To appear in IEEE Trans on Computers Jan 1981

# An Approach for Error Detection and Error Correction in Distributed Systems Computing Numerical Functions *

by M. Karpovsky, Member IEEE

Mailing address:

M. Karpovsky, Computer Science Department

School of Advanced Technology

State University of New York at Binghamton

Binghamton, New York 13901

* This work was supported in part by the National Science Foundation under Grant Number MCS-8008330

## ABSTRACT

We consider methods of error detection and/or error correction in software and hardware of a distributed system computing values of numerical functions. These methods are based on software and hardware redundancy for the computation of additional check functions. The check functions are easily derived for any given multiplicity of errors. The redundancy does not depend on the number of processors in the original system and depends only on the multiplicity of errors. We describe methods for the construction of optimal checks, required software and hardware redundancy and implementation of the corresponding error detecting/correcting procedures by a distributed system.

## I. Introduction

Since technology moves toward very large-scale integration and distributed computer systems find more and more applications, the problem of testing and error correction for software and hardware of these systems begins to be extremely important.

In this paper we shall consider error detection and error correction for the case of distributed computer systems with errors of an arbitrary but a given multiplicity both in software and in hardware.

We shall describe one approach to the solution of these problems for distributed systems computing numerical functions (functions whose values are integer numbers, vectors or matrices), where all processors of a system have the same input data (single input data string). The typical example of these computations are polynomial computations when the processor $P_i$ computes values of a polynomial

$$f_i(x_1,\ldots,x_m) = \sum_{j_1=0}^{d_1} \ldots \sum_{j_m=0}^{d_m} \alpha(j_1,\ldots,j_m)\, x_1^{j_1} \ldots x_m^{j_m} \quad (i=1,\ldots,k).$$

Our approach to the problems of error detection and error correction in distributed systems computing numerical functions will be based on software and hardware redundancy for the computation of some additional functions connected with the functions computed by the original system using linear checking equations. The verification of whether this is indeed the case and analysis of the results of the checks (syndromes) constitute the error detecting and error correcting procedures. We note that these procedures may be used in any multiprogramming environment. The implementation of these procedures for many practical cases may be carried out by adding a few more microprocessors to an existing distributed system.

We shall describe methods for the construction of optimal checks for the detection/correction of errors of an arbitrary multiplicity both in software and in hardware (for the construction of these checks we shall use the techniques of error-correcting codes), estimate the required software and hardware redundancy and discuss some of the implementation issues for these error detecting/correcting techniques. Communication and synchronization aspects of implementation will not be discussed in this paper. Other techniques based on error-correcting codes for error detection in numerical computations for the case of non-distributed computer systems were considered in [1], [2].

Error detecting and error correcting methods proposed in this paper may be used for the testing of manufacturing acceptance of distributed software, for the maintenance testing of distributed hardware and for error correction for permanent and intermittent faults in distributed systems. These methods are also intended to reduce the amount of hardware needed to incorporate fault-tolerance into a system.

## II. Correction of Single Errors and Detection of Double Errors in Distributed Systems

2.1. Consider the following example:

Example 1. Suppose we have k=2 processors computing two numerical functions $f_1$ and $f_2$. We want to correct any single error in software or in hardware of this system. (All single errors are supposed to be independent.) We may use the triplication technique (TMR). In this case we need four additional processors and four additional programs. Let us describe another technique which requires only three additional processors and three additional programs computing the functions $f_3, f_4, f_5$. These functions $f_3, f_4 f_5$ may, for example, satisfy the conditions

$$\begin{aligned} a_{11}f_1+a_{12}f_2+a_{13}f_3 &= 0\ , \\ a_{21}f_1+a_{22}f_2+a_{24}f_4 &= 0\ , \\ a_{31}f_1 \qquad +a_{35}f_5 &= 0\ , \end{aligned} \tag{1}$$

where $a_{ij} \neq 0$ are arbitrary constants.

In equations (1) and later in this paper we suppose that for any function $f_i$ calculated by the $i^{th}$ processor and any x we have $0 \le f_i(x) < Q$, where Q is a large prime number, and all the calculations are carried out modulo Q. For example, if processors produce m-digit binary outputs and $0 \le f_i(x) < 2^m - 1$, where $2^m - 1$ is a prime, we can choose $Q = 2^m - 1$. The results we shall describe in this paper will not depend on the choice of Q.

The error correction procedure will be based on verification of (1). As a result of a single error function $f_p$ ($p \in \{1,\ldots 5\}$) is replaced by function $f_p + e_p$, where $e_p \neq 0$ is an error function. The error vector in this case will be $e = (e_1,\ldots,e_5) = (0,\ldots,0,e_p,0,\ldots,0)$ $(e_p \neq 0)$. We can define the result of an error (syndrome) $S = (S_1,S_2,S_3)$ as follows $S_i = \sum_{j=1}^{5} a_{ij}(f_j + e_j) = a_{ip}e_p$ $(i=1,2,3)$. For example, if there is an error $e_1$ in $f_1$, then the syndrome instead of (0,0,0) will be $(a_{11}e_1, a_{21}e_1, a_{31}e_1)$; for an error $e_2$ in $f_2$ it will be $(a_{12}e_2, a_{22}e_2, 0)$, etc. An error correcting procedure (computation of $e_i$ for $i=1,\ldots,5$) may be implemented by the analysis of these syndromes, since for any two single errors the syndromes are different and not equal to (0,0,0). If a syndrome vector does not correspond with the syndrome vector for any single error and is not equal to (0,0,0), we suppose that a double error exits in our system. For example, suppose that $S_1 \neq 0, S_2 \neq 0, S_3 = 0$. If $S_2\, a_{22}^{-1} = S_1 a_{12}^{-1}$, then we have the error $e_2 = S_2\, a_{22}^{-1}$ in $f_2$; if $S_2\, a_{22}^{-1} \neq S_1 a_{12}^{-1}$, then we conclude that a double error exists.

Since the constants $a_{ij}$ in (1) may be chosen arbitrarily, they may be used to decrease the complexity of programs computing the "check" functions $f_3, f_4, f_5$. We shall see later in this section that for many important distributed computations, these $a_{ij}$ may be chosen so that the check functions become constants. In this case no additional software or hardware is required (except for software for the error correcting procedure).

Comparing the checks (1) with TMR again, we note that TMR requires four additional processors, and corrects any single error.

For the checks (1) we need three additional processors, all single errors are corrected and double errors are detected.

2.2. Let us have k processors which compute numerical functions $f_1,\ldots,f_k$. Errors in the computation processes for $f_1,\ldots,f_k$ are independent, and we want to correct any single software or hardware error. For error correction we shall use r additional processors which will compute check functions $f_{k+1},\ldots,f_{k+r}$, satisfying r linear equations:

$$\sum_{j=1}^{k+r} C_{ij}f_j = 0. \quad (i=1\ldots, r), \text{ where } C_{ij} \text{ are constants.} \tag{2}$$

(Equation (1) is an example of these equations for k=2,r=3).

As a result of a single error our system $(f_1,\ldots,f_{k+r})$ will be replace by a system $(f_1+e_1,\ldots,f_{k+r}+e_{k+r})$, where only one of the components of an error vector $(e_1,\ldots,e_{k+r})$ is not equal to zero.

For the error $e = (0,\ldots,0,e_p,0,\ldots,0)$ $(e_p\neq 0)$ the result of the check (2) (which we shall call the syndrome of the error e) will be

$$S_i = \sum_{j=1}^{k+r} C_{ij}(f_j+e_j) = C_{ip}e_p \quad (i=1,\ldots, r). \tag{3}$$

The necessary and sufficient condition for the correction of single errors by the check system (2) is that any two errors $e^{(1)}$ and $e^{(2)}$ $(e^{(1)}\neq e^{(2)})$ will have different syndromes. If $e^{(1)} = (0,\ldots,0,e^{(1)}_p,0,\ldots,0)$ and $e^{(2)} = (0,\ldots,0,e^{(2)}_t,0,\ldots,0)$, then we have from (3)

$$(C_{1,p}e^{(1)}_p, C_{2,p}e^{(1)}_p,\ldots,C_{r,p}\ e^{(1)}_p) \neq (C_{1,t}e^{(2)}_t, C_{2,t}e^{(2)}_t,\ldots,C_{r,t}\ e^{(2)}_t). \tag{4}$$

It follows from (4) that necessary and sufficient conditions for the correction of single errors is that in a matrix $C=(C_{ij})$ each two columns $C_p$ and $C_t$ satisfy the condition:

$C_p \neq bC_t$, where b is an arbitrary constant. (5)

In particular, we have $C_p \neq 0$ for every $p^{\underline{th}}$ column of $(C_{ij})$.

Let $H = (h_{ij})$ be a check matrix with the dimensions $r \times (k+r)$ for a binary single-error-correcting code [3]. It is well known that in this case $h_{ij} \in \{0,1\}$ and all columns of H are different and not equal to zero. Hence, if we put

$$C_{ij} = a_{ij} h_{ij} \tag{6}$$

where $a_{ij} \neq 0$ are arbitrary constants, then the constructed matrix $C = (C_{ij})$ will satisfy the condition (5), and we may correct single errors using this matrix. For example, equations (1) were constructed by this method with the check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} . \tag{7}$$

Thus, for the construction of checking equations (2) we need only to choose a binary $r \times (k+r)$ matrix $(h_{ij})$ with different columns which are not equal to zero and then use (6). In this case r satisfies the condition $\lceil k+r+1 \leq 2^r \rceil$. (8)

Constants $a_{ij}$ in (6) may be chosen arbitrarily (except that $a_{ij} \neq 0$), and we have $N=N_2=\sum_{i,j} h_{ij}$ free variables $C_{ij}$ in (2); by the appropriate choice of the values of these $N_2$ variables the complexity of computation for check functions $f_{k+1},\ldots,f_{k+r}$ may be decreased. We shall see, for example, in Section 2.4 that, if $f_1,\ldots,f_k$ are polynomials, then for many practical cases with the appropriate choice of $C_{ij}$ in (2) the check functions $f_{k+1},\ldots,f_{k+r}$ will be constants, and we do not need any additional hardware and software for the correction of single errors.

Since the binary check matrix $H=(h_{ij})$ has dimensions $r \times (k+r)$, we have the following exact upper bound for the number $N_2$ of free variables

$$N_2 \leq r2^{r-1}, \tag{9}$$

and we have equality in (9) iff $k+r+1 = 2^r$. Thus, if we want to increase the number $N_2$ of free variables, we have to choose the check matrix H of a single-

error-correcting code with the maximal number of ones.

2.3. Let us describe now another method for the construction of checks (2) which will reduce the number r of redundant functions $f_{k+1},\ldots,f_{k+r}$, but will also reduce the number N of free variables. This method will be based on non-binary single-error-correcting codes.

Let us have a rx(k+r) q-ary matrix $H = (h_{ij})$, $h_{ij}\varepsilon\{0,\ldots,q-1\}(q\geq 2)$ (q doesn't have to be a prime number). Suppose H does not contain zero columns, and for every two columns $h_p$ and $h_t$ of H we have $h_p \neq bh_t$ for all $b = 0,1,\ldots,q-1$ (here all the multiplications are carried out by mod q). If q is a prime number, then H is a check matrix for a q-ary single-error-correcting code.

Let us partition the set of all columns of H into two sets $H_1$ and $H_2$ ($|H_1|+|H_2| = k+r$, where $|H_j|$ is the number of columns in $H_j$,). Column $h_p$ belongs to the set $H_1$ iff $h_{ip}\varepsilon\{0,1\}$ for all $i = 1,\ldots,r$; column $h_t$ of H belongs to $H_2$ if it does not belong to $H_1$. Matrix $C = (C_{ij})$ now may be defined in the following way:

$$C_{ij} = \begin{cases} d_j h_{ij} & \text{, if } h_j\varepsilon H_2 \\ \text{for all } i=1,\ldots,r & , \\ a_{ij}\, h_{ij} & \text{,if } h_j\varepsilon H_1 \end{cases} \qquad (10)$$

where $d_j \neq 0$, $a_{ij} \neq 0$ arbitrary constants and $\frac{a_{it}}{a_{jt}} \notin\{2,\ldots,q-1\}$ for $h_t\varepsilon H_1$. Then, for the matrix $C = (C_{ij})$ we have $C_p \neq bC_t$ for all possible p,t,b.

Thus, any matrix $(C_{ij})$ generated by (10) from the q-ary check matrix $(h_{ij})$ satisfies (6) and may be used for the correction of all single errors distributed system.

Example 2. Let us have a distributed system of k=2 processors computing the functions $f_1,f_2$. Choose q=3 and the check matrix H of a single-error-correcting ternary code $H = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{pmatrix}$ (r=2). Then, $H_1 = \left\{\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}\right\}$, $H_2 = \left\{\begin{pmatrix}1\\2\end{pmatrix}\right\}$

and by (10) and (2) we have the following checks

$$a_{11}f_1+d_2f_2+a_{13}f_3 = 0 \quad ,$$

$$a_{21}f_1+2d_2f_2+a_{24}f_4 = 0 \qquad (a_{1j}\neq 0 \text{ and } \frac{a_{21}}{a_{11}} \neq 2) \ . \tag{11}$$

Comparing these checks with (1), we may see that the transition from the binary error-correcting code to the ternary code reduces the number r of additional functions from 3 to 2 and also reduces the number N of free variables from 8 to 5. This illustrates the tradeoff between a number of processors k+r and the software complexity of the check functions $f_{k+1},\ldots,f_{k+r}$. This tradeoff will be discussed later in this section.

2.4. Let us consider now the relationship between the number of additional processors, r, and the number $N=N_q$ of free variables for the correction of single errors. For a q-ary rx(k+r) check matrix $(h_{ij})$ we have [3]

$$1+(k+r)(q-1)\leq q^r \quad , \tag{12}$$

and the minimal r satisfying this condition decreases with increasing q.

For the number $N_q$ of free variables we have from (10)

$$N_q = \sum_{\{i,j \mid h_j \in H_1\}} h_{ij}+k+r-|H_1| \tag{13}$$

Note, that for the minimal r satisfying (12) we have $k\geq 2^r - r-1$,

$$|H_1| = 2^r-1, \qquad \sum_{\{i,j \mid h_j \in H_1\}} h_{ij} = r\ 2^{r-1} \qquad \text{and} \tag{14}$$

$$N_q = k+2^{r-1}(r-2)+r+1 \ . \tag{15}$$

The minimal number of additional processors for the correction of single errors number N of free variables are presented in Tables I - V for k = 10,20,30,40,50.

It follows from (12) that for any $q > k$ we have r=2. For r=2 the minimal q is k+1, and we may choose $H = \begin{pmatrix} 1 & 1\ldots1 & 1 & 0 \\ 1 & 2\ldots k & 0 & 1 \end{pmatrix}$. Then we have by (10)

the following checks

$$\sum_{i=1}^{k+1} a_{1\,i} f_i = 0 \quad ,$$

$$a_{2\,1} f_1 + \sum_{i=2}^{k+1} i a_{1\,i} f_i + a_{2\,k+2} f_{k+2} = 0 \quad , \tag{16}$$

where $a_{1\,i}$, $a_{2\,1}$, $a_{2\,k+2} \neq 0$ are arbitrary constants and $\frac{a_{2\,1}}{a_{1\,1}} \notin \{2,3,\ldots,k\}$.

The number of free variables $a_{ij}$ for this case is $N_{k+1} = k+3$.

Tables I - V

Methods described in this section may be used for the correction of single errors not only in the software and hardware computing the original system of functions $f_1,\ldots,f_k$ but also in the software and hardware computing the check functions $f_{k+1},\ldots,f_{k+r}$ (but not in the software and hardware implementing the error correcting procedures). The correcting procedures do not depend on the original system of functions $f_1,\ldots,f_k$.

We note also that these methods may be used for the detection of all double errors. In this case we need only to verify that the syndrome $S_i = \sum_j C_{ij}(f_j+e_j) = \sum_j C_{ij}e_j$ is not equal to zero for at least one i.

2.5. We shall consider in this section the special case when the distributed system of k processors computes k polynomials $f_1(x),\ldots,f_k(x)$, where

$$f_i(x) = \sum_{j=1}^{d} \alpha_{ij} x^j \qquad (i=1,\ldots,k;\ \alpha_{id} \neq 0) \text{ and } \alpha_{ij} \text{ are some constants.} \tag{17}$$

This case is very important from a practical point of view since any analytical function may be approximated by a polynomial.

Applying the methods described in Sections 1.1-2.3 to this special case, we have

$$f_{k+i}(x) = \sum_{j=1}^{k} C_{ij} f_j(x) \quad (i=1,\ldots,r), \tag{18}$$

where $C_{ij}$ is defined by (6) or (10). If for the given i we have N(i) free variables $C_{ij}$, then, by an appropriate choice of these variables, the degree of the polynomial $f_{k+i}(x)$ may be always decreased to $(d-N(i)+1)^+$ $\left(x^+ = \begin{cases} x, & \text{for } x \geq 0 \\ 0, & \text{for } x<0 \end{cases}\right)$.

For example, if k=2, d=3 and $f_3(x) = a_{11}f_1(x)+a_{12}f_2(x)$ (See Example 1).

$$f_4(x) = a_{21}f_1(x)+a_{22}f_2(x)$$

$$f_5(x) = a_{31}f_1(x) \qquad (a_{ij}\neq 0),$$

then N(1) = N(2) = 2. If, for example, we choose $a_{11}=a_{21}=1, a_{12}=a_{22}= -\alpha_{13}(\alpha_{23})^{-1}$, then $f_3(x), f_4(x)$ are polynomials of degree 2. If $N(i)\geq d+1$ for every $i=1,\ldots,r$, then the free variables may be chosen in such a way, that the degrees of all $f_{k+i}(x)$ $(i=1,\ldots,r)$ will be equal to 0, and no additional processors will be required for the correction of single errors. For the estimation of N(i) we may use the formula $N(i) = [(N-r)r^{-1}]$ for all $i=1,\ldots,r$, where [a] is the greatest integer $\leq a$. The maximal degree of polynomials, d, such that no additional processors are needed for the correction of single errors is given also in Tables I - V.

## III. Detection and Correction of Multiple Errors in Distributed Systems

3.1. Consider a system of k processors computing k functions $f_1,\ldots,f_k$. We shall add to this system r additional processors computing $f_{k+1},\ldots,f_{k+r}$ such that

$$\sum_{j=1}^{k+r} C_{ij} f_j = 0 \quad (i = 1,\ldots,r). \tag{19}$$

As a result of an error $e = (e_1,\ldots,e_{k+r})$ in the new redundant system, $(f_1,\ldots,f_{k+r})$ is replaced by $(f_1+e_1,\ldots,f_{k+r}+e_{k+r})$.

By a multiplicity of an error we mean a number of nonzero components in a vector e. All errors of a multiplicity at most L are detected by (19), if for any error of this type, the syndrome $S = (S_1,\ldots,S_r)$ of this error is not equal to $(0,\ldots,0)$, where

$$S_i = \sum_{j=1}^{k+r} C_{ij}(f_j+e_j) = \sum_{j=1}^{k+r} C_{ij}e_j \quad (i = 1,\ldots,r). \tag{20}$$

All errors of a multiplicity at most L are corrected by (19) if any two different errors have different syndromes. It is easy to show that (19) corrects all errors of a multiplicity at most L iff it detects all errors of a multiplicity at most 2L. If an error, e, has a multiplicity 2L, then there exists $1\le j_1<j_2<\ldots<j_{2L}\le k+r$, such that $e_{j_1}\neq 0,\ldots,e_{j_{2L}}\neq 0$ and $e_j=0$ iff $j\notin\{j_1,\ldots,j_{2L}\}$. The syndrome of this error will be

$$S_i = C_{ij_1}e_{j_1}+\ldots+C_{ij_{2L}}e_{j_{2L}} \quad (i = 1,\ldots,r), \tag{21}$$

and this error is detected by (19) iff $S \neq (0,\ldots,0)$ for every $j_1,\ldots,j_{2L}$ and every $e_{j_1},\ldots,e_{j_{2L}}$. Thus, equations (20), (21) show that the system (19) detects all errors of a multiplicity at most 2L iff each (2Lx2L) submatrix of $(C_{ij})$ is nonsingular. Methods for the construction of matrices $(C_{ij})$ satisfying this condition will be discussed in Sections 3.2 and 3.3. We note also that the minimal number r of processors for the detection of all errors of a multiplicity at most 2L (or for the correction of all errors of a multiplicity at most L) is equal to $r_{min} = 2L$.

3.2. Let $d_1,\ldots,d_{k+r}$ be arbitrary constants ($d_j\neq 0$ for all j), r=2L and

$$C_{ij} = j^{i-1}d_j \quad (i = 1,\ldots, r). \tag{22}$$

Then any (rxr) submatrix F of $(C_{ij})$ may be represented in a form:

$$F = \begin{pmatrix} C_{1j_1} & C_{1j_2} & \cdots & C_{1j_r} \\ C_{2j_1} & C_{2j_2} & \cdots & C_{2j_r} \\ \cdot & \cdot & \cdot & \\ C_{rj_1} & C_{rj_2} & \cdots & C_{rj_r} \end{pmatrix} = \begin{pmatrix} d_{j_1} & d_{j_2} & \cdots & d_{j_r} \\ j_1 d_{j_1} & j_2 d_{j_2} & \cdots & j_r d_{jr} \\ \cdot & \cdot & \cdot & \\ j_1^{r-1} d_{j_1} & j_2^{r-1} d_{j_2} & \cdots & j_r^{r-1} d_{j_r} \end{pmatrix} \qquad (23)$$

where $1 \le j_1 < j_2 \ldots < j_r \le k+r$. Hence, for det F we have

$$\det F = d_{j_1} \ldots d_{j_r} \Delta \neq 0 \qquad (24)$$

where $\Delta$ is the Vandermonde's determinant [4].

It follows now from (24) that the matrix $(C_{ij})$ constructed by (22) detects all errors of a multiplicity at most r=2L or corrects all errors of a multiplicity at most L.

Example 3. Suppose that the given system of k processors computes the Krawtchouk polynomials [5,p. 151]

$$P_i(x) = \sum_{j=0}^{i} (-1)^j \binom{x}{j} \binom{k-x}{i-j} \quad (i=1,\ldots,k; x \in \{0,1,\ldots,k\}),$$

and we want to detect all errors of a multiplicity at most three. If we choose $d_i = 1$ $(i=1,\ldots,k)$ and $d_{k+1} = d_{k+2} = d_{k+3} = -1$, then by (19), (22), the check functions $f_{k+1}$, $f_{k+2}$, $f_{k+3}$ for the Krawtchouk polynomials [5, p. 153] are:

$$f_{k+1}(x) = \sum_{i=1}^{k} P_i(x) = 2^k \delta_{x,0} - 1,$$

$$f_{k+2}(x) = \sum_{i=1}^{k} i\, P_i(x) = 2^{k-1} (k\, \delta_{x,0} - \delta_{x,1}),$$

$$f_{k+3}(x) = \sum_{i=1}^{k} i^2 P_i(x) = 2^{k-2} (k(k+1)\, \delta_{x,0} - 2k\delta_{x,1} + 2\delta_{x,2})$$

(where $x \in \{0,1,\ldots,k\}$ $\delta_{x,t} = 1$ if $x=t$, $\delta_{x,t} = 0$ if $x \neq t$ is the Kronecker symbol), and an additional software for the computation of $f_{k+1}(x)$, $f_{k+2}(x)$ and $f_{k+3}(x)$ is very simple.

3.3. The check system (19) constructed by (22) is a system with the minimal number of checks, but this system is not unique. We shall describe now another solution of the same problem which may be more efficient in the case of faults of a high multiplicity.

Let $0 \le f_i(x) < Q$ for all i=1...,k and x; $\xi$ be a primitive element of GF(Q), r=2L and

$$C_{ij} = \begin{cases} \xi^{(i-1)(j-1)} d_j, \ j=1,\ldots,k ; \\ \qquad\qquad (i = 1,\ldots,r) \\ \delta_{ij} d_j \ , \ j = k+1,\ldots k+r \ ; \end{cases} \quad , \tag{25}$$

where $\delta_{ij}$ is the Kronecker delta and $d_1,\ldots,d_{k+r}$ are arbitrary constants not equal to 0. Then any (rxr) submatrix of $(C_{ij})$ is nonsingular, and the check system (19) with $C_{ij}$ defined by (25) detects all errors of a multiplicity at most 2L (or corrects all errors of a multiplicity at most L).

We note that for the system (25) $(-d_{k+1}f_{k+1},\ldots,-d_{k+r}f_{k+r})$ may be considered as the first r coefficients of the discrete Fourier Transform (see, e.g., [6]. of $(d_1 f,\ldots ,d_k f_k)$.

The number of free variables $d_j$ for both solutions (22) and (25) is equal to N=k+r. If $f_i(x)$ (i=1,...,k) are polynomials of a degree at most d, then, applying the results from Section 2.5, it is easy to show that for $d \le \left[\frac{k}{2L}\right]$ no additional processors are needed for the correction of errors of a multiplicity at most L.

IV. Implementation of Error Detecting and Error Correcting Procedures

4.1. For error detection we need to compute the syndrome $S = (S_1,\ldots,S_L)$ (see (20), r=L) and check whether S = (0,...,0). If $P_1$ is a processor, computing $f_i$ (i = 1,...,k+L) then for these computations we need only communications $P_i \to P_{i+1}$ between processors. The required time $T_d$ for the detection of at most L errors may be estimated as

$$T_d = A_d L(k+r) = A_d(Lk+L^2) \quad , \tag{26}$$

where $A_d$ is a constant characterizing the processor's speed and the communication's speed for the given distributed system.

Let us describe now the error detecting capability of the system (19) with r additional check processors, where $C_{ij}$ is constructed by (22) or (25).

We denote the relative frequency of errors of a multiplicity L which cannot be detected by $\eta(r,L)$. It was proven in the previous section that $\eta(r,L) = 0$ for $1 \le L \le r$. Note also, that the check system (19) detects a high percentage of errors with a multiplicity $L>r$. We assume, that for every $i \in \{1,\ldots,k+r\}$ values of $f_i$ are represented by binary vectors with m components, then an error $e = (e_1,\ldots,e_{k+r})$ with $e_j \neq 0$ iff $j \in \{j_1,\ldots,j_L\}(L>r)$ cannot be detected by (19) iff

$$S_i = \sum_{j=1}^{k+r} C_{ij}e_j = C_{ij_1}e_{j_1}+\ldots+C_{ij_L}e_{j_L} = 0 \qquad (i=1,\ldots,r). \tag{27}$$

Since for any fixation of $e_{j_{r+1}},\ldots,e_{j_L}$ (27) has at most one solution for $e_{j_1},\ldots,e_{j_r}$, we finally have:

$$\eta(r,L) \le \begin{cases} 0, & 1 \le L \le r \\ (2^m-1)^{-r}, & \text{for } L>r. \end{cases} \tag{28}$$

For unidirectional errors such that $e_i \ge 0$ $(i = 1,\ldots,k+r)$ ( or $e_i \le 0$ for all $i = 1,\ldots,k+r$), we have $\eta(r,L) = 0$ for all $r,L>0$. Thus, for many practical cases one or at most two additional processors are usually enough to provide the required error detecting capability.

4.2. We shall consider in this section the problem of implementing of the error correcting procedures for a distributed system. The error correcting procedure is divided into two steps, as is usually done in coding theory: first, we compute results of the checks (19) (syndromes) and secondly, we correct errors (compute $f_i+e_i$ for all $i = 1,\ldots,k+r$) using the computed syndromes.

For the syndrome computation, (20) and (22) or (25) may be used. For the case $L \simeq (1/2)k$ or $r \simeq k$ and $C_{ij}$, which is defined by (25), the very efficient

algorithm of Fast Fourier Transform [6] may be used. This decreases the total number of additions and multiplications, but increases the number of communication links between processors.

For the correction of an error $e = (e_1,\ldots,e_{k+r})$, such that $e_j \neq 0$ iff $j \in \{j_1,\ldots,j_L\}$ by (19, (22) and the computed syndrome $S = (S_1,\ldots,S_r)$, we need to solve the system

$$j_1^{i-1} d_{j_1} e_{j_1} + \ldots + j_L^{i-1} d_{j_L} e_{j_L} = S_i \quad (i=1,\ldots,r). \tag{29}$$

where $r=2L$, and $j_1,\ldots,j_L$, $e_{j_1},\ldots,e_{j_L}$ are unknowns. It follows from the results of Section III, that (29) has a unique solution for every $d_{j_1},\ldots,d_{j_L} \neq 0$ and every $S_1,\ldots,S_{2L}$, if a multiplicity of the error $e$ is no more than L. This solution for the correction of single errors (L=1, r=2) may be described as follows:

1. If $S_1 = S_2 = 0$, then there are no errors.
2. If $S_1 \neq 0$, $S_2 \neq 0$, compute $j = S_2 (S_1)^{-1}$; if $j \in \{1,\ldots,k+2\}$ then,

$$e_i = \begin{cases} 0 & ,i \neq j \\ S_1 & ,i=j \end{cases}$$

; if $j \notin \{1,\ldots,k+2\}$, then there is a double error.

3. If $S_1 = 0$, $S_2 \neq 0$ or $S_1 \neq 0$, $S_2 = 0$, then there is a double error.

4.3. Let us consider now the correction of double errors by the computed syndrome. For this case r=4, and from (29) we have the following system of four equations with four unknowns $j_1, j_2, e_{j_1} e_{j_2}$:

$$j_1^{i-1} d_{j_1} e_{j_1} + j_2^{i-1} d_{j_2} e_{j_2} = S_i \quad (i=1,2,3,4). \tag{30}$$

If $S_1 = \ldots = S_4 = 0$, then $e_{j_1} = e_{j_2} = 0$. If $S_2 (S_1)^{-1} = S_3 (S_2)^{-1} = S_4 (S_3)^{-1} \in \{1,\ldots,k+4\}$, then the multiplicity of an error is one, and $j_1 = S_2 (S_1)^{-1}$, $e_{j_1} = S_1 \quad d_{j_1}^{-1}$. If the two previous conditions are not satisfied, then the solution of (30) may be found in the following way.

Denote

$$v = \frac{(S_3)^2 - S_2 \cdot S_4}{(S_2)^2 - S_1 \cdot S_3}, \quad w = \frac{S_2 \cdot S_3 - S_1 \cdot S_4}{(S_2)^2 - S_1 \cdot S_3}. \tag{31}$$

Then, it is easy to check that

$$j_1 = -\frac{w}{2} + \sqrt{\frac{w^2}{4} - v}, \quad j_2 = -\frac{w}{2} - \sqrt{\frac{w^2}{4} - v}, \tag{32}$$

$$e_{j_1} = (j_2 S_1 - S_2)\, d_{j_1}^{-1} (j_2 - j_1)^{-1}, \quad e_{j_2} = (j_1 S_1 - S_2)\, d_{j_2}^{-1} (j_1 - j_2)^{-1}.$$

4.4. For a multiplicity L of errors more than two analytical solutions for the system (29) are very cumbersome. We note that the checks (19), (22) or (25) are very similar to the checking equations for Reed-Solomon error-correcting codes [3], and an analogue of the Berlekamp-Massey decoding procedure [3] can be used for the solution of (29). This procedure however is again rather complicated for $L \geq 3$. For many practical applications k is not too big; hence, to find the solution of (29) we may fix $1 \leq j_1 < \ldots < j_L \leq k+2L$ arbitrarily, and then try to solve (29) for r=2L with respect to $e_{j_1}, \ldots, e_{j_L}$. If there is no solution, then we have to change $j_1, \ldots, j_L$, etc.

If $P_i$ is a processor computing $f_i$ $(i = 1, \ldots, k+2L)$, then for the implementation of the error correcting procedures described in Section IV we need only communications $P_i \rightarrow P_{i+1}$.

As a concluding remark we note again that methods described in this paper may be applied to simultaneous error detection/correction both in a software and a hardware of a distributed system, and very small hardware and software redundancy for most practical cases is needed.

The main limitation of these methods is that they are efficient mainly for numerical computations, when the output values computed by the distributed system are integers, and all the processors within the distributed system have the same input data.

REFERENCES

1. Karpovsky, M., "Error Detection for Polynomial Computations", IEE J. on Computers and Digital Techniques, Vol. 2, N1, Feb. 1979, pp. 49-57.

2. Karpovsky, M., "Testing for Numerical Computations", IEE Proc., Vol. 127, N2, March 1980, pp. 69-76.

3. Peterson, W.W. and E.Y. Weldon, Jr., Error Correcting Codes, Cambridge, Ma., 1972.

4. Korn, G.A. and T.M. Korn, Mathematical Handbook for Scientists and Engineers, McGraw-Hill, 1968.

5. F.J. MacWilliams and N.J.A. Sloane, "The Theory of Error-Correcting Codes", Part 1, North Holland, 1977.

6. Brigham, E.O., The Fast Fourier Transform, Prentice-Hall, 1974.

k=10

| q | 2 | 3 | ... | 10 | 11 |
|---|---|---|---|---|---|
| r | 4 | 3 | ... | 3 | 2 |
| N | 31 | 18 | ... | 18 | 13 |
| d | 5 | 4 | ... | 4 | 5 |

TABLE I

k=20

| q | 2 | 3 | 4 | 5 | ... | 20 | 21 |
|---|---|---|---|---|---|---|---|
| r | 5 | 4 | 4 | 3 | ... | 3 | 2 |
| N | 73 | 41 | 41 | 28 | ... | 28 | 23 |
| d | 12 | 8 | 8 | 7 | ... | 7 | 9 |

TABLE II

k=30

| q | 2 | 3 | 4 | 5 | 6 | ... | 30 | 31 |
|---|---|---|---|---|---|---|---|---|
| r | 6 | 4 | 4 | 4 | 3 | ... | 3 | 2 |
| N | 138 | 51 | 51 | 51 | 38 | ... | 38 | 33 |
| d | 21 | 10 | 10 | 10 | 10 | ... | 10 | 14 |

TABLE III

k=40

| q | 2 | 3 | 4 | 5 | 6 | ... | 40 | 41 |
|---|---|---|---|---|---|---|---|---|
| r | 6 | 5 | 4 | 4 | 3 | ... | 3 | 2 |
| N | 174 | 94 | 61 | 61 | 48 | ... | 48 | 43 |
| d | 27 | 16 | 15 | 15 | 14 | ... | 14 | 19 |

TABLE IV

k=50

| q | 2 | 3 | 4 | 5 | 6 | 7 | ... | 50 | 51 |
|---|---|---|---|---|---|---|---|---|---|
| r | 6 | 5 | 4 | 4 | 4 | 3 | ... | 3 | 2 |
| N | 190 | 104 | 71 | 71 | 71 | 58 | ... | 58 | 53 |
| d | 29 | 18 | 15 | 15 | 15 | 17 | ... | 17 | 24 |

TABLE V

INDEX TERMS

Distributed Systems

Numerical Computations

Error Detection

Error Correction

Error Correcting Codes

MARK KARPOVSKY was born in Leningrad, USSR on October 27, 1940. He received the M.S. and Ph.D. degrees in computers from Leningrad Electrotechnical Institute, Leningrad, USSR in 1963 and 1967 respectively.

His research interests include fault-tolerant computing, distributed systems, error-correcting codes, reliable software and hardware. During the past 15 years, he has been active in computer research and has published about 40 papers in these areas.

Dr. Karpovsky is the co-author of Spectral Methods of Analysis and Synthesis of Digital Devices (Energy, USSR, 1973 in Russian); and the author of Finite Orthogonal Series in the Design of Digital Devices (John Wiley, 1976).

Dr. Karpovsky is currently an Associate Professor at the Department of Computer Science, State University of New York at Binghamton, Binghamton, NY.