Dynamic Fault Diagnosis

William Hurwood

Yale University
will@math.yale.edu

Abstract

We consider a dynamic fault diagnosis problem: There are n processors, to be tested in a series of rounds. In every testing round we use a directed matching to have some processors report on the status (good or faulty) of other processors. Also in each round up to t processors may break down, and we may direct that up to t processors are repaired. We show that it is possible to limit the number of faulty processors to $O(t \log t)$, even if the system is run indefinitely. We present an adversary which shows that this bound is optimal.

1 Introduction

Consider a computer system made up of thousands of processors. It is inevitable that some of the processors will fail. Fault tolerant systems will function under the presence of a limited number of faults, but even these must be located and fixed before too many accumulate. One approach to finding these faulty processors is by using some of the processors to test others.

This problem has led to the development of system level fault diagnosis. Each of n atomic processing units is assumed to be able to test every other unit. But when a faulty unit carries out a test the result is unreliable. Such testing protocols have been usefully implemented [5, 4].

Preparata, Metze and Chien first proposed a fault diagnosis model in [11]. They suggested viewing the system as a graph with the processors as nodes and tests as edges. Nakajima [10] introduced *adaptive* tests, where the tests performed in later rounds were chosen after considering the results of the earlier rounds.

This naturally led to asking how many rounds of adaptive tests are needed to carry out a complete diagnosis. This model was used in [6, 8, 2, 7, 3] in which the upper bound on the number of rounds of testing as a function of n was eventually reduced from a linear bound, through logarithmic and doubly-logarithmic bounds to a large constant bound. Recent work in [1] has reduced the upper bound to 10 rounds. [1] also shows that at least 5 rounds are needed to carry out complete diagnosis in the most general case.

In this paper we will consider a natural extension of this model. We will remove the requirement that every processor have a fixed status throughout the testing procedure. Such *dynamic* models have been considered before [9, 5, 4]. In these papers the authors presented self-stabilizing distributed algorithms: if the processors stopped changing status then the remaining good processors would eventually discover the status of all the processors in the system.

We will consider a different model, the *ongoing* fault diagnosis model defined in Section 2. In this model up to t processors will break down, and up to t processors can be repaired in each round. The object is to select the tests that are performed in such a way that we can find an upper bound K on the number of faulty nodes that could ever be simultaneously present in the system.

We will show two major results. First we will show, by presenting an algorithm, that we can take $K = O(t \log t)$, independent of the size of n. That is that regardless of the number of processors it is possible to ensure that at any time $O(t \log t)$ processors are simultaneously faulty.

Second we will show that up to a constant factor this bound is the best possible. An adversary can force at least $t \log t$ simultaneously faulty processors.

Section 2 defines the model. In Sections 3 and 4 the testing algorithm is presented, and analyzed. The lower bound is proved in Section 5.

2 Definitions

We formalize the *ongoing dynamic parallel model* of fault diagnosis below:

- The computing system is considered to be partitioned into n complex indivisible units called *processors* or *nodes.*
- The operations on the system are performed as a sequence of *rounds.* Each round is divided into three parts. In the first part testing is performed, in the second part some processors are repaired, in the third part some processors are corrupted.
- At the beginning of each round every processor has a *status.* It can either be *good* or *faulty.* At the start of the procedure all of the processors have *good* status.
- A *test* is an interaction between two processors, a *tester* and a *testee.* The tester claims to determine the status of the testee. If the tester is good then it will correctly diagnose the testee. But if the tester is itself faulty then it has the option of incorrectly diagnosing the testee. A processor is not permitted to test itself. If the tester reports that the testee was good we shall say that the tester likes the testee, or that the *test was good.*
- Two or more tests may be performed simultaneously within the testing part of a round. But no processor may participate in more than one test during a round of testing. (A round of tests is a directed matching on the processors, but not necessarily a perfect matching.)
- In the *repair* part of the round up to t processors may be repaired. When a processor is repaired, its status immediately becomes good.
- In the third part of the round, up to t processors may be corrupted. When a processor is *corrupted* its status is immediately changed to faulty. It is possible for a processor to be repaired and then corrupted again in the same round.

We will find it convenient to view the testing process as a contest between a deterministic *controller* and an omniscient *adversary.* The controller decides which tests to perform and which processors to repair, knowing only the results of earlier tests. The adversary hinders the controller by using its knowledge of the controller's future behavior to select which processors are corrupted.

The procedure described above will be performed indefinitely. The controller's objective is to ensure that there is an upper bound K on the number of faulty processors present at the start of every round, no matter what strategy the adversary follows. Also the controller must only repair processors that it can prove to be faulty. The adversary seeks to ensure that there is some round in which more than K processors are faulty.

3 Sifting Procedure

The object of this section and the following section is to describe an algorithm that the controller can use to perform ongoing diagnosis. In this section we will describe a procedure called SIFTING, which will be used repeatedly in the ongoing diagnosis.

SIFTING is performed on a set M of nodes. SIFTING tells the controller which tests to perform between the nodes in M, and how to use the test results to partition M into a class G that is mostly good and a class F that is mostly faulty. F contains *all* the nodes that were already faulty before SIFTING was applied. F may also contain some nodes that were good then, but there is an upper bound on the number of these initially good nodes that get placed in F. G does not contain *any* nodes that were faulty in the first round in which SIFTING was applied.

However nodes that become faulty during the performance of SIFTING could be placed in either class. Since the algorithm takes a known number of rounds, we have an upper bound on the number of faulty nodes which could be present in G at the end of the algorithm.

The way that SIFTING works is to grow large subsets of M, which have the property: either all the nodes in the set are faulty at the *end* of the procedure, or none of the nodes in the set were faulty at the *start* of the procedure. Provided we have an upper bound k on the number of nodes that were faulty before we started to apply SIFTING we can conclude that a sufficiently big subset with the property cannot contain any nodes that were faulty at the start of the procedure.

The subsets are grown by a procedure DOUBLE (Algorithm 2) which pairs off the elements of two subsets. Algorithm 1 is called DOUBLING. It initializes the subsets, and calls DOUBLE to grow them. It returns the subsets A_i^d that it grew, along with a set of processors B that were discarded during the doubling operation; these processors are said to have been *rejected.*

3.1 Double and Doubling

Suppose we apply DOUBLING to a set M of m processors. Let d be the number of times that DOUBLE is called, and suppose that $2^d|m$. We have the following lemma

Lemma 1 *If at most k processors were faulty before we applied* DOUBLING *then*

a. DOUBLING *takes $2d$ rounds of testing to carry out.* $|C_j^d| = 2^d$.

b. If any one of the processors in A_j^i was faulty at the start of DOUBLING *then the entire set A_j^i is faulty when it is generated by* DOUBLE.

c. $0 \leq l \leq k + (2d-1)t$. *At least l of the rejected processors are faulty at the end of* DOUBLING.

d. $|B| \leq 2^d + 2l - 2$.

Proof: Part a is clear. Part b is easily obtained by induction on the index i of DOUBLING. Suppose that A_{2j}^i and A_{2j-1}^i both satisfy the property, and DOUBLE forms A_j^{i+1} from them. Suppose A_j^{i+1} contains an initially faulty node p, wlog $p \in A_{2j}^i$. Consider $q \in A_j^{i+1}$, so $q \in A_{2j}^i$ or $q \in A_{2j-1}^i$. In the first case q is faulty by induction. In the second case, by

Input: (M, d): M is a set of m processors, d is the number of times DOUBLE will be called.

Output: $(A_1^d, \ldots, A_{m/2^d}^d, B, l)$: $A_1^d, \ldots, A_{m/2^d}^d, B$ is a partition of M. l is an integer such that B contains at least l faulty processors.

Method: Suppose $M = \{a_1, a_2, \ldots, a_m\}$ and that $2^d | m$. Set $l_0 = 0$ and

$$\begin{array}{llllll} A_1^0 & = \{a_1\} & A_2^0 & = \{a_2\} & \ldots & A_m^0 & = \{a_m\} \\ B_1^0 & = \emptyset & B_2^0 & = \emptyset & \ldots & B_m^0 & = \emptyset \\ C_1^0 & = \{a_1\} & C_2^0 & = \{a_2\} & \ldots & C_m^0 & = \{a_m\} \end{array}$$

```
For $i = 0$ to $d - 1$ do $(C_*^{i+1}, A_*^{i+1}, B_*^{i+1}, l_{i+1}) = \text{DOUBLE}(C_*^i, A_*^i, B_*^i, l_i)$;
Set $B = \bigcup_j B_j^d$ and $l = l_d$.
```

Algorithm 1: The procedure DOUBLING

examining Algorithm 2, we see that q liked its partner from A_{2j}^i which is faulty by induction. Since q liked a faulty node while DOUBLE was being performed, q must by faulty at that time. So A_j^{i+1} is entirely composed of faulty nodes, when it is constructed. The base case for the induction is immediate.

For part c observe that l counts the number of pairs that are put into B because one element of the pair doesn't like the other. So at least one of the processors in the pair must be faulty, and so l is a lower bound on the number of faulty processors in B. The upper bound on l follows since at most $k + (2d - 1)t$ nodes of M are faulty when the tests are performed.

The reason that the upper bound on l is $k + (2d - 1)t$ not $k + 2dt$ is because the testing part of each round comes before the part in which processors are corrupted. So any processors which are corrupted during the final round of DOUBLE still had good status when all the tests took place. So they connot contribute to the size of l.

For part d observe that there are two ways a node could be rejected. Either it is in a test in which one node doesn't like the other one, or for some i, j it is one of the $|A_{2j}^i| - |A_{2j-1}^i|$ nodes from A_{2j}^i which are rejected because there are no nodes in A_{2j-1}^i to pair them with. As we have seen there are exactly $2l$ nodes which are rejected for the first reason. The reordering of the A_j^i described by equation (1) ensures that in the ith call to DOUBLE at most $|C^{i-1}| = 2^{i-1}$ nodes can be rejected for second reason. However when $i = 1$, every A_j^1 is a single processor, so no processors are rejected on the first call to DOUBLE. So on summing we get that at most $2l + 2 + 4 + \cdots + 2^{d-1}$ nodes can be rejected. □

3.2 Sifting

DOUBLING returns sets A_i^d, and B which satisfy lemma 1. SIFTING (see Algorithm 3) calls DOUBLING and constructs a partition of M, into two sets G and F. It defines F to be the

Input: $(C_1, \ldots, C_{2r}, A_1, \ldots, A_{2r}, B_1, \ldots, B_{2r}, l)$: The C_j are equally sized disjoint sets. Each C_j is partitioned into two sets: A_j and B_j. l is an integer such that $\cup B_j$ contains at least l faulty processors.

Output: $(C'_1, \ldots, C'_r, A'_1, \ldots, A'_r, B'_1, \ldots, B'_r, l')$: The C'_j are equally sized disjoint sets with $\cup C_j = \cup C'_j$. Each C'_j is partitioned into two sets: A'_j and B'_j. l' is an integer such that $\cup B'_j$ contains at least l' faulty processors.

Method: We can reorder the A_i so they satisfy

$$0 \le |A_1| \le |A_2| \le \ldots \le |A_{2r}| \le |C_1|. \tag{1}$$

Set $C'_j = C_{2j-1} \cup C_{2j}$, for $1 \le j \le r$. The procedure takes two rounds. In one round have every processor in A_{2j-1} test a different processor in A_{2j}. In the second round, let each processor in A_{2j} that was tested in the first round, test the processor from A_{2j-1} that tested it.

Let A'_j consist of those pairs of processors from A_{2j-1} and A_{2j} which liked each other. Let B'_j contain any other processors from C'_j. Set l' to be l plus the number of pairs tested where some processor in the pair didn't like the other processor.

Algorithm 2: The procedure DOUBLE

union of B and those A^d_j satisfying

$$|A^d_j| \le k + (2d-1)t - l. \tag{2}$$

We say that G is the set of processors *accepted* by SIFTING, F is the set of *rejected* processors. We say that A^d_j is *rejected* if its members are placed in F.

Lemma 2 *All the processors accepted by* SIFTING *were good at the start of the procedure.*

Proof: Suppose $a \in M$ and a was initially faulty. If $a \in B$ then a is rejected, since $B \subseteq F$.

Input: (M, d, k): M is a set of m processors, d is the number of times that DOUBLE is to be called and k is an upper bound on the number of faulty processors initially present in M.

Output: (G, F): A partition of M.

Method: Let $(A^d_1, \ldots, A^d_{m/2^d}, B, l) =$ DOUBLING(M, d). Set $G = M \setminus F$ where

$$F = B \cup \bigcup \{A^d_j : |A^d_j| \le k + (2d-1)t - l\}.$$

Algorithm 3: The procedure SIFTING

Otherwise we have $a \in A_j^d$ for some j, and by lemma 1 part b all the processors in A_j^d are faulty during the last round of tests. There are at most $k + (2d-1)t$ faulty processors in M, and by part c at least l of them are in B and so can't be in A_j^d. Thus we have that

$$|A_j^d| \leq k + (2d-1)t - l.$$

But this is precisely the condition that will cause A_j^d to be rejected. $\square$

Now we will use the bound on $|B|$ given in lemma 1, to prove an upper bound on the size of F.

Suppose A_j^d is rejected. Then since $|A_j^d| + |B_j^d| = 2^d$ we have

$$|B_j^d| \geq 2^d - k - (2d-1)t + l. \tag{3}$$

Consider a set C_j^d whose A_j^d part is rejected. Then equation (3) gives a lower bound on $|B_j^d|$. But lemma 1 gave an upper bound on $|B|$, the union of all the B_j^d. So we have an upper bound on the number of C_j^d whose A_j^d parts were rejected. $|F|$ cannot be greater than the value obtained by taking B and each rejected A_j^d to be as large as possible, and by rejecting as many A_j^d as possible. That is

$$|F| \leq \frac{\max(|B|).\max(|\text{rejected } A_j^d|)}{\min(|\text{rejected } B_j^d|)} + \max(|B|) \tag{4}$$

where by "rejected B_j^d" we mean a B_j^d whose corresponding A_j^d was rejected.

Substituting in the lower and upper bounds for these quantities that were proved in (2, lemma 1(d), 3) and simplifying we get

$$|F| \leq \frac{(2^d - 2 + 2l)2^d}{2^d - k - (2d-1)t + l}. \tag{5}$$

This bound on the size of F depends on l. But the adversary is able to control the size of l by the manner in which it arranges the faulty processors, so we need a bound on F independent of l.

Theorem 3

$$|F| \leq \begin{cases} 2^d + 2k + 2(2d-1)t - 2 & \textit{when } 2k + 2(2d-1)t - 2 \leq 2^d \\ \frac{(2^d-2)2^d}{2^d-k-(2d-1)t} & \textit{when } k + (2d-1)t < 2^d \leq 2k + 2(2d-1)t - 2. \end{cases} \tag{6}$$

Proof: Let $f(l)$ be the expression on the right hand side of (5). Lemma 1(c) states that $0 \leq l \leq k + (2d-1)t$. It is easy to see that $f(l)$ is monotonic within this range. If $2k+2(2d-1)t-2 \leq 2^d$ then f is increasing and so it is maximized by taking $l = k+(2d-1)t$. In the other case f is decreasing, and so $f(l)$ is maximized by setting $l = 0$. $\square$

Let $E(d, k, t)$ denote the value of this bound for particular values of d, k and t.

Input: (T, U, Z, E, d, k_T): T, U, Z, E are four disjoint sets of processors satisfying $|T| = |U|$ and $|Z| + |E| = (2d+1)t$ for the given integer d. k_T is a bound on the number of faulty processors initially present in T.

Output: (T', U', Z'): A new partition of $T \cup U \cup Z \cup E$ into T', U' and Z'.

Method: Pair up the processors from T and U. Spend one round having every processor from T testing its partner in U. Let $(G, F) = \text{SIFTING}(T, d, k_T + t)$. (We assume that $2^d \| T|$ so that SIFTING can be applied.)

Spend the repair parts of these $2d+1$ rounds repairing all the processors in Z. Perform no action at all with the processors in E.

Let U_u be those processors from U which were paired with some processor from F. Partition $U \setminus U_u$ into U_f and U_g where U_g contains those processors from U whose partner in G diagnosed that they were good, and U_f is the processors whose partner diagnosed that they were faulty. Set $T' = G \cup Z \cup U_g$, $U' = F \cup U_u \cup E$ and $Z' = U_f$.

Algorithm 4: The procedure WINNOWING

4 Ongoing Diagnosis

Our objective is to provide an algorithm to be used by the controller which ensures that at all times there is a known bound on the number of faulty processors in the system, and which never repairs a good processor. The algorithm SIFTING presented in the last section is unable to detect any faulty processors with certainty. Instead it identifies some processors that were good at the start of the application of SIFTING.

This suffices, since we can make each processor a carry out some test T_a before we apply SIFTING. In effect SIFTING validates some of the T_a by showing that the processors which made the tests were good at the start of SIFTING, and so were good when they made their test. If a was good, and T_a said its testee was faulty, we now have a definite diagnosis of a faulty processor.

This suggests the WINNOWING procedure given as Algorithm 4. In a call to WINNOWING the set T of testers is used to test a set U of processors with unknown status. We are given an initial bound k_T on the number of faulty processors in T, so we can apply SIFTING to T. This will let us diagnose most of the processors from U. The processors in Z have been previously identified as faulty and are repaired during the call. Nothing is done with the extra processors in E; they are there in case we do not want to repair the maximum possible number of processors in this call.

It is easy to establish that WINNOWING returns: in T' processors that were good at some point during the call, in U' processors that were not diagnosed, and in Z' processors that were faulty at the start of the call. We deduce that

Lemma 4 *The sets returned by* WINNOWING(T, U, Z, E, d, k_T) *satisfy*

a. All the processors in Z' are faulty.

b. $|U'| \leq 2E(d, k_T + t, t) + (2d+1)t$.

c. At most $(2d+1)t$ of the processors in T' are faulty.

Proof: Part a follows easily from the definition of WINNOWING.

Every processor in U' was either in F, U_u, or in E. (See Algorithm 4 for this notation.) But $|U_u| = |F|$, so $|U'| = 2|F| + |E|$ and the bound for part b follows from theorem 3.

For part c observe that every processor in T' is known either to have been good at the beginning of the application of SIFTING (those processors from G and Y_g) or was repaired while WINNOWING was executing. So if a processor in T' is faulty at the end of the procedure it must have changed status to faulty during the procedure. Since WINNOWING takes $2d+1$ rounds to perform, at most $(2d+1)t$ processors can have done this. □

The ongoing diagnosis is performed by repeatedly calling WINNOWING. Given the sets T', U' and Z' returned by WINNOWING, we have to construct new sets T'', U'', Z'' and E'' suitable for the next call to WINNOWING. Lemma 4 shows that elements of Z' are suitable to be used in Z'' (since they are known to be faulty) and that elements of T' are suitable to be used in T'' (since there is a known bound on the number of faulty nodes in T'). There is no requirement that elements of U'' must satisfy, so we assemble U'' from elements of U' and any left over elements of T' and Z'.

Since we want to repair as many processors as possible, we make Z'' as large as possible, so that $E'' = \emptyset$ if possible. But if $|Z'| < (2d+1)t$ we don't have enough processors that are known to be faulty. In that case we have $E'' \neq \emptyset$. There is no special requirement on the processors in E'' since nothing is done with them anyway. So they can be taken from T' or U' as needed.

All that remains to be shown is that for a suitable choice of d, and for sufficiently large n, there will always be enough elements in T' to fill T''. By lemma 4(c) this means that we can take $k_T = (2d+1)t$, and so $k = (2d+2)t$ for each call to SIFTING.

Now choose d to be the smallest integer such that

$$2^d \geq 2k + 2(2d-1)t - 2 = 8dt + 2t - 2 \tag{7}$$

and so by theorem 3

$$E(d,k,t) = 2^d + 8dt + 2t - 2.$$

By lemma 4 we have

$$|Y_i'| \leq 2 \cdot 2^d + 18dt + 5t - 4. \tag{8}$$

Let us call a processor *concealed* if on the call to WINNOWING it is in fact faulty at the end of the call but is not placed in Z'. Let C' be the set of concealed processors. Since we have a bound on the size of U' and on the number of faulty processors in T', we have a bound on C', namely $|C'| \leq 2 \cdot 2^d + 20dt + 6t - 4$.

On calling WINNOWING the total number of faulty processors can grow by at most $(2d+1)t - |Z|$ processors. Hence $|Z'|$ is less than or equal to the maximum number of concealed processors (present after the previous call) plus $(2d+1)t$. We have

$$|Z_i'| \leq 2 \cdot 2^d + 22dt + 7t - 4. \tag{9}$$

Since $|T'| + |U'| + |Z'| = n$, the upper bounds given above for $|U'|$ and $|Z'|$ give us the needed lower bound on $|T'|$:

$$|T'| \geq n - (4 \cdot 2^d + 42dt + 13t - 8) \tag{10}$$

We need n sufficiently large so that, even if $|T'|$ is minimal, we can still find the partition into T'', U'', Z'', and W''. Since $|T''| = |U''|$ and $|Z''| + |E''| = (2d+1)t$ we want $2|T'| + (2d+1)t \geq n$. So using (10) we need

$$n \geq 8 \cdot 2^d + 82dt + 25t - 16$$

In order to apply SIFTING we need $2^d \Big| |T|$. This increases the minimum n by at most $2 \cdot 2^d$.

We also need to estimate d. For $t > 70$, we can take $d \leq 2\log t$. However we can estimate 2^d more closely by recalling that d is selected so that $2^{d-1} \leq 8dt + 2t - 2 \leq 2^d$. Hence

$$2^d \leq 16dt + 4t - 4 \leq 48t\log t + 4t - 4$$

In the first call to WINNOWING no processors are faulty yet, so we can arbitrarily place processors in T, U and E, with $Z = \emptyset$. In summary we have shown

Theorem 5 *Let $t > 70$ and $n > 562t\log(t) + 65t$. Then we can perform ongoing diagnosis so that at most $118t\log t + 15t$ processors are ever simultaneously faulty.*

Theorem 5 doesn't cover the cases with small t. However the analysis is still mostly accurate. For example, it can be shown that for $t = 1$, if we take $n = 560$ a variant of the algorithm can be run indefinitely, with never more than 131 processors simultaneously faulty.

5 Lower Bound

The aim of this section is to prove that in the ongoing diagnosis model it is impossible for the controller to prevent $t\log t$ processors from being simultaneously faulty. So Theorem 5 is optimal up to a constant factor. Obviously we cannot prove that $t\log t$ processors become faulty if $n < t\log t$. We shall assume that $n \geq 3t\log t$.

We will view the ongoing diagnosis as a series of *phases*. Each phase will consist of a finite number of rounds. In every round the adversary will break t processors. We will show that during every phase at least one of the following statements must be true

1. The controller repairs a processor which in fact had good status.

2. There is some round in which the controller fails to repair t processors.

3. At least $t\log t$ processors were faulty at the start of the phase.

Its is clear that each phase in which statement 1 or 2 is true increases the number of faulty processors in the system by one. So after at most $t \log t$ phases, statement 3 will be true.

Suppose the adversary declares some round to be the first round of a new phase. Let N be the set of processors, partitioned into a set X_1 of initially faulty processors and Y_1 of initially good processors. It is probable that the controller will have some information about the status of some of the processors, which it gained from results of tests carried out in previous phases.

We will allow for this information by supposing that the adversary presents the controller with the status of every processor at the start of each phase. Thus the controller starts the phase knowing the entire state of the system.

5.1 The Designated Strategy

Since the controller is deterministic and the adversary is omniscient, the adversary can precisely predict what the controller will do in response to any particular set of test results. The adversary can foresee how the controller would respond to the following scenario:

- Every faulty processor which performs a test announces that its testee is faulty.
- In round i, define R_i to be the set of processors that the controller would have repaired.
- In round i, corrupt some $S_i \subseteq R_i$, where $S_1 = R_1$ and $|S_i| = \max\{t - 2^{i-1}, 0\}$ for $i > 1$.

We call this scenario the *designated* strategy. The status that a processor would have had in round i if the adversary had used the designated strategy is called the processor's *designated* status.

The adversary will corrupt nodes in such a way that the controller sees *precisely* the same test results as it would have seen if the adversary had been following the designated strategy. So the deterministic controller will repair the same sets R_i as it repairs for the designated strategy.

Define the *true* strategy to be the strategy that the adversary actually follows. A node's status under this strategy is called its *true* status. Let $\rho \in N$ be the first node that the controller repairs which it didn't know to be faulty because it had been given its status at the start of the phase. Let $k+1$ be the round in which it performs this repair. (So it is possible that ρ was faulty at the start of the phase, and the controller repairs it for the *second* time in round k.) We shall show that the adversary can find a true strategy which satisfies the following during the first $\lceil \log t \rceil$ rounds:

- All test results are the same as the results from the designated strategy. We say that a strategy with this property is *consistent* (with the designated strategy).
- The adversary corrupts precisely t processors in each round.
- The adversary never corrupts the processor ρ.

This suffices to show that the adversary has the required properties. If $k \geq \log t$ and statement 2 is false, then 3 must be true, since the controller repaired at least $t \log t$ faulty processors from X_1. But if $k < \log t$, then the controller will have repaired a processor whose true status was good, and so statement 1 will be true.

5.2 The Adversary's Strategy

We will call a processor a *masquerade* in a given round if its true status and its designated status are different. Let M_i denote the set of masquerades in round i. Since the adversary wants to follow a consistent strategy, it wants to have as few masquerades as possible. The next lemma will show that it will greatly simplify matters if the adversary follows a strategy in which the only masquerade which is designated "faulty" (and thus is truly good) is ρ.

Lemma 6 *If ρ is the only masquerade designated "faulty" then for $1 \leq i \leq k-1$ we have*
a. $M_i \subseteq M_{i+1}$
b. If $\rho \in M_i$ then $|M_i| = 2^i$. Otherwise $|M_i| = 2^i - 2$.

Proof: First we prove part a. Let $p \in M_i$ and consider two cases: First suppose p is designated "faulty" in round i, so $p = \rho$. By the definition of ρ the controller will not repair p until round $k+1$, so p will have designated status "faulty" in round $i+1$. Since a true strategy never corrupts ρ, p's true status in round $i+1$ will be good. So $p \in M_{i+1}$.

Otherwise p was designated "good" in round i. But in the designated strategy no processor ever changes state from "good" to "faulty", since $S_i \subseteq R_i$. So p will be designated "good" in round $i+1$. In rounds 1 to k the controller only repairs processors that it knows to be faulty. All of these processors must have designated status "faulty". p has designated status "good" so it will not be repaired. Thus its true status will be faulty in round $i+1$, and so $p \in M_{i+1}$.

For part b we have two cases: First suppose $\rho \notin M_i$. In round 1 there are no masquerades. Round j introduces $|R_j| - |S_j|$ new masquerades that will be present in round $j+1$. So in round i the number of masquerades will be $0 + 2 + 4 + \cdots + 2^{i-1} = 2^i - 2$. But if $\rho \in M_i$ then the number of masquerades is increased by two, since ρ being designated "faulty" forces an additional faulty processor to be designated "good". $\square$

So in moving from round i to round $i+1$ the number of masquerades present increases by at least a factor of two. So for every masquerade who position is forced in round $i+1$, (either because it is ρ becoming a masquerade for the first time, or because it is an existing masquerade in round i) the adversary has a new masquerade to place whose location is undecided. It can place these to conceal the location of the existing masquerades. That is, it will place them to ensure that the true strategy is consistent.

Consider a test that is carried out in round $i+1$. The adversary uses its power to choose which processors to corrupt in round i, to ensure that the test gives a consistent result.

The following table shows every possible combination of the true and designated status of the tester v and the testee u. The entry in the table states the result that the controller would see if the adversary was following the designated strategy. Those entries marked with a $\surd$ are the ones where the adversary can give the required test result. Since the adversary seeks a consistent strategy, it must avoid the combinations marked with an $\times$.

Round $i+1$ test			**Status of testee u**			
			Designated "good"		**Designated "faulty"**	
			truly good	**truly faulty**	**truly good**	**truly faulty**
Status of tester v	**designated "good"**	**truly good**	good √	good ×	faulty ×	faulty √
		truly faulty	good √	good √	faulty √	faulty √
	designated "faulty"	**truly good**	faulty ×	faulty √	faulty ×	faulty √
		truly faulty	faulty √	faulty √	faulty √	faulty √

From the table we can see that if the tester v is truly faulty then the test will be consistent. So if $v \neq \rho$ and $u \in M_i$, it is sufficient for the adversary to corrupt v in round i (if v isn't already faulty). Similarly if $v = \rho$ and it is designated "faulty" in round $i+1$, then we see from the table that it is sufficient for the adversary to corrupt u in round i. (Again the adversary would do nothing if u were already faulty.)

Since the adversary must corrupt t processors, it is possible that the last two paragraphs failed to specify all the processors that it needs to corrupt. In that case it is free to choose the other processors to corrupt in any way that doesn't lead to an inconsistent strategy. If it corrupts processor u in round i, then it must ensure that v is truly faulty in round $i+1$, if v tests u in round $i+1$.

But there is a problem if ρ tests a masquerade (which is necessarily a faulty processor) when ρ is designated "good". The adversary gains nothing by corrupting the testee (indeed it is already faulty), nor can it corrupt the tester since it is ρ. So it can do nothing in this situation.

The way to avoid this problem is to plan ahead so it doesn't occur. If it ever happens that ρ is a masquerade then the problem cannot occur in subsequent rounds since ρ will never be designated "good" in the future. As long as ρ doesn't become a masquerade, the only way that the adversary can be forced to make a processor into a masquerade is if it tests a processor that was made into a masquerade in an earlier round. So by examining the testing graphs we can find all the processors which, if they were made into masquerades, would force ρ to test a masquerade in a later round. As long as we avoid corrupting any of these processors, the problem will not arise. Define the sets $D_k, D_{k-1}, \ldots, D_1$ recursively like this.

- D_k contains ρ and also the processor which tests ρ in round k, if there is one.
- D_i contains all of D_{i+1} and any processors which test a processor from D_{i+1} in round $i+1$.

Observe that $|D_i| \leq 2^{k-i+1}$. We argued that as long as we never make any processor in D_i into a masquerade in round i we will obtain a consistent strategy. A processor is unsuitable to be a new masquerade if its already in M_i, if it is designated "faulty" in round $i+1$, if it is in D_{i+1} or if it is tested by a processor in D_{i+1}. Since we are assuming that statement 3 is not true, fewer than $t \log t$ processors are masquerades or designated "faulty". Since

$$2 \cdot 2^{k-(i+1)+1} = 2^{k-i+1} \leq 2^k \leq t \log t$$

at most $t \log t$ processors are eliminated because of the D_{i+1} condition. Because we are assuming $n \geq 3t \log t$ we will have more than enough processors available to choose the new masquerades.

6 Conclusion

We have shown in Theorem 5 that no matter how large n is, the controller is able to follow a strategy in which $O(t \log t)$ processors become simultaneously faulty. Section 5 shows that there is no strategy that the controller can follow that can prevent $t \log t$ processors from becoming simultaneously faulty. So Theorem 5 is optimal, up to a constant factor.

A few other observations: Since the adversary presented in Section 5 corrupts t processors every round, it will not help the controller if it is told how many processors are corrupted each round. Similarly it will be no help to the controller if it isn't required to *prove* a processor is faulty before it repairs it. (It might seem that this is useful in situations where the adversary doesn't corrupt t processors in some round.)

It is not hard to modify the arguments in Section 5 to apply without the restriction $n \geq 3t \log t$. We find that for smaller n we can prove that eventually at least one ninth of the processors must become simultaneously faulty.

However if is interesting to note that if we consider a changed model in which the controller is permitted to repair $2t$ processors each round, and isn't required to prove that a processor is faulty before repairing it, then an algorithm can be found in which the bound on the total number of faulty processors simultaneously present is linear in t.

References

[1] Richard Beigel, William Hurwood, and Nabil Kahale. Fault diagnosis in a flash. Technical Report 1051, Yale University, 1994.

[2] Richard Beigel, S. Rao Kosaraju, and Gregory F. Sullivan. Locating faults in a constant number of testing rounds. In *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 189–198, 1989.

[3] Richard Beigel, Grigorii Margulis, and Daniel A. Spielman. Fault diagnosis in a small constant number of parallel testing rounds. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1993.

[4] Ronald P. Bianchini, Jr. and Richard Buskens. An adaptive distributed system level-diagnosis algorithm and its implementation. In *Proceedings of 21st Int. Symp. on Fault Tolerant Computing*, pages 222–229, 1991.

[5] Ronald P. Bianchini, Jr., K. Goodwin, and D. S. Nydick. Practical application and implementation of distributed system-level diagnosis theory. In *Proceedings of 20th Int. Symp. on Fault Tolerant Computing*, pages 332–339, June 1990.

[6] S. Louis Hakimi and Kazuo Nakajima. On adaptive system diagnosis. *IEEE Transactions on Computers*, C-33(3):234–240, March 1984.

[7] S. Louis Hakimi, M. Otsuka, Edward F. Schmeichel, and Gregory F. Sullivan. A parallel fault identification algorithm. *Journal of Algorithms*, 11:231–241, 1990.

[8] S. Louis Hakimi and Edward F. Schmeichel. An adaptive algorithm for system level diagnosis. *Journal of Algorithms*, 5:526–530, 1984.

[9] S. H. Hosseini, Jon G. Kuhl, and Sudhakar M. Reddy. A diagnosis algorithm for distributed computing systems with dynamic failure and repair. *IEEE Transactions on Computers*, C-33(3):223–233, March 1984.

[10] Kazuo Nakajima. A new approach to system diagnosis. In *Proc. 19th Annu. Allerton Conf. Commun. Contr. and Comput.*, pages 697–706, September 1981.

[11] Franco Preparata, Gernot Metze, and Robert Chien. On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16(6):848–853, December 1967.