Deterministic Voting in Distributed Systems Using Error-Correcting Codes

Lihao Xu and Jehoshua Bruck, *Senior Member*, *IEEE*

Abstract—Distributed voting is an important problem in reliable computing. In an *N* Modular Redundant (*NMR*) system, the N computational modules execute identical tasks and they need to periodically vote on their current states. In this paper, we propose a deterministic majority voting algorithm for *NMR* systems. Our voting algorithm uses error-correcting codes to drastically reduce the average case communication complexity. In particular, we show that the efficiency of our voting algorithm can be improved by choosing the parameters of the error-correcting code to match the probability of the computational faults. For example, consider an *NMR* system with 31 modules, each with a state of *m* bits, where each module has an independent computational error probability of 10^{-3}. In this *NMR* system, our algorithm can reduce the average case communication complexity to approximately 1.0825*m* compared with the communication complexity of 31*m* of the naive algorithm in which every module broadcasts its local result to all other modules. We have also implemented the voting algorithm over a network of workstations. The experimental performance results match well the theoretical predictions.

Index Terms— NMR system, communication complexity, majority voting, error-correcting codes, MDS code.

1 Introduction

Distributed voting is an important problem in the creation of fault-tolerant computing systems, e.g., it can be used to keep distributed data consistent, to provide mutual exclusion in distributed systems. In an *N* Modular Redundant (*NMR*) system, when the N computational modules execute identical tasks, they need to be synchronized periodically by voting on the current computation state (or result, and they will be used interchangeably hereafter), and then all modules set their current computation state to the majority one. If there is no majority result, then other computations are needed, e.g., all modules recompute from the previous result. This technique is also an essential tool for task-duplication-based checkpointing [12]. In distributed storage systems, voting can also be used to keep replicated data consistent.

Many aspects of voting algorithms have been studied, e.g., data approximation, reconfigurable voting, and dynamic modification of vote weights, metadata-based dynamic voting [3], [5], [9]. In this paper, we focus on the *communication complexity* of the voting problem. Several voting algorithms have been proposed to reduce the *communication complexity* [4], [7]. These algorithms are nondeterministic because they perform voting on signatures of local computation results. Recently, Noubir and Nussbaumer [8] proposed a majority voting scheme based on error-control codes: Each module first encodes its local result into a codeword of a designed error-detecting code and sends part of the codeword. By using the error-detecting code, discrepancies of the local results can be detected with some probability and, then, by a retransmission of full local results, a majority voting decision can be made. Though the scheme drastically reduces the *average case* communication complexity, it can still fail to detect some discrepancies of the local results and might reach a *false* voting result, i.e., the algorithm is still a probabilistic one. In addition, this scheme only uses the error-detecting capabilities of codes. As this paper will show, in general, using only *error-detecting codes* (*EDC*) does not help to reduce communication complexity of a deterministic voting algorithm. Though they have been used in many applications such as reliable distributed data replication [1], *error-correcting codes* (*ECC)* have not been applied to the voting problem.

For many applications [12], *deterministic* voting schemes are needed to provide more accurate voting results. In this paper, we propose a novel *deterministic* voting scheme that uses error-correcting / detecting codes. The voting scheme generalizes known simple deterministic voting algorithms. Our main contributions related to the voting scheme include:

1) using the correcting in addition to the detecting capability of codes (only the detection was used in known schemes) to drastically reduce the chances of retransmission of the whole local result of each node, thus the communication complexity of the voting,
2) a proof that our scheme provably reaches the same voting result as the naive voting algorithm in which every module broadcasts its local result to all other modules, and
3) the tuning of the scheme for optimal average case communication complexity by choosing the parameters of the error-correcting / detecting code, thus making the voting scheme adaptive to various application environments with different error rates.

The paper is organized as follows: In Section 2, we describe the majority voting problem in NMR systems. Our

• *The authors are with the Electrical Engineering Department, California Institute of Technology, Mail Code 136-93, Pasadena, CA 91125. E-mail: {lihao, bruck}@paradise.caltech.edu.*

Manuscript received 12 Jan. 1998; revised 30 June 1998.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 106146.

voting algorithm, together with its correctness proof, are described in Section 3. Section 4 analyzes both the worst case and the average case communication complexity of the algorithm. Section 5 presents experimental results of performances of the proposed voting algorithm, as well as two other simple voting algorithms for comparison. Section 6 concludes the paper.

2 The Problem Definition

In this section, we define the model of the NMR system and its communication complexity. Then, we address the voting problem in terms of the communication complexity.

2.1 NMR System Model

An NMR system consists of N computational modules which are connected via a communication medium. For a given computational task, each module executes a same set of instructions with independent computational error probability p. The communication medium could be a bus, a shared memory, a point-to-point network, or a broadcast network. Here, we consider the communication medium as a *reliable broadcast network*, i.e., each module can send its computation result to all other modules with only one error-free communication operation. The system evolution is considered to be synchronous, i.e., the voting process is round-based.

2.2 Communication Complexity

The *communication complexity* of a task in an NMR system is defined as the *total* number of bits that are sent through the communication medium in the whole execution procedure of the task. In a broadcast network, let m_{ij} be the number of the bits that the ith module sends at the jth round of the execution of a task, then the communication complexity of the task is $\sum_{i=1}^{N}\sum_{j=1}^{K} m_{ij}$, where N is the number of the modules in the system and K is the number of rounds needed to complete the task.

2.3 The Voting Problem

Now, consider the voting function in an NMR system. In an NMR system, in order to get a final result for a given task, after each module completes its own computation separately, it needs to be synchronized with other modules by voting on the result. Denote X_i as the local computational result of the ith module, the *majority function* is defined as follows:

$$Majority(X_1, \ldots, X_N) := \begin{cases} X_{i_1} & \text{if } \exists i_1, \ldots, i_{\lfloor \frac{N}{2} \rfloor + 1} : X_{i_1} = \cdots = X_{i_{\lfloor \frac{N}{2} \rfloor + 1}} \\ \phi & \text{otherwise,} \end{cases}$$

where in general, N is an odd natural number and ϕ is any predefined value different from all possible computing results.

EXAMPLE 1. If X_1 = 0000, X_2 = 0001, X_3 = 0100, X_4 = 0000, X_5 = 0000, then Majority(X_1, X_2, X_3, X_4, X_5) = 0000; if X_5 changes to 0010 and other X_is remain unchanged, then Majority(X_1, X_2, X_3, X_4, X_5) = ϕ.

The result of voting in an NMR system is that each module gets $Majority(X_1, \cdots, X_N)$ as its final result, where X_i (i = 1, $\cdots$, N) is the local computation result of the ith module.

One obvious algorithm for the voting problem is that, after each module computes the task, it broadcasts its own result to all the other modules. When a module receives all other modules' results, it simply performs the majority voting locally to get the result. The algorithm can be described as follows:

Algorithm 1 *Send-All Voting*

For Module P_i, $i \in [1 : N]$:
  Broadcast(X_i);
  Wait Until Receive all X_j, $j \in [1 : N] \setminus i$;
  $X := Majority(X_1, \cdots, X_N)$;
  Return(X);

This algorithm is simple: Each module only needs one communication (i.e., broadcast) operation but, apparently, its communication complexity is too high. If the result for the task has m bits, then the communication complexity of the algorithm is Nm bits. In most cases, the probability of a module having a computational error is rather low, namely, at most times all modules shall have the same result, thus, each module only needs to broadcast part of its result. If all the results are identical, then each module shall agree with that result. If not, then modules can use *Algorithm 1*. Namely, we can get another simple improved voting algorithm as follows:

Algorithm 2 *Simple Send-Part Voting*

For Module P_i, $i \in [1 : N]$:
  Partition the local result X_i into N symbols: $X_i[1 : N]$;
  Broadcast($X_i[i]$);
  Wait Until Receive all $X_j[j]$, $j \in [1 : N] \setminus i$;
  $X := X_1[1] * \cdots * X_N[N]$;
  $F_i := (X = X_i)$;
  Broadcast(F_i);
  If $Majority(F_1, \cdots, F_N) = TRUE$
    Return(X);
  Else
    Broadcast($X_i[j]$), $j \in [1 : N] \setminus i$;
    Wait Until Receive all X_j, $j \in [1 : N] \setminus i$;
    Return($Majority(X_1, \cdots, X_N)$);

In the above algorithm, * is a concatenation operation of strings, e.g., 000*100 = 000100,and = is an equality evaluation:

$$(X = Y) := \begin{cases} TRUE & \text{if X equals to Y} \\ FALSE & \text{otherwise.} \end{cases}$$

Some padding may be needed if the local result is not an exact multiple of N. The following example demonstrates a rough comparison of the two algorithms.

EXAMPLE 2. $X_1 = X_2 = X_3 = X_4$ = 00000, X_5 = 10000, with *Algorithm 1*, one round of communication is needed and, in total, 25 bits are transmitted. On the other hand, with *Algorithm 2*, P_is (i = 1, $\cdots$, 5) all broadcast 0, and X = 00000, thus (F_1, $\cdots$, F_5) = 11110 , so $Majority(F_1, \cdots, F_5) = 1$, and X is the majority voting result. In this case, two rounds of communication are done, and 10 bits (5 bits for X and 5 bits for F) are transmitted.

If X_5 = 00001, and all other X_is remain the same, then, with *Algorithm 2*, X = 00001, which results in (F_1, $\cdots$, F_5) = 00001 , thus *Majority*(F_1, $\cdots$, F_5) = 0 , and the *Else* part of the algorithm is executed; finally, the majority voting result is obtained by voting on all the X_is, i.e., X = *Majority*(X_1, $\cdots$, X_5) = 00000. Now three rounds of communication are needed and, in total, 30 bits (25 bits for X_is and 5 bits for F) are transmitted.

From the above example it can be observed:

1) *Algorithm 1* always requires only one round of communication and *Algorithm 2* requires two or three rounds of communication;
2) The *Else* part of *Algorithm 2* is actually *Algorithm 1*;
3) The communication complexity of *Algorithm 1* is always Nm, but the communication complexity of *Algorithm 2* may be $m + N$ or $Nm + N$, depending on the X_is;
4) In *Algorithm 2*, by broadcasting and voting on the *voting flags*, i.e., F_is, the chance for getting a *false* voting result is eliminated.

If the *Else* part of *Algorithm 2*, i.e., *Algorithm 1*, is not executed too often, then the communication complexity can be reduced to $m + N$ from Nm and, in most cases, $m \gg N$, thus $m + N \approx m$. So, the key idea used to reduce the communication complexity is to reduce the chance to execute *Algorithm 1*. In most computing environments, each module has low computational error probability p, thus, most probably all modules either

1) get the same result or
2) only few of them get different results from others.

In Case 1, *Algorithm 2* has low communication complexity, but, in Case 2, *Algorithm 1* is actually used and the communication complexity is high (i.e., $Nm + N$), but if we can detect and correct these discrepancies of the minor modules' results, then the *Else* part of the *Algorithm 2* does not need to be executed, the communication complexity can still be low. This detecting and correcting capability can be achieved by using error-correcting codes.

3 A Solution Based on Error-Correcting Codes

Error-correcting codes (*ECC*) can be used in the voting problem to reduce the communication complexity. The basic idea is that instead of broadcasting its own computation result X_i directly, P_i, the ith module, first encodes its result X_i into a codeword Y_i of some code and, then, broadcasts one *symbol* of the codeword to all other modules. After receiving all other *symbols* of the codeword, it reassembles them into a vector again. If all modules have the same result, i.e., all X_is are equal, then the received vector is the codeword of the result, thus it can be decoded to the result again. If the majority result exists, i.e., *Majority*(X_1, $\cdots$, X_N) $\neq \phi$, and there are t ($t \leq \lfloor \frac{N}{2} \rfloor$) modules whose results are different from the *majority result* X, then the *symbols* from all these modules can be regarded as error symbols with respect to the *majority result*. As long as the code is designed to correct up to t errors, these error symbols can be corrected to get the codeword corresponding to the majority result, thus *Algorithm 1* does not need to be executed. When the code length is less than Nm, the communication complexity is reduced compared to *Algorithm 1*. On the other hand, if only *error-detecting codes* are used, once error results are detected, *Algorithm 1* still needs to be executed and, thus, increases the whole communication complexity of the voting. Thus, error-correcting codes are preferable to error-detecting codes for voting. By properly choosing the error-correcting codes, the communication complexity can *always* be lowered than that of *Algorithm 1*.

But, it is possible that the *majority result* does not exist, i.e., *Majority*(X_1, $\cdots$, X_N) = ϕ, yet the vector that each module gets can still be decoded to a result. As observed from the above example, introduction of the voting flags can avoid this *false* result.

3.1 A Voting Algorithm with ECC

With a properly designed error-correcting code which can detect up to d and correct up to t *error symbols* ($0 \leq t \leq d$), a complete voting algorithm using this code is as follows:

Algorithm 3 *ECC Voting*

For Module P_i, $i \in [1 : N]$:
  Y_i := Encode(X_i), partition Y_i into N symbols: $Y_i[1 : N]$;
  Broadcast($Y_i[i]$);
  Wait Until Receive all $Y_j[j]$, $j \in [1 : N] \setminus i$;
  $Y := Y_1[1] * \cdots * Y_N[N]$;
  If Y is undecodable
    Execute *Algorithm 1* ;
  Else
    X := Decode(Y);
    $F_i := (X = X_i)$;
    Broadcast(F_i);
    If Majority(F_1, $\cdots$, F_N) = *TRUE*(1)
      Return(X);
    Else
      Execute *Algorithm 1*;

Notice that to execute *Algorithm 1*, each module P_i does not need to send its whole result X_i. It only needs to send additional $N - (d + t) - 1$ *symbols* of its codeword Y_i. Since the code is designed to detect up to d *and* correct up to t *symbols*, it can correct up to $d + t$ *erasures*, thus the unsent $d + t$ symbols of Y_i can be regarded as *erasures* and recovered, hence, the original X_i can be decoded from Y_i.

To see the algorithm more clearly, the flow chart of the algorithm is given in Fig. 1, and the following example shows how the algorithm works:

Example 3. There are five modules in an NMR system, and the task result has 6 bits, i.e., N = 5 and m = 6. Here, the EVENODD code [2] is used, which divides 6-bit information into three *symbols* and encodes information *symbols* into a five-*symbol* codeword. This code can correct one error *symbol*, i.e., $d = t = 1$.

Now, if X_i = 000000, i = 1, 2, 3, 4, and X_5 = 000011, then, with the EVENODD code, Y_i = 0000000000, i = 1, 2, 3, 4, and Y_5 = 0000111101; after each module broadcasts one *symbol* (i.e., 2 bits) of the codewords, the reassembled vector is Y = 0000000001. Since Y has only

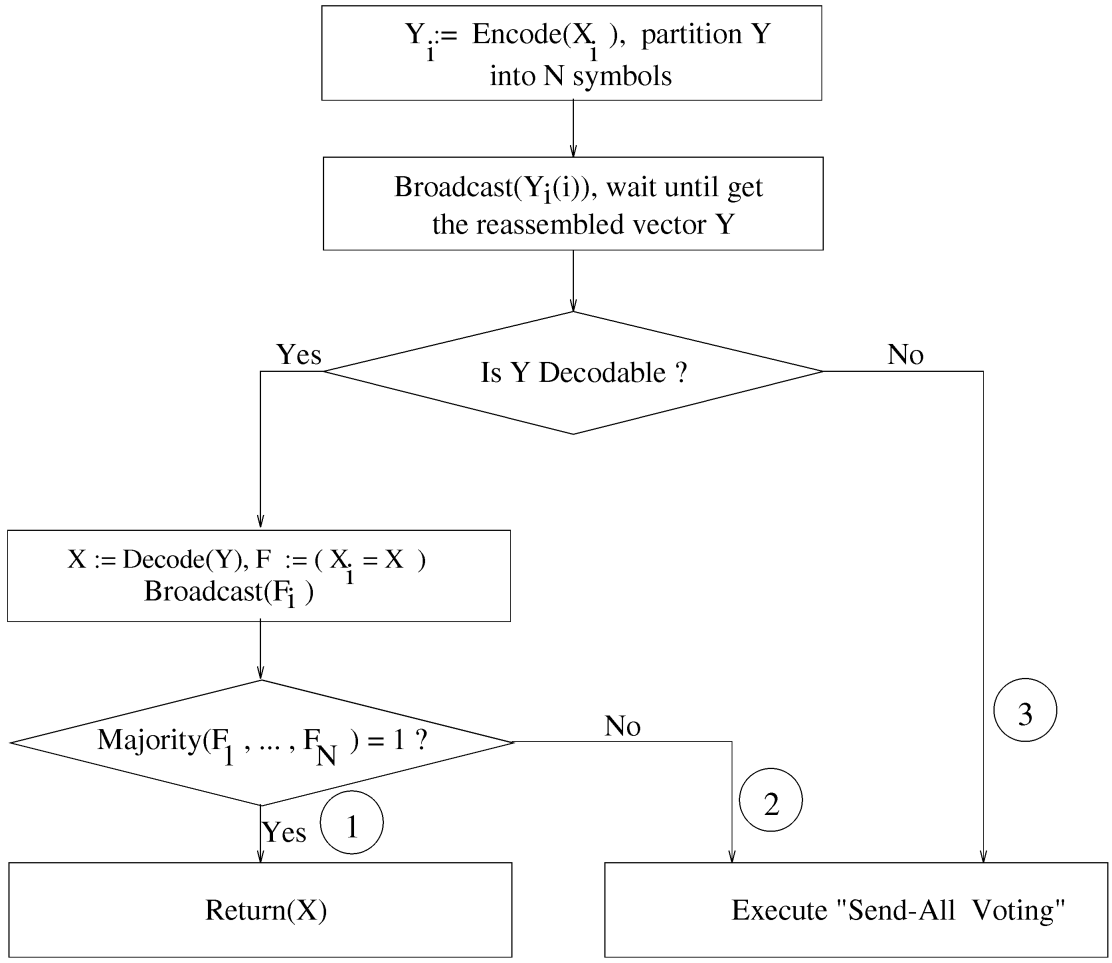


Fig. 1. Flow chart of *Algorithm 3*.

one error symbol, it can be decoded into X = 000000. From the flow chart of the algorithm, we can see that $F_i = 1$, $i = 1, 2, 3, 4$, and $F_5 = 0$, thus, *Majority*$(F_1, \cdots, F_5) = 1$, so X = 000000 is the majority voting result.

In this case, there are two rounds of communication, and the communication complexity is 15 bits. As a comparison, *Algorithm 1* needs one round of communication and its communication complexity is 30 bits; on the other hand, *Algorithm 2* needs three rounds of communication and the communication complexity in this case is 35 bits. In this example, the EVENODD code is used but, actually, the code itself does not affect the communication complexity as long as it has same properties as the EVENODD code, namely, an *MDS* code with $d = t = 1$.

From the flow chart of the algorithm, the introduction of voting on F_is ensures not reaching a *false* voting result, and going to the *Send-All Voting* in worst case guarantees not failing to reach the majority result if it exists. Thus, the algorithm can give a correct majority voting result. A rigorous correctness proof of the algorithm is as follows.

3.2 Correctness of the Algorithm

THEOREM 1. *Algorithm 3 gives Majority*$(X_1, \cdots, X_N)$ *for a given set of local computational results* X_is $(i = 1, \cdots, N)$.

PROOF. From the flow chart of the algorithm, it is easy to see that the algorithm terminates in the following two cases:

1) Executing the *Send-All Voting algorithm*: The correct majority voting result is certainly reached;
2) Returning an X: In this case, since *Majority*$(F_1, \cdots, F_N) = TRUE$, i.e., majority of X_is are equal to X, X is the majority result. □

To see how the algorithm works with various cases of the local results X_is $(i = 1, \cdots, N)$, we give two stronger observations about the algorithm which also help to analyze the communication complexity of the algorithm.

OBSERVATION 1. *If Majority*$(X_1, \cdots, X_N) = \phi$, *then Algorithm 3 outputs* ϕ, *i.e., Algorithm 3 never gives a* false *voting result.*

PROOF. It is easy to see from the flow chart that, after the first round of communication, each module gets a same *vote vector Y*. According to the decodability of Y, there are two cases:

1) If Y is undecodable, then the *Send-All Voting* algorithm is executed, and the output will be ϕ;
2) If Y is decodable, the decoded result X now can be used as a reference result. But, since there does not exist a majority voting result, the majority of the X_is are not equal to the X, i.e., *Majority*$(F_1, \cdots, F_N) =$

FALSE(0), so the *Send-All Voting* algorithm is executed, and the output again will be ϕ. □

OBSERVATION 2. *If Majority*($X_1, \cdots, X_N$) = X ($\neq \phi$), *then Algorithm 3's output is exactly the* X, *i.e., Algorithm 3 will not* miss *the majority voting result.*

PROOF. Suppose there are e modules whose local results are different from the majority result X, then $e \leq \left\lfloor \frac{N-1}{2} \right\rfloor$.

1) If $e \leq t$, then there are e *error symbols* in the *vote vector* Y with respect to the corresponding codeword of the majority result X, so Y can be correctly decoded into X, and the majority of X_is is equal to X, i.e., the majority of F_is is *TRUE*(1), hence, the correct majority result X is output.
2) If $e > t$, then Y is either undecodable or incorrectly decoded into another X', where $X' \neq$ X. In either case, the *Send-All voting* algorithm is executed and the correct majority result X is reached. □

3.3 Proper Code Design

In order to reduce the communication complexity, we need an error-correcting code which can be used in practice for *Algorithm 3*. Consider a block code with length M. Because of the symmetry among the N modules, M needs to be a multiple of N, i.e., each codeword consists of N *symbols* and each *symbol* has k bits, thus $M = Nk$. If the *minimum distance* of the code is d_{min}, then $d_{min} \geq (d + t)k + 1$, where $0 \leq t \leq d \leq \left\lfloor \frac{N}{2} \right\rfloor$, since the code should be able to *detect* up to d error symbols and *correct* up to t error symbols [6]. Recall that the final voting result has m bits, the code to design is an (Nk, m, $(d + t)k + 1$) block code.

To get the smallest value for k, by the *Singleton Bound* in coding theory [6],

$$Nk - m + 1 \geq (d + t)k + 1, \tag{1}$$

we get

$$k \geq \frac{m}{N - (d + t)}. \tag{2}$$

Equality holds for all *MDS Codes* [6].

So, given a designed (d, t), the smallest value for k is $\left\lceil \frac{m}{N-(d+t)} \right\rceil$. If $\frac{m}{N-(d+t)}$ is an integer, all *MDS Codes* can achieve this lower bound of k. One class of commonly used MDS codes for arbitrary distances is the Reed-Solomon code [6]. If $\frac{m}{N-(d+t)}$ is not an integer, then *any* (Nk, m, $(d + t)k + 1$) block code can be used, where $k = \left\lceil \frac{m}{N-(d+t)} \right\rceil$; one of the examples is the BCH code, which can also have arbitrary distances [6]. The exact parameters of (k, d, t) can be obtained by *shortening* (i.e., setting some information symbols to zeros) or *puncturing* (deleting some parity symbols) proper codes [6].

Notice that $0 \leq t \leq d \leq \frac{N-1}{2}$, thus $\frac{m}{N} \leq k \leq m$. In most applications, $N \ll m$, thus the N bits of F_is can be neglected, and k is approximately the number of the bits that each module needs to send to get final voting result, so the communication complexity of *Algorithm 3* is always lower than that of *Algorithm 1*.

In this paper, only the communication complexity of voting is considered since, in many systems, computations for encoding and decoding on individual nodes are much faster than reliable communications among these nodes, which need rather complicated data management in different communication stacks, retransmission of packets between distributed nodes when packet loss happens. However, in real applications, design of proper codes should also make the encoding and decoding of the codes as computationally efficient as possible. When the distances of codes are relatively small, which is the case for most applications when the error probability p is relatively low, more computation-efficient MDS codes exist, such as codes in [2], [10], and [11], all of which require only bitwise exclusive OR operations.

4 COMMUNICATION COMPLEXITY ANALYSIS

4.1 Main Results

From the flow chart of *Algorithm 3*, we can see if the algorithm terminates in *branch 1*, i.e., the algorithm gets a majority result, then the communication complexity is $N(k + 1)$; if it terminates in *branch 2*, then the communication complexity is $N(m + 1)$; finally, if the algorithm terminates in *branch 3*, the communication complexity is Nm, thus the *worst case* communication complexity C_w is $N(m + 1)$. When $m \gg 1$, $C_w \approx Nm$.

Denote C_a as the *average case* communication complexity of *Algorithm 3*, and define the *average reduction factor* α as the ratio of C_a over the communication complexity of the *Send-All Voting* algorithm (i.e., Nm), namely $\alpha = \frac{C_a}{Nm}$, then, the following theorem gives the relation between α and the parameters of an NMR system and the corresponding code:

THEOREM 2. *For an NMR system with N modules, each of which executes an identical task of m-bit result and has computational error with probability p independent of other modules' activities, if Algorithm 3 uses an ECC which can detect up to d and correct up to t error symbols, and* $m \gg N > 1$, *then the following relation holds for the average reduction factor of Algorithm 3*:

$$\alpha < \frac{P_1}{N - (d + t)} + (1 - P_1) + \frac{1}{m}, \tag{3}$$

where

$$P_1 = \sum_{i=0}^{t} \binom{N}{i} p^i (1 - p)^{N-i}. \tag{4}$$

PROOF. To get the average case communication complexity C_a of *Algorithm 3*, we need to analyze the probability P_i of the algorithm terminating in the *branch i*, i = 1, 2, 3. First, assume that if a module has an erroneous result X_i, then it contributes an error symbol to the voting vector Y. From the proof of *Observation 2*, if the algorithm terminates in the *branch 1*, then at most t modules have computational errors, thus, the probability of this event is exactly P_1. The event that the algorithm reaches the *branch 2* corresponds to the *decoder error* event of a code with *minimum distance* of $d + t + 1$, thus [6]

$$P_2 = \sum_{i=d+t+1}^{N} A_i \sum_{k=0}^{\lfloor \frac{d+t}{2} \rfloor} P_{ik}, \quad (5)$$

where $\{A_i\}$ is the *weight distribution* of the code being used and P_{ik} is the probability that a received vector Y is exactly Hamming distance k from a *weight-i* (binary) codeword of the code. More precisely,

$$P_{ik} = \sum_{j=0}^{k} \binom{i}{k-j} \binom{N-i}{j} p^{i-k+2j} (1-p)^{N-i+k-2j}. \quad (6)$$

If the weight distribution of the code is unknown, P_2 can be approximately bounded by

$$P_2 \le 1 - \sum_{i=0}^{\lfloor \frac{d+t}{2} \rfloor} \binom{N}{i} p^i (1-p)^{N-i}, \quad (7)$$

since the second term in the right side of the inequality above is just the probability of event that *correctable* errors happen. Finally, P_3 is the probability of the *decoder failure* event,

$$P_3 = 1 - P_1 - P_2. \quad (8)$$

Now, notice the fact that a module has an erroneous result can also contribute a correct symbol to the voting vector, the average case communication complexity is

$$C_a \le P_1 N(k+1) + P_2 N(m+1) + P_3 Nm \quad (9)$$

and the average reduction factor is

$$\alpha \le \frac{k}{m} P_1 + (1 - P_1) + \frac{P_1 + P_2}{m}. \quad (10)$$

Notice that $k = \left\lceil \frac{m}{N-(d+t)} \right\rceil$ and $P_1 + P_2 < 1$, we get the result of α as in (3). □

REMARKS ON THE THEOREM. From (3), we can see the relation between the average reduction factor α and each branch of *Algorithm 3*. The first term relates to the first branch whose reduction factor is $\frac{k}{m}$ or $\frac{1}{N-(d+t)}$ when m is large enough relative to N, the round-off error of partition can be neglected, and P_1 is the probability of that branch. One would expect this term to be the dominant one for the α, since, with a properly designed code tuned to the system, the algorithm is supposed to terminate at *Branch 1* in most cases. The second term is simply the probability that the algorithm terminates at either *Branch 2* or *Branch 3*, where the reduction factor is 1 (i.e., there is no communication reduction since all the local results are transmitted), without considering the 1 bit for F_is in *Branch 2*. The last term is due to the 1 bit for voting on F_is. When the local result size is large enough, i.e., $m \gg 1$, this 1 bit can be neglected in our model. Thus, in most applications, the result in the theorem can be simplified as

$$\alpha \approx \frac{P_1}{N - (d+t)} + (1 - P_1), \quad (11)$$

since the assumption that $m \gg 1$ is quite reasonable.

From the above theorem and its proof, it can be seen that for a given *NMR* system (i.e., N and p), P_1 is only a function of t, so, if t is chosen, from (3) or (11), it is easy to see that α monotonically decreases as d decreases. Recall that $0 \le t \le d$, thus, for a chosen t, setting $d = t$ can make α minimum when $m \gg 1$. Even though it is not straightforward to get a closed form of t which minimizes α, it is almost trivial to get such an optimal t by numerical calculation.

Fig. 2 shows relations between α and (t, p, N). Fig. 2a and Fig. 2b show how α (using (11)) changes with t for some setup of (N, p) when $d = t$. It is easy to see that, for small p and reasonable N, a small t (e.g., $t \le 2$, for $N \le 51$ with $p = 0.01$) can achieve minimal α. These results show that, for a quite good *NMR* system (e.g., $p \le 0.01$), only by putting a small amount of redundancy of the local results and applying error-correcting codes on them, the communication complexity of the majority voting can be drastically reduced. Since the majority result is of m bits and each module shall get an identical result after the voting, the communication complexity of the voting problem is at least m bits, thus $\alpha \ge \frac{1}{N}$, i.e., $\frac{1}{N}$ is the lower bound of α. Fig. 2c shows the closeness of the theoretical lower bound of α and the minimum α that *Algorithm 3* can achieve for some setup of *NMR* systems.

4.2 More Observations

From the above results, we can see that the communication complexity of the *Algorithm 3* is determined by the code design parameters (d, t). In an *NMR* system with N modules, we only need to consider the case where at most $\lfloor \frac{N}{2} \rfloor$ modules have different local results with the majority result, thus the only constraint of (d, t) is $0 \le t \le d \le \lfloor \frac{N}{2} \rfloor$. For some specific values of (d, t), the algorithm reduces to the following cases:

1) When $d = t = \frac{N-1}{2}$, i.e., a *repetition* code is used, the algorithm becomes *Algorithm 1*. Since a repetition code is always the worst code in terms of redundancy, it should always be avoided for reducing the communication complexity of voting. On the other hand, when $d = t = 0$, the algorithm becomes *Algorithm 2* and, from Fig. 2, we can see that, for a small enough p and reasonable N, e.g., $p = 10^{-5}$ with $N = 31$, *Algorithm 2* actually is a best solution of the majority voting problem in terms of the communication complexity. Besides, *Algorithm 2* has low computational complexity since it does not need any encoding and decoding operations. Thus, the *ECC* voting algorithm is a generalized voting algorithm and its communication complexity is determined by the code chosen.
2) $t = 0$, then the code only has detecting capability but, if $m \gg N$, then from the analysis above, increasing d actually makes α increasing. Thus, it is not good to put some redundancy to the local results only for detecting capability when $m \gg N$, i.e., using only *EDC* (*error-detecting code*) does not help to reduce the communication complexity of voting. The scheme proposed in [8] is in this class with $d = \lfloor \frac{N}{2} \rfloor$.

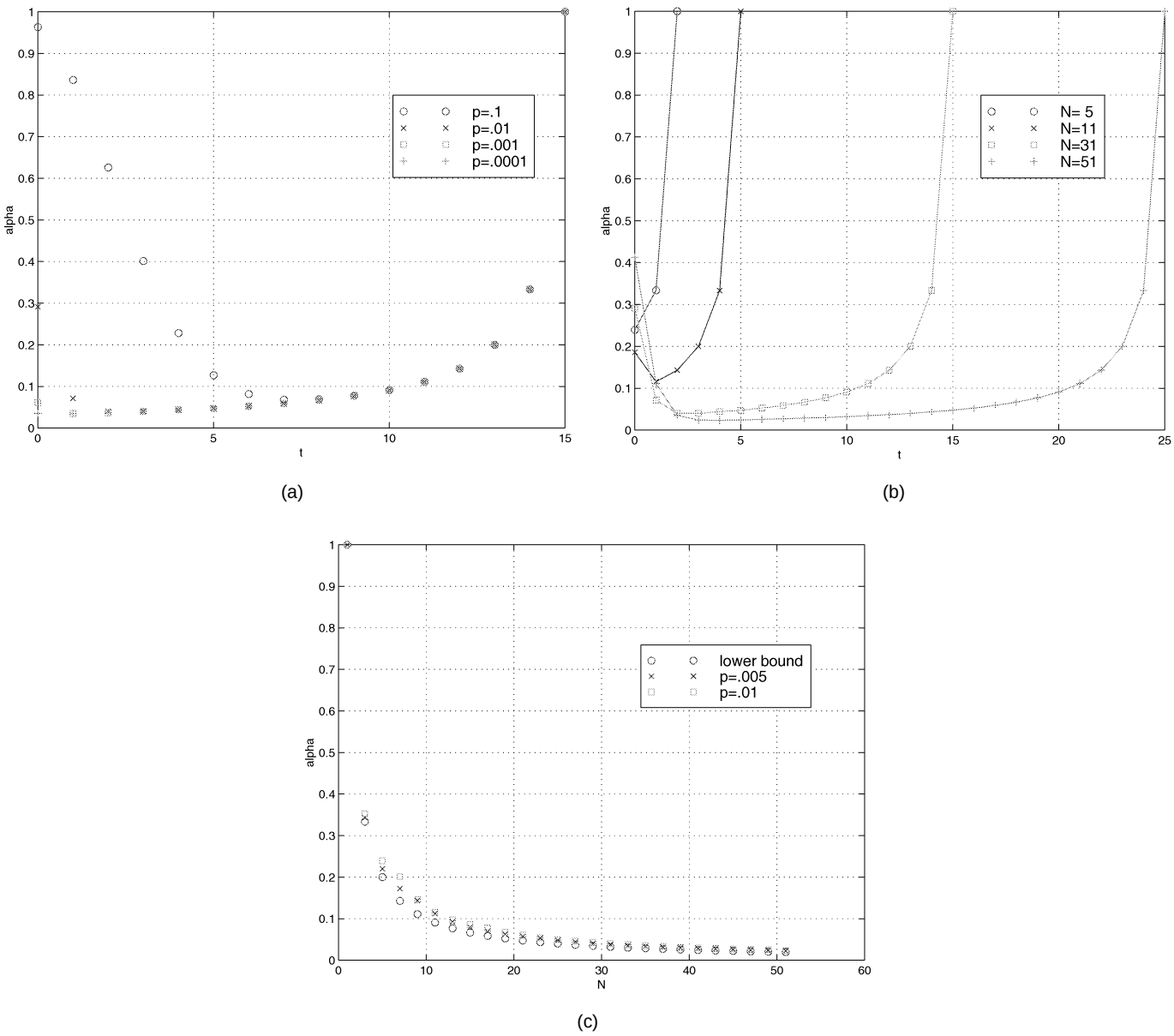


Fig. 2. Relations between α and (t, p, N). (a) α vs. t for different p with fixed N = 31. (b) α vs. t for different N with fixed p = 0.01. (c) Minimum α vs. N.

3) $d = \lfloor \frac{N}{2} \rfloor$: As analyzed above, in general it is not good to have $d > t$ in terms of α since increase of d will increase α when t is fixed. But, in this case, *Algorithm 3* has a special property: *Branch 2* of the algorithm can directly come to declare there is *no* majority result *without* executing the *Send-All Voting* algorithm, simply because the code now can detect up to $\lfloor \frac{N}{2} \rfloor$ errors, so, if there was a majority result, then Y (refer to Fig. 1) can have at most $\lfloor \frac{N}{2} \rfloor$ erroneous modules and, since Y is decodable, the majority of the local results should agree with the decoded result X, i.e., *Majority*(F_1, ⋯, F_N) = *TRUE*; this contradicts with the actual *Majority*(F_1, ⋯, F_N), so there is *no* majority result. By setting d to $\lfloor \frac{N}{2} \rfloor$, *Algorithm 3* always has *two rounds* of communication and the *worst case* communication complexity is, thus, Nm, instead of $N(m + 1)$, for the general case, and this achieves the lower bound of the *worst case* communication complexity of the distributed majority voting problem [8].

5 Experimental Results

In this section, we show some experimental results of the three voting algorithms discussed above. The experiments are performed over a cluster of Intel Pentium/Linux-2.0.7 nodes connected via a 100 Mbps Ethernet. Reliable communication is implemented by a simple improved UDP scheme: Whenever there is a packet loss, the voting operation is considered as a failure and redone from beginning. By choosing suitable packet size, there is virtually no packet loss using UDP.

To examine real performances of the above three voting algorithms, N nodes vote on a result of length m using all

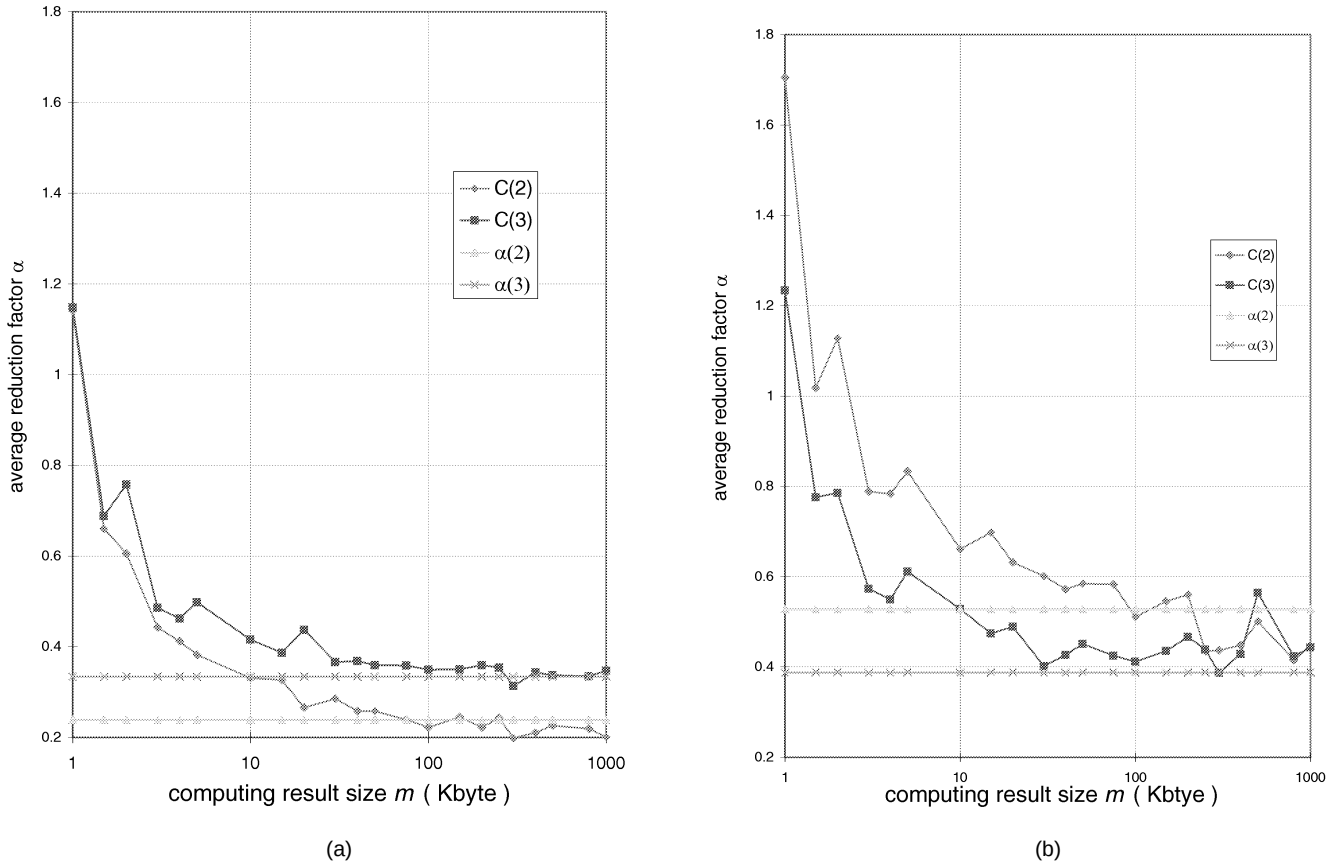


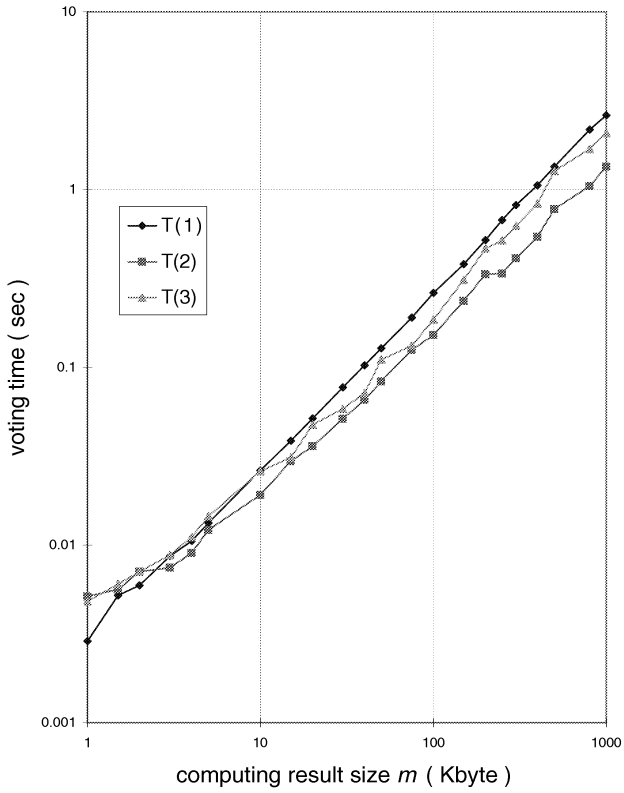
Fig. 3. Average reduction factors (C(*i*) is the experimental average reduction factor of communication time for voting using *Algorithm i*, and α(i) is the theoretical bound of the average communication reduction factor using *Algorithm i*, *i* = 2, 3). (a) Error probability *p* = 0.01. (b) Error probability *p* = 0.1.

the three voting algorithms. For the *ECC Voting* algorithm, an *EVENODD Code* is used, which corrects 1 *error symbol*, i.e., $d = t = 1$ for the *ECC Voting* algorithm. Random errors are added to local computing results with a preassigned error probability *p*, independent of results at other nodes in the NMR system. Performances are evaluated by two parameters for each algorithm: the total time to complete the voting operation *T* and the communication time for the voting operation *C*. Among all the local *T*s and *C*s, the maximum *T* and *C* are chosen to be the *T* and *C* of the whole NMR system, since, if the voting operation is considered as a collective operation, the system's performance is determined by the *worst* local performance in the system. For each set of the NMR system parameters (*N* nodes and error probability *p*), each voting operation is done 200 times and random computation errors in each run are independent of those in other runs, and the arithmetic average of *C*s and *T*s are regarded as the performance parameters for the tested NMR system.

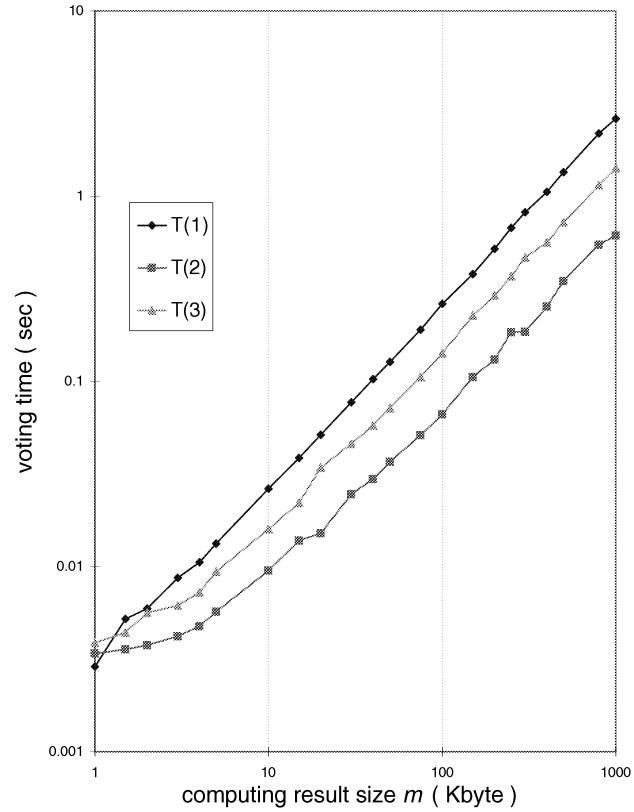
Experimental results are shown in Figs. 3, 4, and 5. Fig. 3 compares the experimental *average reduction factors* of the voting algorithms with the theoretical results as analyzed in the previous section, when they are applied in an NMR system of five nodes. Fig. 4 shows the performances (*T* and *C*) of the voting algorithms. Detailed communication patterns of the voting algorithms are shown in Fig. 5 to provide some deeper insight into the voting algorithms.

Fig. 3a and Fig. 3b show the experimental *average reduction factors* of the voting communication time (*C*) for the *Simple Send-Part Voting* algorithm and the *ECC Voting* algorithm. Fig. 3a and Fig. 3b also show the theoretical average reduction factors of Algorithms 2 and 3 as computed from (11). Notice that the average communication time reduction factors α of both *Algorithm 2* and *Algorithm 3* are below 1 and, as the computing result size increases, the reduction factor approaches the theoretical bound, with the exception of the smallest computing result size of 1 Kbyte.

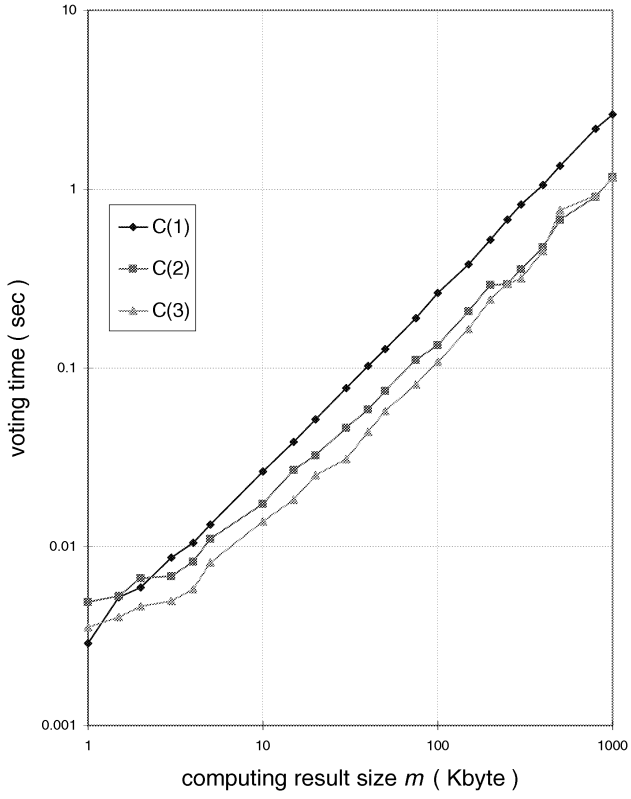
Fig. 4 shows the performances of each voting algorithm applied in an NMR system of five nodes. Figs. 4a and 4b show the *total* voting time *T* and Figs. 4c and 4d show the *communication* time *C* for voting. The only different parameter of the NMR systems related to Figs. 4a and 4b (symmetrically, Figs. 4c and 4d) is the error probability *p*: *p* = 0.1 in Figs. 4a and 4c, while *p* = 0.01 in Figs. 4b and 4d. It is easy to see from the figures that, for the voting *Algorithm 1* (*Send-All Voting*), *T* and *C* are the same since, besides communication, there is no additional local computation. Figs. 4a and 4b show that Algorithms 2 (*Simple Send-Part Voting*) and 3 (*ECC Voting*) perform better than Algorithm 1 (*Send-All Voting*) in terms of the total voting time *T*. On the other hand, Figs. 4c and 4d show, in terms of *C*, i.e., the communication complexity, the *ECC Voting* algorithm is better than the *Simple Send-Part Voting* algorithm when the error probability is relatively large (Fig. 4c) and worse than the *Simple Send-Part Voting* algorithm when the error probability is relatively small (Fig. 4d), which is consistent with the analysis results in the previous section.



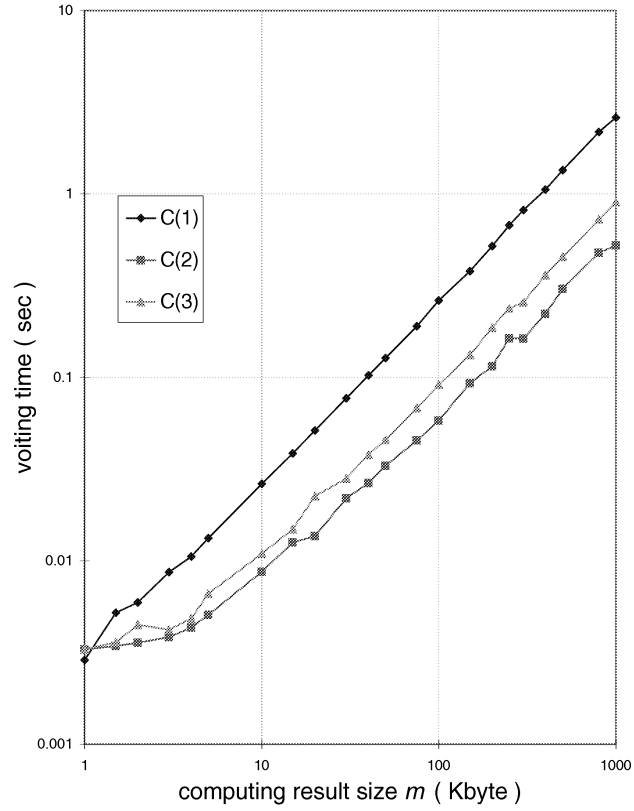
(a)



(b)

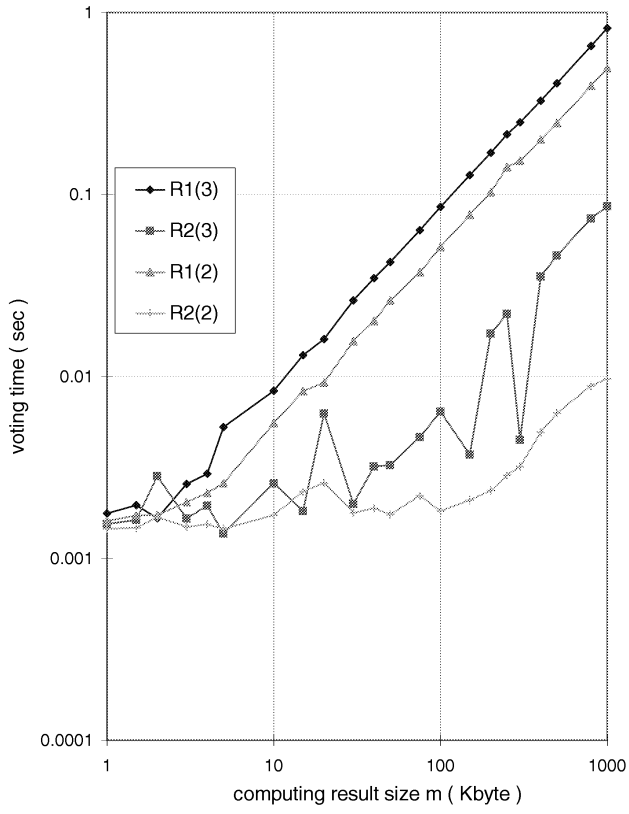


(c)

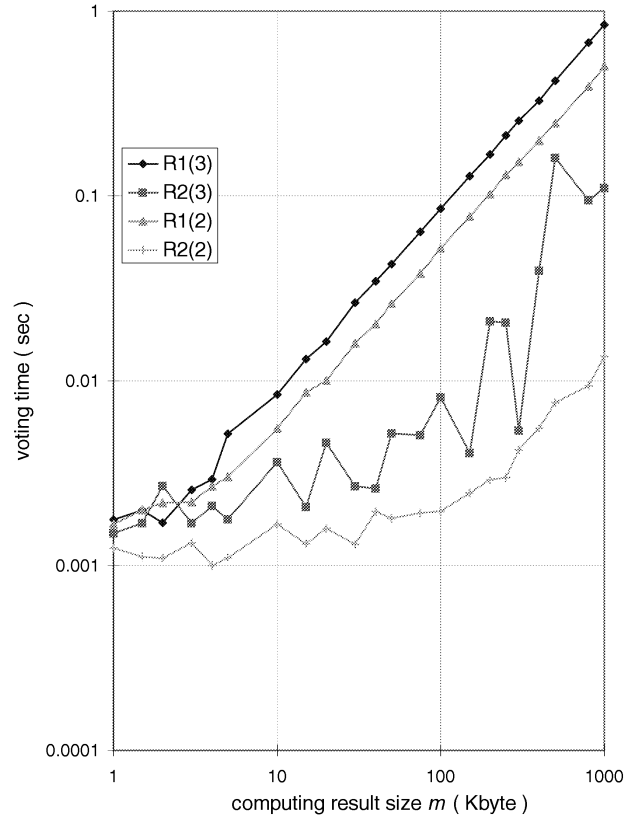


(d)

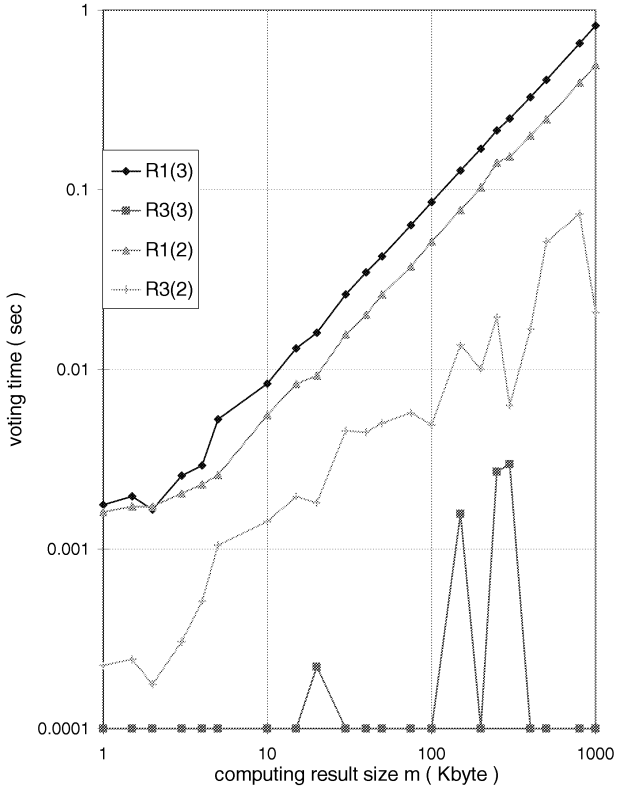
Fig. 4. Experimental voting performances of five-node NMR system (T(*i*) and C(*i*) are the total and communication time for voting using *algorithm i*, respectively, *i* = 1, 2, 3). (a) error probability *p* = 0.1., (b) error probability *p* = 0.01, (c) error probability *p* = 0.1, (d) error probability *p* = 0.01.



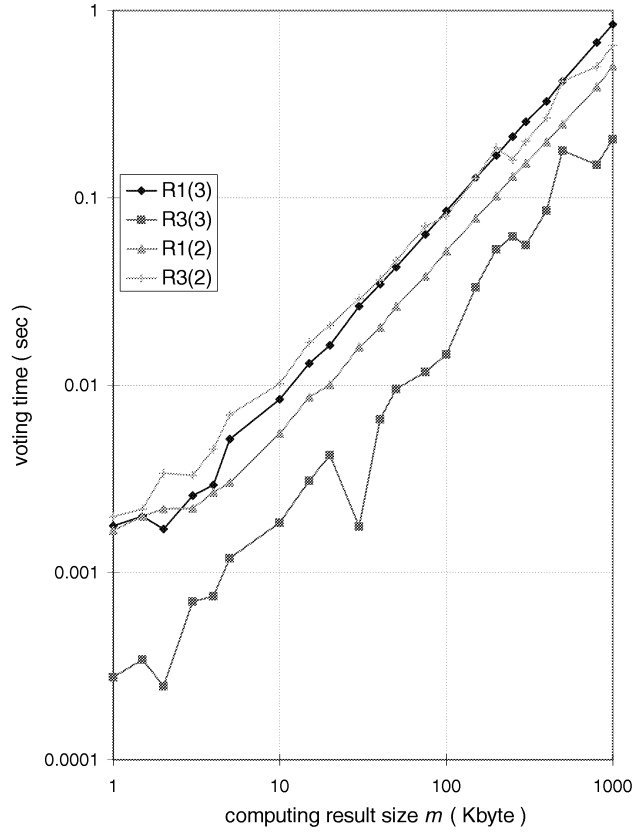
(a)



(b)



(c)



(d)

Fig. 5. Detailed communication time pattern of voting (R*i*(*k*) is the communication time in *round i* using the voting *algorithm k*, *i* = 1, 2, 3, and *k* = 2, 3). a) error probability *p* = 0.01., (b) error probability *p* = 0.1, (c) error probability *p* = 0.01, (d) error probability *p* = 0.1.

In the analysis in the previous section, the size of local computing result m does not show up as a variable in the average reduction factor function α, since the communication complexity is only considered as proportional to the size of the messages that need to be broadcast. But, practically, communication time is *not* proportional to the message size, since startup time of communication also needs to be included. More especially, in the Ethernet environment, since the maximum packet size of each physical send (broadcast) operation is also limited by the physical ethernet, communication completion time becomes a more complicated function of the message size. Thus, from the experimental results, it can be seen that, for a computing result of small size, e.g., 1 Kbyte, the *Send-All Voting* algorithm actually performs best in terms of both C and T, since the startup time dominates the performance of communication. Also, the communication time for broadcasting the 1-bit *voting flags* cannot be neglected, as analyzed in the previous section, particularly for a small size computing result. This can also be seen from the detailed voting communication time pattern in Figs. 5a and 5b: *Round 2* of the communication is for the 1-bit *voting flag*, even though it finishes in a much more shorter time than *round 1*, but is still not negligibly small. This explains the fact that, for small size computing results, the average communication time reduction factors of *Algorithm 2* and *Algorithm 3* are quite apart from their theoretical bound.

Further examination of the detailed communication time pattern of voting provides a deeper insight into *Algorithm 3*. From Figs. 5c and 5d, it is easy to see that, in the first round of communication, *Algorithm 2* needs *less* time than *Algorithm 3* since the size of the message to be broadcast is smaller for *Algorithm 2*. Besides, the first round of communication time does not vary as the error probability p varies for both algorithms. The real difference between the two algorithms lies in the third round of communication. From Fig. 5c, this time is small for the both algorithms since the error probability p is small (0.01). But, as the error probability p increases to 0.1, as shown in Fig. 5d, for *Algorithm 2*, this time also increases to be bigger than the first round time, since it has no *error-correcting* capability and, once full message needs to be broadcast, its size is much bigger than in the first round. On the other hand, for *Algorithm 3*, though it also increases, the communication time for the third round is still much smaller than in the first round; this comes from the error-correcting codes that *Algorithm 3* uses, since the code can correct errors at one computing node, which is the most frequent error pattern that happens. Thus, even though the error probability is high, in most cases, the most expensive third round of communication can still be avoided, and *Algorithm 3* performs better (in terms of communication complexity or time) than *Algorithm 2* in high error probability systems, just as the predicted analysis in the previous section.

6 Conclusions

We have proposed a deterministic distributed voting algorithm using error-correcting codes to reduce the communication complexity of the voting problem in NMR systems. We also have given a detailed theoretical analysis of the algorithm. By choosing the design parameters of the error-correcting code, i.e., (d, t), the algorithm can achieve a low communication complexity which is quite close to its theoretical lower bound. We have also implemented the voting algorithm over a network of workstations, and the experimental performance results match the theoretical analysis well. The algorithm proposed here needs two or three rounds of communication. It is left as an open problem whether there is an algorithm for the distributed majority voting problem with its *average case* communication complexity less than Nm using *only* 1 round of communication.

Acknowledgments

This work was supported in part by U.S. National Science Foundation Young Investigator Award CCR-9457811, by the Sloan Research Fellowship, and by DARPA through an agreement with NASA / OSAT.

References

[1] K.A.S. Abdel-Ghaffar and A. El Abbadi, "An Optimal Strategy for Computing File Copies," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 1, Jan. 1994.

[2] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," *IEEE Trans. Computers*, vol. 44, no. 2, pp. 192-202, Feb. 1995.

[3] D.M. Blough and G.F. Sullivan, "Voting Using Predispositions," *IEEE Trans. on Reliability*, vol. 43, no. 4, pp. 604-616, 1994.

[4] K. Echtle, "Fault-Masking with Reduced Redundant Communication," *Proc. 16th Ann. Int'l Symp. Fault-Tolerant Computing Systems*, vol.16, pp. 178-183, 1986.

[5] D.D.E. Long and J.-F. Pâris, "Voting Without Version Numbers," *Proc. Int'l Conf. Performance, Computing, and Comm.*, pp. 139-145, Feb. 1997.

[6] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes*. Amsterdam: North-Holland, 1977

[7] J.F. Nebus, "Parallel Data Compression for Fault Tolerance," *Computer Design*, pp. 127-134, Apr. 1983.

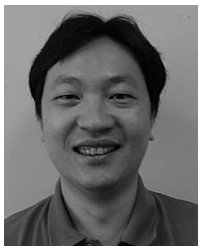
[8] G. Noubir and H.J. Nussbaumer, "Using Error Control Codes to Reduce the Communication Complexity of Voting in NMR Systems," technical report, Dept. of Computer Science, Swiss Federal Inst. of Technology in Lausanne (EPFL), 1995.

[9] B. Parhami, "Voting Algorithms," *IEEE Trans. Reliability*, vol. 43, no. 4, pp. 617-629, 1994.

[10] L. Xu and J. Bruck, "X-Code: MDS Array Codes with Optimal Encoding," *IEEE Trans. Information Theory*, to appear, 1998.

[11] L. Xu, V. Bohossian, J. Bruck, and D. Wagner, "Low Density MDS Codes and Factors of Complete Graphs," *Proc. 1998 IEEE Symp. Information Theory*, Aug. 1998

[12] A. Ziv and J. Bruck, "Checkpointing in Parallel and Distributed Systems," *Parallel and Distributed Computing Handbook*, pp. 274-302. McGraw-Hill, 1996.



Lihao Xu received the BSc and MSc degrees in electrical engineering from the Shanghai Jiao Tong University, China, in 1988 and 1991, respectively. From 1991 to 1994, he was a lecturer in the Electrical Engineering Department of the Shanghai Jiao Tong University. He is currently a PhD candidate in the Electrical Engineering Department of the California Institute of Technology. His current research interests include parallel and distributed computing, fault-tolerant computing, error-correcting codes, and server systems. He holds one pending patent.

Jehoshua Bruck received the BSc and MSc degrees in electrical engineering from the Technion, Israel Institute of Technology, in 1982 and 1985, respectively, and the PhD degree in electrical engineering from Stanford University in 1989.

He is a professor of computation and neural systems and electrical engineering at the California Institute of Technology. His research interests include parallel and distributed computing, fault-tolerant computing, error-correcting codes, computation theory, and biological systems. Dr. Bruck has extensive industrial experience, including serving as manager of the Foundations of Massively Parallel Computing Group at the IBM Almaden Research Center from 1990 to 1994, a research staff member at the IBM Almaden Research Center from 1989 to 1990, and as a researcher at the IBM Haifa Science center from 1982 to 1985.

Dr. Bruck is the recipient of a 1997 IBM Partnership Award, a 1995 Sloan Research Fellowship, a 1994 U.S. National Science Foundation Young Investigator Award, a 1992 IBM Outstanding Innovation Award for his work on “Harmonic Analysis of Neural Networks,” and a 1994 IBM Outstanding Technical Achievement Award for his contributions to the design and implementation of the SP-1, the first IBM scalable parallel computer. He has published more than 130 journal and conference papers in his areas of interests and he holds 21 patents. Dr. Bruck is a senior member of the IEEE and a member of the editorial board of the *IEEE Transactions on Parallel and Distributed Systems*.