudication function are immediately evident:

- design diversity allows two results which are both correct but different;
- it can be required that a correct result be guessed also when less than a majority of replicas is correct.

Median adjudication function. It can be defined for outputs consisting of a single value in an ordered space (e.g. real numbers). The median value among the replica outputs is selected, i.e., the value whose position is central in the set R (if n is odd), otherwise the value in position n/2 or (n/2+1) (if n is even). Assuming that, for each input value, no incorrect result lies between two correct results (there is an interval in the output space that contains all and only the correct output values), and that a majority of the replica outputs are correct, this function produces a correct adjudged output.

It can be extended to non-scalar values, provided the output space is a metric space: the algorithm (*generalized median voter* in [15]) consists in repeatedly selecting the pair of replica outputs with maximum reciprocal distance and discarding them, until only one replica output is left, and using this as the adjudged output. However, an absolute majority of correct replica outputs no longer guarantees correct adjudication, unless all correct values coincide.

The median adjudication function can be proven to yield a correct result under limiting hypothesis on the number of incorrect results.

Mean adjudication function [8]. If we assume that the probabilities of the values of the replica outputs decrease with increasing distances from the ideal result (their *error*, in the terminology of physical measurements), we are led to choose a mean adjudication function [8]. The syndrome is constituted by the set of replica outputs, and r* is selected as the mean value from this set.

Let ε be the maximum distance between a replica output and the ideal correct output (this definition will be valid throughout the rest of the paper).

Instead of the simple mean, a weighted average of the replica outputs can be used. The weights can be assigned in different ways, using additional information related to the trustworthiness of the replicas, and known a-priori (and perhaps continually updated) or defined directly at invocation time (as, for example, by assigning replica outputs weights inversely proportional to their distances from all the other results).

[1]A distance function is a function D defined over the sets of couples (a, b) in a set, such that D(a,b)=0 iff a=b, and D(a,b)+D(b,c) ≥ D(a,c). Simple examples of distance functions are: the Euclidean distance between vectors, $D([V_1,..,V_k], [W_1,..,W_k])=(\sum_{i=1}^{k}(V_i-W_i)^2)^{1/2}$ a function D(a, b) that is 0 iff a=b and 1 otherwise.

Another example is defined for character strings in [5], so that different strings may be considered equivalent if they only differ in minor misspellings or non significant characters (like excess blank characters).

Actually, when the function that assigns the weights depends on the past weights as well as on the current results, two items of information are updated at each invocation of the adjudicator: the weight to be assigned to each replica output for the current invocation and the estimate of the trustworthiness of the replicas for future invocations (diagnostic information). The two items are quite different. For instance, if the replica outputs are the outputs of noisy channels, a value which is occasionally very distant from the others, but within the expected amplitude of the noise, should be considered in an adjudication, but, if repeated systematically, should affect the diagnosis of its channel.

3.2 Adjudication functions using additional information besides the replica outputs.

A first enrichment to the information known to the adjudication function, if the output space is a metric space, is a parameter ζ, indicating the maximum distance allowed between two correct output values for the same input (based on the specification function). Alternatively (as, typically, in numerical algorithms), the allowed outputs may be specified as an ideal output and a maximum distance, δ, from it (then, ζ would be equal to 2δ). Adjudication functions using such parameters rely on the (usually realistic) hypothesis that different correct implementations of a same specification function can produce results that are different, but quite close together.

Additional information can be obtained from diagnostic mechanisms, acceptance tests, reasonableness tests and such.

Among the proposed adjudication functions that exploit such information, it is useful to distinguish *two-step adjudication functions*, where the steps consist in :

- defining a *feasibility set* FS, by discarding the replica outputs judged to be erroneous by some criterion. Usual criteria are the acceptance tests, or a requirement that all elements in the FS differ by less than ζ;
- choosing r* among the values in FS. Possible criteria for the choice are again the majority, or the median, or the average, or a random selection and so on.

Of course, the good behaviour of these adjudication functions depends on a good choice of the value for ζ: if it is too large, the selected FS may include incorrect values, and one of these may be selected as r* (in particular, if r* is randomly selected in FS). On the other hand, if ζ is chosen too small, it may happen that no feasibility set is found, even when there are enough correct values.

3.2.1. Two-step adjudication functions.

Formalized majority voter [15]. This voter starts with the selection of a replica output, x, and constructs the feasibility set FS as the subset of R containing x and all the other values in R differing from x by less than ζ. If the values in FS constitute at least a majority of all the values in R, then the value r* is randomly selected from FS.

Formalized plurality voter [15]. It is a generalization of the formalized majority voter. It defines a partition of the set R such that each couple of replica outputs in each subset of this partition differ by at most ζ, and the subset can not be augmented with additional elements while preserving that property. Then, one of these subsets is chosen as the FS, if possible, for instance the one with the highest cardinality, or anyone that has cardinality higher than an assigned value. r* is randomly selected from the FS. This algorithm is non-deterministic: for a same syndrome, different choices of the initial replica output x may lead to find a FS or to terminate unsuccessfully. So, for such adjudicators to produce a correct r*, it is necessary that a FS with the required cardinality exists and that it is found; and, once it is found, a sufficient condition for r* to be correct is that all the replica outputs in the FS be correct. Unfortunately, no simple sufficient condition is known for this last property.

Other two-step adjudication functions apply more complex criteria to define the FS and/or to select from it the value r*. They are:

MCNV (modified interactive convergence algorithm) in [16]. Instead of defining a FS as a subset of R, MCNV examines the couples of replica outputs and builds a set C of all those couples $\{r_i, r_j\}$ such that $|r_i - r_j| \leq \zeta$. From the set C, the replica output, r', that is present in the highest number of couples is selected as the "most reliable" replica output, and r* is obtained by adding to r' the mean of its differences from all the replica outputs that differ from it by at most ζ. r* is a correct result under the usual sufficient condition that a majority of the replicas are correct.

SLIDE (sliding majority decision algorithm) in [16]. It defines FS as the subset having the highest cardinality among the (non disjoint) subsets of R in which each couple of results differ for at most ζ (if more than one such subsets exist with the same cardinality, FS is defined as the union of these). r* is then obtained as the median value in FS, or the median adjusted by the average of its differences from the other elements in FS, or the average of all values in FS.

Filter strategy [9]. The set FS is constituted by the replica outputs judged good by their acceptance tests. Then, r* is selected as the majority value in FS. This adjudication function does not require that O be a Cartesian space, but its success depends on the quality of the acceptance tests; besides, applying the majority criterion only helps if correct replica outputs are often identical.

Among complex adjudication functions that do not use the notion of a FS, there is the one proposed for numerical values in the *DEDIX system*, in [5]. In the first step, it computes the median values among the replica outputs, say r', and among n values of δ which are proposed by the n replicas, say δ'. In the second step, r' is chosen as r* if there is a majority of values in R differing from r' by at most δ', otherwise an exception is declared.

The adjudication function in the so-called *consensus recovery block* [18] first compares the replica outputs and, if at least two are identical, selects their value as the adjudged result. If all the replica outputs are different, they are submitted to an acceptance test, in order of decreasing estimated reliability, and the first that passes the acceptance test is used as the adjudged result. If no replica output passes the acceptance test, a failure is declared. Of course, any common-mode failure among two replicas will beat this strategy. In a variation discussed in [9], a majority of identical results is required in the first phase, and, if it does not exist, the acceptance tests are used. Again, this strategy is vulnerable to common mode failures resulting in equal incorrect results, and fails to use the acceptance test results as a defense against such occurrences.

3.2.2. Other adjudication functions.

Counter strategy [9]. A score is assigned to each replica output, given by the number of other replica outputs that are equal to it, plus an additional term *a* when the replica output passes its acceptance test (*a* is taken as a design parameter). The replica output with the highest score is chosen as r*.

A further refinement consists in varying the weights of both the results of comparisons and of acceptance tests to:

i) reduce the importance of comparisons between replicas belonging to different variants, compared to comparisons within a same variant[1]; and

ii) increase the weight associated to the replica outputs coming from variants with fewer replicas, to compensate for the "advantage" that the algorithm would otherwise give variants that have more replicas.

Stepwise negotiating voting [11]. It uses a-priori, diagnostic information on the system state. It is "negotiated" in that the corroboration required for an adjudged output can be varied accord-

[1]This is done in order to take into account the possibility of different correct replica outputs; unfortunately, it also neglects the evidence for correctness given by identical outputs from diverse replicas.

ingly to the criticality of the output of the MRC for the system. For example, it can be required that the values of all (or an assigned number of) replicas agree and the replicas are judged non-faulty, or that only one of the two conditions be met, and so on.

Confidence voter [12]. This adjudication function generally applies the majority criterion, but is able to work with an even number of replicas, resolving split vote situations where two different results are proposed by equal numbers of replicas. It is described for a system of four components. The confidence voter maintains a record of the frequency of coincident failures for each pair of replicas (when 2-1-1 syndromes are observed, that is, two replica outputs are equal and the other two differ from these and from each other, the latter two replicas are deemed to suffer a coincident failure). For a 2-2 syndrome, the adjudication function selects r* as the replica output produced by the couple having the lowest frequency of coincident error.

3.2.3. Final remarks. It is not possible, in general, to give a complete, comparative evaluation of the behaviour of the complex adjudication functions described in section 3.2: they cannot be described by a simple mathematical function. The authors themselves often evaluate their behaviour only in a few specific situations.

In comparison with the simpler adjudication functions described in 3.1, these surely lose in simplicity, but this is supposedly compensated by an improvement of the adjudication.

A direction that has not been investigated in the surveyed literature is that of introducing probabilistic information, both in the definition of the adjudication function and in the criterion for the evaluation of different adjudication functions.

4 Evaluation of adjudication functions.

4.1 Realistic evaluation of adjudication functions.

To choose among different candidate adjudication functions for a system, it is necessary to have a realistic evaluation criterion. Given a system configuration, one can first observe the behaviour of different adjudication functions on a number of ideal experiments. However, there is no guarantee that an adjudicator is found to be consistently the best for every syndrome. In fact, it is impossible, without any further knowledge, to choose between two adjudicators if one behaves better for a syndrome and the other for another syndrome.

The most realistic comparison criterion is that of comparing the probabilities that each adjudication function has of producing a correct adjudged output for the MRC.

Based on the definitions given in Section 2, and defining STATES as the set of all possible values for the *system state* variables (including the syndrome) that determine which adjudged output is correct, the probability that an adjudication function $F_{adj\text{-}k}$ produces a correct adjudged output is:

$$P(\mathit{correct}\ (F_{adj\text{-}k})) = \sum_{x \in STATES} P(x)*V_k(x) \qquad (1)$$

where:
- P(x) is the probability of occurrence of the system state x;
- $V_k(x)$ is a binary function having value "1" iff the adjudication function $F_{adj\text{-}k}$ produces, for the syndrome implied by the system state x, an output that is correct for system state x.

(1) can be rewritten in terms of syndromes as:

$$P(\mathit{correct}\ (F_{adj\text{-}k})) = \sum_{s \in SYN} P(\mathit{correct}\ (F_{adj\text{-}k}) \cap s), \quad (2)$$

where s is the event "the value of the syndrome is s", and:

$$P(\mathit{correct}\ (F_{adj\text{-}k}) \cap s) = \sum_{x \in States(s)} P(x)*V_k(x) \qquad (3)$$

where States(s) is the set of states that imply the syndrome s.

Supposing we know the system configuration, and information about probabilities of relevant events in the system, the total probability that an adjudication function produces a correct result, with the given system configuration and assigned probabilities, is found by enumerating all the system states and computing expression (3). In terms of the tables used in our examples in Section 2, this procedure consists in selecting all the rows in Table 5 where r* is equal to the correct result, and summing the probabilities associated to those rows.

Of course, this is an idealized procedure. Two considerations are important here:

- the probabilities of the system states can only be defined with respect to some given experiment: this implies specifying the distribution of inputs to the MRC, of faults and all other factors that influence the behaviour of the MRC;
- none of these probabilities can be known with certainty.

So, the procedure described is not a simplification with respect to an experimental test of an adjudicator. It is just a description in probabilistic and analytical terms of what, in an actual test, would be described in synthetic and statistical terms. On the other hand, given our uncertain knowledge of the values of the parameters involved, we can run thought experiments, choosing consistent values of the parameters, and evaluating the sensitivities of the final result to these different parameters. The parameter values must be consistent both with the axioms of probabilities and with additional hypotheses we make on the system: this is how The effects of specific assumptions, like the independence of certain errors or faults, can thus be tested.

4.2 The probability assignment problem.

The main problem in this evaluation is the difficulty of finding the correct probabilities of all the error states. Typically, error statistics cannot be collected on a complex system (in particular if the system does not yet exist!) with the levels of both confidence and detail needed: rather, statistics about the behaviour of individual components are known or estimated from previous experience. In particular, fault statistics of individual components must be used to estimate their failure probabilities (that is, the probability that the fault produces an observable error) in the system under consideration.

In a typical real-world system, as the distributed system used in our examples, we may know estimates for such probabilistic parameters as:

- hardware fault rates for the hosts of the system;
- some idea of the joint probabilities of hardware faults in different nodes (or the knowledge that they are independent);
- software error probabilities for each variant (estimated for the expected distribution of inputs);
- "something" (normally very little) about joint error probabilities for different software variants;;For highly reliable software, observing failures is a rare event, and observing common failures is (if design diversity is of any use at all) even rarer.
- something about error probabilities for the self-checks of the replicas, and the probabilities of joint errors with the replicas themselves. Typically, we could assign inherent error probabilities to the acceptance tests conditional on the correctness or erroneousness of the replica outputs;
- etc.

So, we are more likely to know some fault probabilities than the error probabilities listed in Table 5. If we also knew such probabilities as, e.g., the probability that a hardware fault produce an error in the software component running on it, we could then derive the probabilities of error states.

But considering the probabilities of faults as well enlarges the problem further. We can define a *fault state* as a vector with one bit for each processor used by the MRC plus one bit for each replica variant utilized plus one bit for each variant of the acceptance test. An element of this vector is 1 if the component associated to its position is not faulty, 0 otherwise.

Given a value for the fault state vector, consequent errors can manifest themselves in different ways, e.g.: a fault in a processor running more than one software component can produce an error only in some of these software components; a fault in a variant can produce different or equal erroneous replica outputs. So, different values of the fault state can be associated to the same value of the error state.

Going back to the example of Table 5, let us consider, for the sake of simplicity, that the three replicas belong to a same variant v_r, and the three acceptance tests belong to a same variant v_a. Besides, each replica runs on a distinct host, together with its acceptance test.

Table 6 represents again the states corresponding to the syndrome in Tables 4 and 5, completed with the fault state vectors. The assumption has been made that a correct replica output implies a non-faulty variant (no fault compensation between hardware and software): this is why, in the first two rows, the error state and syndrome imply only one possible fault state.

Additional symbols are used in the columns corresponding to processors, to underscore the different effects of faults implied by the values of the error states on the same rows:

- 0^r, indicating that a fault in the processor produces an error only in the replica it runs;
- 0^a, indicating that a fault in the processor produces an error only in the acceptance test it runs;
- 0, indicating that a fault in the processor produces an error both in the replica and in the acceptance test.

	Fault state					Error state			Syndrome						
P	p_1	p_2	p_3	v_r	v_a	E_1	E_2	E_3	$r_{1,2}$	$r_{1,3}$	$r_{2,3}$	A_1	A_2	A_3	r^*
x	1	0^a	0	1	1	1	1	0							
y	0	0^r	1	1	1	0	0	1							
z_1	0	0^r	0	1	1										
z_2	0^a	1	0^a	0	1										
z_3	0	1	0^a	0	1				1	0	0	1	0	1	r_1
z_4	0^a	0^r	0^a	0	1										
z_5	0	0^r	0^a	0	1	0	0	0							
z_6	0^a	1	0	0	1										
z_7	0	1	0	0	1										
z_8	0^a	0^r	0	0	1										
z_9	0	0^r	0	0	1										

Table 6

The situation with three erroneous replica outputs can be due to many different fault states. In Table 5, a probability z had been assigned to this error state, but adding the information about the fault state, this probability z is decomposed into the nine probabilities $z_1, .., z_9$, having sum z.

The choice of the best adjudication function is very sensitive to the probability assignment. The probability that a result r produced by the adjudication function k for a given syndrome s is wrong is the sum of the probabilities of the error states possible with the syndrome s and for which the result r is wrong. For a new system, the assignment of probabilities is necessarily based on insufficient evidence for some parameters (fault probabilities of new components) and on mere guesses for others (probabilities of intricate dependencies among faults/errors). In choosing an adjudication function, a sensitivity analysis must of course be performed with respect to the assumed probabilities parameters.

The steps for choosing a good adjudication function to use in a given system are:

1 assuming some initial assignment of the probabilistic parameters. This depends on the particular system under examination (for example, low redundancy systems or ultra-high reliability systems), and can be chosen either as plausible in these systems or as

representative of some particular or pathological situations believed to be interesting (for example, totally symmetric or totally asymmetric assignments, strong dependencies or independence);

2 evaluating different adjudication functions for this probability assignment;

3 iterating, giving different values to the probabilities at point 1), to evaluate the sensitivity of the results of step 2) to these parameters. A satisfactory decision is sought with respect to both the probability of correct adjudged output and its insensitivity to errors in the hypotheses on the (expected) parameters.

Given the complexity of the problem as described here, it is worth noticing that, for a comparison between two adjudication functions, only those rows where the values of r* are different between the tables describing the two functions need to be taken into account. This reduces the size of the assignment problem, and may allow a rapid decision as to whether one of the two adjudicators is certainly better than the other, or whether no judgment is possible, given the uncertainties about the values of the parameters.

4.3 The optimal adjudication function.

As seen in the survey, most adjudication functions are based on limiting hypotheses on the faults present in the system. In reality, some applications (such as critical applications) may require that every fault combination be taken into account, because their effects are relevant to obtain the required reliability.

There is no adjudication function that can mask all conceivable fault combinations. This is easily derived from the considerations made in the Section 2: different possible error states can lead to the same observed syndrome. To obtain a correct forecast of the probability that the output of an adjudicator is correct, one needs to know the probabilities of all the combinations of error states and syndromes, as outlined above.

With this knowledge, one could produce an *optimal* adjudication function, i.e., one that has the highest theoretically possible probability of producing a correct result for any input to a particular MRC. *An adjudication function that for any syndrome chooses that result that has the highest probability of being correct, conditional on the occurrence of that syndrome, is such an optimal adjudication function.*

In fact, the probability that an adjudication function F_{adj} produces a correct adjudged output is given by

$$P(\mathit{correct}\,(F_{adj\text{-}k})) = \sum_{s \in SYN} P(\mathit{correct}\,(F_{adj\text{-}k}) \mid s) * P(s) \qquad (4)$$

That adjudication function that, for each syndrome s, chooses that output o* such that:

$$P(o^* \text{ is correct} \mid s) = \underset{o \in O}{MAX} \{P(o \text{ is correct} \mid s)\}$$

maximizes expression (4), and hence is the function that has the highest probability to select the correct value, that is the optimal adjudication function. Knowing the conditional probabilities needed is of course a hard problem.

The notion of "optimality" needs some detailing, regarding what is optimized, and under which constraints. What we mean here is that if the adjudication function is to be a deterministic function of a syndrome, and once a set of information has been chosen as the syndrome to be used, our optimal adjudication function maximizes the probability of a correct adjudged result on any individual invocation (from the distribution of invocations, i.e. of input values and error states, from which the parameters in (4) are derived). This is a general result, in that it does not depend on the choice of which information comprises the syndrome.

Of course, the optimal adjudication function is not always the best adjudication function to use in practice; for example, performance requirements could make it unusable (it requires more time than simpler adjudication functions). As for the possible uses of the concept of an optimal adjudication function, we can observe that:

- the probability that the ideal adjudication function produces a correct result is an upper bound on what can be obtained from any MRC. If the probability estimates for the parameters of the adjudication table are such that this bound is lower than the requirements for the MRC under considerations, the designer knows in advance that the components must be redesigned, possibly by providing additional relevant information to the adjudicator;
- other, simpler adjudication functions, that appear desirable for some practical reason, can be compared against the optimal adjudication function. If the difference in the probabilities of correct adjudged output is small, compared to the probable error due to the uncertainty on the estimation of the parameters, then the simpler adjudication function can be accepted for the design;
- implementing the adjudication function as a look-up table, as in our examples, makes it possible to change it towards optimality, based on information collected during the lifetime of the system. This can be applied on different time scales:
  - during a mission, the probabilities associated to different events can be reassigned based on observed symptoms. This is a formalized version of the "confidence voter" in [12], or an extension of the well-known concept of "self-purging redundancy": instead of disconnecting faulty component, one reduces the trust in components suspected of being faulty;
  - during the lifetime of a design, its trustworthiness can be thought to increase in time as successful use proceeds, and to decrease again after catastrophic failures or new releases. This information can be incorporated into successive versions of an optimal adjudicator.

4.4 Extension to more than two worth levels.

Let us consider again the case where the results produced by an MRC are graded not just as correct or erroneous, but with different levels of worth (or cost) to the system, as for instance when the output is classified as exceptional (fail-safe output), non exceptional and erroneous (catastrophic failure), or non-exceptional and correct (success).

Then, additional criteria are needed to guide the selection of an adjudication function. If we stated simply that a fail-safe output is always preferred to the chance of a catastrophic failure, the best adjudication function would be one with a constant output, equal to the fail-safe output. If we decided that a fail-safe output does not exist in our application, then we would go back to the two-worth-levels case discussed in the previous sections. To obtain a rational compromise between the two contrasting requirements, that we can loosely call "safety" and "reliability", it is natural to assign a numerical cost, or utility, to each type of output. More than two such levels would of course be possible.

The assessment of an adjudication function, then, is no longer based on its probability of producing a correct result, but on the expected value of its worth to the system (or cost). The evaluation of the probability of a correct result fits as a limiting case, in which the worth of a correct result is 1 and that of any other result is 0. Given an assignment of a utility (cost) function, the definition of the optimal adjudication function can easily be restated in terms of the expected value of this function.

Another aspect of this problem is how to transform an adjudication function, which has been defined without the possibility of fail-safe outputs, to include this possibility. Let us assume the simple case where only three cost levels exist, say, 0 for success, C_s for fail safe output, C_f for catastrophic failure. For each syndrome, the following decision criterion applies. Let r^* be the adjudged result previously chosen for the syndrome, and let P_r be its probability of being correct (conditional to the occurrence of that syndrome). Then, changing r^* to the fail-safe result would change the expected value of the cost to C_s (the cost of producing a fail-safe output with conditional probability 1) from a previous value of $(1-P_r)$ times C_f (the cost of catastrophic failures). The change should be made if

$$(1-P_r)C_f > C_s$$

i.e.,

$$1\text{-}P_r > C_s / C_f$$

When the cost of the fail-safe output is equal to that of the catastrophe, the change is never worthwhile; when the cost of the catastrophe is very high, only the fail-safe output should be produced (or, equivalently, the MRC is useless).

4.5 What is a proper utility function?

The considerations that led us to define the optimal adjudicator are all based on the idea that what is desired is a high probability of producing a correct result on any individual invocation, considered as isolated from the previous invocations in the history of the MRC. This is consistent with the attitude taken by most other authors, so that our results are applicable for improving the adjudicator designs proposed so far. Intuitive reasons for such an attitude are: i) the design choice of using as an input to the adjudicator only information pertaining to an individual invocation (for reasons of simplicity); ii) the idea that the utility of a series of invocations is the sum of the utilities of the individual invocation; iii) the implicit assumption that the parameters (inputs and error states) of individual invocations have independent distributions.

Both assumptions ii) and iii) are questionable. It is easy to imagine applications for which a reasonable utility function must be defined over a sequence (or a population) of invocations, and different from the sum of intuitive utility functions for individual invocations. For instance:

- in a flight-critical application with only two worth values for an individual invocation ('1' for correct result and '0' for incorrect result), a reasonable utility function for a whole mission is the product of all the utility values of the individual invocations during the mission;
- for a design that is used in many installed system, a global utility function should take into account the possibility that a few bad systems, even on a population with very good average performance, may ruin the whole market for that design.

As for the independence assumption, both the inputs to the MRC and the fault states are likely to be strongly correlated between consecutive invocations. This is quite obvious for physical faults. For inputs to the MRC, non-independence can be easily argued, for instance, for a control system, where inputs are subsequent samples of time-continuous physical variables. As for software faults, we can expect the subsets of the input space I where the software fails to be (at least if I is a Cartesian space) multi-dimensional 'blobs' in I (see [1, 2, 7] for experimental observation of such 'blobs', and [10, 13, 14] for discussions of failure correlation issues in diverse software variants). Given non-independence among consecutive inputs, this implies that software errors in subsequent invocations are not independent either.

In practical terms, there is a trade-off between the benefit obtainable by a design closer to (theoretical) optimality and the cost of designing and operating such a design. Most components are built striving for a good individual and local behaviour, in the hope that this will improve global system behaviour, and this is usually shown to be the case.

5 Conclusions

We have:

- defined and discussed the problem of adjudication in modular-redundant components;
- reviewed the motivations and characteristics of the various proposals existent in the literature, and given a general method for their comparison;
- proven the existence of an optimal adjudication function, which is useful both as an upper bound on the probability of correct adjudged output obtainable and as a guide for design decisions.

Several directions are open for further research:

- implementation of a general tool for the evaluation of adjudication functions. This tool should help the user in defining consistent sets of probabilistic error hypotheses and running thought experiments to observe the reliability of the adjudication function and its sensitivity to the hypotheses made;

- study of different adjudication functions under different, simple error hypotheses to obtain boundaries to the applicability of the comparison criteria used in literature, while avoiding the difficulties of a complete probabilistic evaluation;
- study of the applications and implementation of the concept of optimal adjudication function;
- extension of this study to consider implementation constraints such as computational complexity and communication problems (reliability and complexity).

Acknowledgments

The authors would like to acknowledge the critical contributions of their colleagues in the ESPRIT PDCS project, in particular Jean Arlat, Sarah Broklehurst, Karama Kanoun, Jean-Claude Laprie, Bev Littlewood, David Wright.

References

[1] Amman, P. E., and J.C. Knight, "Data Diversity: an Approach to Software Fault-Tolerance", Proc. 17th FTCS, Pittsburgh, Pennsylvania, USA, July 1987, pp. 122-126.

[2] Amman, P.E., and J.C. Knight, "Data Diversity: an Approach to Software Fault Tolerance", IEEE TC, Vol. 37, No. 4, April 1988, pp. 418-425.

[3] Anderson, T., "A structured decision mechanism for diverse software", Proc. 5th Symposium on Reliability in Distributed Software and Database Systems, Jan. 1986, Los Angeles, California, pp. 125-129.

[4] Arlat, J., K. Kanoun, J.C. Laprie "Dependability Modelling and Evaluation of Software Fault Tolerant Systems", IEEE Transactions on Computers, Vol.39 N.4 April 1990, pp.504-513.

Asthana, A. "A Fault Tolerant Architecture with Dynamically Reconfigurable Redundant Partitions", Report TM-851109-01, AT&T Bell Laboratories, November 1985.

[5] Avizienis, A., P. Gunningberg, J.P.J Kelly, R.T. Lyu, L. Strigini, P.J. Traverse, K.S. Tso, U. Voges, "Software by Design Diversity; Dedix: a Tool for Experiment", Proc. IFAC Workshop SAFECOMP '85, Como, Italy, October 1985, pp. 173-178.

[6] Avizienis, J.P.J. Kelly "Fault-Tolerance by Design Diversity: Concepts and Experiments", IEEE Computer, Vol.17, N.8, August 1984, pp. 67-80.

[7] Bishop, P.G., F.D. Pullen, "Error Masking: a Source of Failure Dependency in Multi-Version Programs", Preprints of the 1st International Working Conference on Dependable Computing in Critical Applications, Santa Barbara, California, August 1989, pp. 25-32.

[8] Broen, R.B., "New voters for redundant systems", Journal of Dynamic Systems, Measurement, and Control, March 1985.

Caglayan, A.K., P.R. Lorczak, D.E. Echkardt, "An experimental investigation of software diversity in a fault tolerant avionics application", 18h International Symposium on Fault Tolerant Computing, June 1988, Tokyo.

Chen, L., and A. Avizienis "N-Version programming: a fault-tolerance approach to reliability of software operation", 8th Annual International Symposium on Fault Tolerant Computing, June 1978, Toulouse

Echtle, K., "Distance Agreement Protocols", Proc. 19th FTCS, Chicago, Illinois, June 1989, pp. 191-198.

[9] Echtle, K., "Fault diagnosis by combination of absolute and relative tests", presented at EWDC-1, 1st European Workshop on Dependable Computing, March 1989, Toulouse.

[10] Eckhardt, D.E., L.D. Lee, "A Theoretical Basis for the Analysis of Multiversion software subject to coincident errors", IEEE TSE, Vol. SE-11, No 12, December 1985, pp. 1511-1517.

[11] Kanekawa, K., H. Maejima, H. Kato, H. Ihara "Dependable onboard computer systems with a new method -Stepwise negotiating voting", 19th International Symposium on Fault Tolerant Computing, June 1989, Chicago, pp. 13-19.

[12] Lala, J.H., and L.S. Alger, "Hardware and software fault-tolerance: a unified architectural approach", Proc. 18-th International Symposium on Fault Tolerant Computing, June 1988, Tokyo, pp. 240-245.

[13] Littlewood, B., and D.R. Miller, "A conceptual model of multi-version software", Proc. 17th FTCS, Pittsburgh, July 1987, pp. 150-155.

[14] Littlewood, B., and D.R. Miller, "A conceptual model of the effect of diverse methodologies on coincident failures in multi-version software", Proc. 3rd International Fault-Tolerant Computer Systems Conference, Bremerhaven, Germany, September 1987.

[15] Lorczak, P. R., Alper K. Caglayan, Dave E. Eckhardt, "A Theoretical Investigation of Generalized Voters for Redundant Systems", Proc. 19th FTCS, Chicago, Illinois, June 1989, pp. 444-451.

[16] Makam, S. V., "Design Study of Fault-Tolerant Computer to Execute N-Version Software", Ph. D. Dissertation, UCLA Computer Science Department, 1982.

Traverse, A. Avizienis "Decision algorithms for the DEDIX system", Technical Report, UCLA 1985.

[17] Powell, D., G. Bonn, D. Seaton, P. Verissimo, F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", Proc. 18-th FTCS, Tokyo, June 1988, pp. 246-251.

[18] Scott, R. K., J. W. Gault, D. F. McAllister, J. Wiggs "Fault-Tolerant Software Reliability Modeling", IEEE TSE, Vol. SE-13, No. 5, May 1987, pp. 582-592.

[19] Voges, U. (ed.), "Software diversity in computerized control systems", Dependable Computing and Fault-Tolerance series, Vol. 2, Springer-Verlag 1988.