RECONFIGURATION ALGORITHM FOR FAULT-TOLERANT ARRAYS WITH MINIMUM NUMBER OF DANGEROUS PROCESSORS

Chang Chen
An Feng
Tohru Kikuno
Koji Torii

Reprinted from PROCEEDINGS OF FAULT-TOLERANT COMPUTING:
THE TWENTY-FIRST INTERNATIONAL SYMPOSIUM,
Montréal, Canada, June 25-27, 1991

IEEE COMPUTER SOCIETY PRESS REPRINT

IEEE Computer Society
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

Washington, DC • Los Alamitos • Brussels • Tokyo

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE COMPUTER SOCIETY

RECONFIGURATION ALGORITHM FOR FAULT-TOLERANT ARRAYS WITH MINIMUM NUMBER OF DANGEROUS PROCESSORS

Chang Chen, An Feng, Tohru Kikuno and Koji Torii

Department of Information and Computer Sciences
Faculty of Engineering Science, Osaka University
Toyonaka, Osaka 560, Japan
Phone: 81-6-844-1151 (Ext. 4840)
Fax: 81-6-841-9741
e-mail: kikuno@ics.osaka-u.ac.jp

Abstract

This paper discusses a new algorithm for a reconfiguration problem (called the SPA problem) for $n \times n$ ordinary processors using spare processors. The SPA problem, originally presented by Melhem(1989), is to find an assignment of spare processors to faulty processors that minimizes the number of dangerous processors. Here, dangerous processors are non-faulty processors for which there remains no longer any spare processor to be assigned if one more fault occurs. In this paper, we develop an $O(n^2)$ algorithm for a basic SPA problem where $2n$ spare processors are provided. Then, we define an extension of the SPA problem and clarify several interesting properties to solve them. In the extension, the spare processors are assumed to become faulty. Thus, it is expected that ideas presented in this paper greatly contribute to the development of reconfiguration algorithms for other fault-tolerant systems.

1. Introduction

Recently, digital system applications that demand high reliability and continuous operation are strongly desired. Since it is impossible to guarantee that any given component of the system will never fail, the system needs to be designed to tolerate failures of components. The development of such fault-tolerant systems is extensively surveyed in (1), (13).

From the viewpoint of computing architecture, it is very reasonable to provide a system with spare processors and to reconfigure the system when faults occur. Many reconfiguration algorithms have already been developed for the hypercube architecture[(3)(4)]. Especially for array architectures[(5)(7)-(12)(14)(15)], numerous researchers have focused on developing efficient reconfiguration algorithms. In typical algorithms [(1)(7)(9)] for arrays, when a faulty processor is detected, an entire row or column containing the faulty processor is disconnected and a spare row or column is used to replace it. The goal is to utilize the minimum number of spare rows and/or columns.

The problem (called the SPA problem) is stated as follows[(11)]: the array AP consists of n^2 ordinary processors and $2n$ spare processors. Some of ordinary processors are specified faulty. Then, we should find a spare processor assignment (that is a reconfiguration) to faulty processors that minimizes the number of dangerous processors. Here, dangerous processors are ordinary processors for which there remains no longer any spare processor to be assigned if one more fault occurs in the future.

In real time applications, continuous operation of the system must be assured. For this purpose, the algorithms should take notice of not only the faulty processors but also of the processors that may become faulty in the future. However, most of the proposed algorithms deal with only the given faulty processors. Melhem [(11)] recently proposed a new algorithm which tries to find reconfigurations with respect to all possible faults that may happen in a given array architecture. Unfortunately, the proposed algorithm was inefficient from the viewpoint of time complexity.

In this paper, we solve the SPA problem by presenting an efficient algorithm. The proposed algorithm finds an optimal assignment of spare processors to faulty processors that minimizes the number of dangerous processors. Then, we try to extend the model on which the SPA problem is defined, and present an extended SPA problem. In the extension, the spare processors may become faulty.

This paper is organized as follows: Section 2 gives the definitions of models (Models 1 and 2) for reconfiguration arrays. Section 3 gives the definitions of the SPA problem on Models 1 and 2, and compares the SPA problem with other related problems. Then, Section 4

CH2985-0/91/0000/0452/$01.00

describes necessary and sufficient conditions and an efficient algorithm to solve the SPA problem on Model 1. Section 5 presents necessary and sufficient conditions to solve the SPA problem on Model 2. Finally, Section 6 summarizes the main results and future research work.

2. Reconfigurable Array Model

We introduce two kinds of reconfigurable array models. Each model is specified by giving an array AP, a set of faulty processors F and a cover τ.

2.1 Basic model (Model 1)

(A) Array processor AP_1

In Model 1, an array processor AP_1 is defined to be a 3-tuple $AP_1 = (OP_1(n), S_1(n), L_1(n))$, where $OP_1(n) = \{ c_{ij} \mid 1 \le i, j \le n \}$ is a set of $n \times n$ ordinary processors, $S_1(n) = \{ a_j, b_i \mid 1 \le j, i \le n \}$ is a set of $2n$ spare processors with $OP_1(n) \cap S_1(n) = \phi$, and $L_1(n) = \{(a_j, c_{ij}), (b_i, c_{ij}) \mid 1 \le i, j \le n \}$ is a set of $2n^2$ links.

Example 1 Figure 1 shows an array processor AP_1 for $n = 6$, $AP = (OP_1(6), S_1(6), L_1(6))$. In the figure, the ordinary processors c_{ij}'s are represented by circles and the spare processors a_j's and b_i's are represented by rectangles, respectively. Only the connections between a_3 and c_{i3}'s and the connections between b_4 and c_{4j}'s are illustrated by arcs in Figure 1.

(B) Cover τ_1 and set F_1

When an ordinary processor, say c_{53}, fails, one of the spare processors takes over the task. In Model 1, b_5 and a_3 have connections to c_{53}. Thus, either b_5 or a_3 is assigned to take over c_{53}. More generally, for a set of faulty ordinary processors F_1, a set of the spare processors are assigned to realize array reconfiguration.

A set of faulty processors F_1 is specified to be a subset of the set of ordinary processors $OP_1(n)$. Thus $F_1 \subseteq OP_1(n)$. Then an injection $\tau_1 : F_1 \to S_1(n)$ satisfying the conditions (1) and (2) is called a cover τ_1 for the given F_1.

(1) For any $c_{ij} \in F_1$, if $\tau(c_{ij}) = a_k$ then $k = j$, and if $\tau(c_{ij}) = b_k$ then $k = i$.

(2) For any $a_j, b_i \in S_1(n)$, at most one faulty processor c_{ij} is assigned to a_j or b_i.

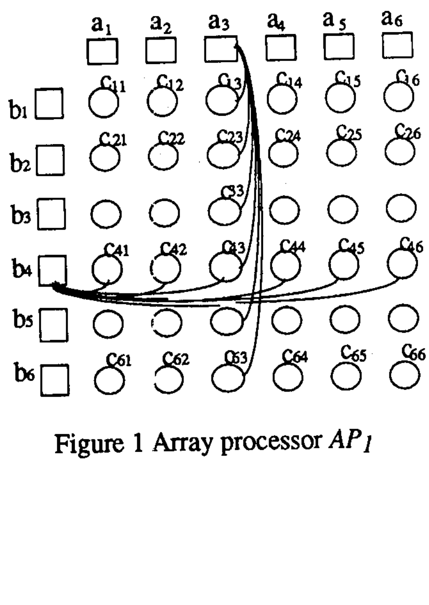


Figure 1 Array processor AP_1

Example 2 Consider the array processor AP in Example 1. Assume that $F_1 = \{ c_{22}, c_{24}, c_{31}, c_{32}, c_{34}, c_{46}, c_{53}, c_{55} \}$, and each faulty processor in F_1 is shown by a cross in Figure 2. Then, an example of τ_1 is shown by the arcs in Figure 2.

2.2 Extended model (Model 2)

In Model 1, spare processors are assured to be non-faulty. Now, we will relax this condition and extend Model 1 by allowing any of spare processors to be faulty.

Thus, in Model 2 for an array processor $AP_2 = (OP_2(n), S_2(n), L_2(n))$ with $OP_2(n) = OP_1(n)$, $S_2(n) = S_1(n)$, $L_2(n) = L_1(n)$, a set of faulty processors F_2 is defined to be a subset of the union of two sets $OP_2(n)$ and $S_2(n)$. That is to say, $F_2 \subseteq OP_2(n) \cup S_2(n)$. Then a cover τ_2 for a given F_2 is an injection $\tau_2 : F_2 \to S_2(n) - F_2$ that satisfies conditions (1) and (2) mentioned in 2.1

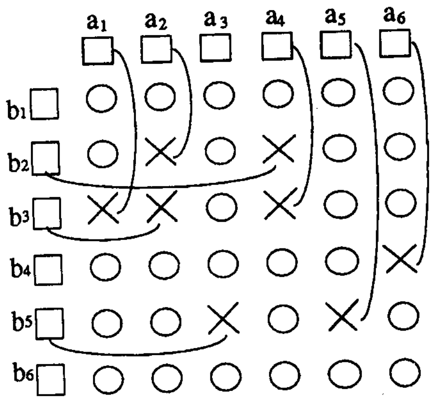


Figure 2 Cover τ_1

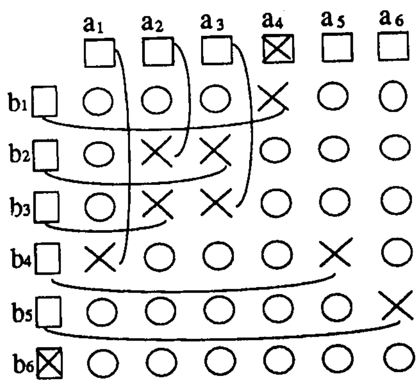


Figure 3 Cover τ_2

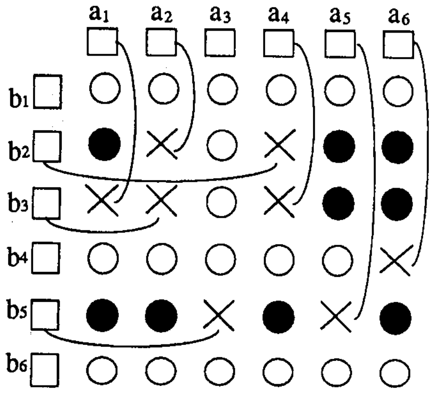


Figure 4 Dangerous processors

(note that F_1 and $S_1(n)$ in the conditions must be replaced by F_2 and $S_2(n)$, respectively).

Example 3 Consider AP in Figure 1 again. Assume that $F_2= \{a_4, b_6, c_{14}, c_{22}, c_{23}, c_{32}, c_{33}, c_{41}, c_{45}, c_{56}\}$. Thus, two spare processors a_4 and b_6 are assumed to be faulty, which is shown by a cross in a rectangle in Figure 3. Then, an example of τ_2 is shown by the arcs in Figure 3.

3. Spare Processor Assignment Problem

3.1 Definition of problem

For the given cover τ_μ and F_μ (μ=1, 2), any ordinary processor d satisfying the conditions (3) and (4) is called a *dangerous processor*.

(3) $d \in OP_\mu(n) - (F_\mu \cap OP_\mu(n))$

(4) if $d=c_{ij}$, then all of spare processors b_i and a_j are assigned to faulty processors with respect to τ_μ or are faulty spare processors.

We represent a set of dangerous processors with respect to τ_μ by $D(\tau_\mu) = \{d\}$, and the number of elements in $D(\tau_\mu)$ by $\#D(\tau_\mu)$.

The definition of dangerous processors implies that even if an ordinary processor $c_{ij} \notin D(\tau_\mu)$ fails afterwards, one of b_i and a_j is surely assigned to c_{ij}, and thus the system can survive. On the other hand, if an ordinary processor $c_{ij} \in D(\tau_\mu)$ fails afterwards, then there remain no spare processors to take over c_{ij}.

Example 4 Consider the cover τ_1 given in Example 2. Then $D = \{c_{21}, c_{25}, c_{26}, c_{35}, c_{36}, c_{51}, c_{52}, c_{54}, c_{56}\}$. These nine dangerous processors are shown by dark circles in Figure 4.

The spare processor assignment problem (shortly **the SPA problem**) on Model μ (μ = 1, 2) is defined as follows: When an array processor AP_μ and a set of faulty processors F_μ are given as the input of the problem, we should find a cover $\tau_\mu{}^*$ for F_μ such that $\#D(\tau_\mu{}^*)$ is minimum among all possible covers τ_μ's for F_μ.

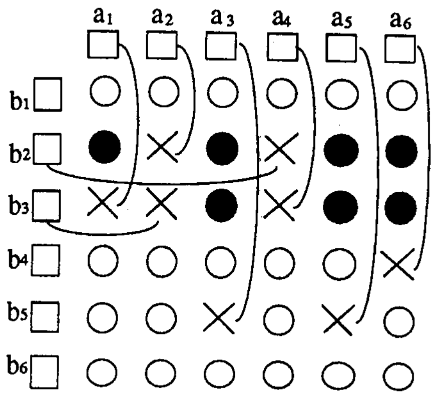


Figure 5 Optimal cover τ_1

Example 5 Consider, as an input of the SPA problem on Model 1, the array processor AP_1 in Figure 1 and the set of faulty processors F_1 in Figure 2. Then, a cover $\tau_1{}^*$ shown in Figure 5 realizes the minimum number of dangerous processors. In this case $\#D(\tau_1{}^*)$=7, that is less than $\#D(\tau_1)$=9 in Figure 4.

3.2 Comparison with other related problems

The SPA problem on Model μ is closely related to the spare allocation (reconfiguration) problem in References (1), (9). The spare allocation problem can be modeled as a rectangular array with $M \times N$ cells with SR spare rows and SC spare columns. The reconfiguration algorithm should select the minimum number of spare rows and/or columns that cover all the faulty cells[1].

There exist three differences (a)-(c) between the definitions of the spare allocation problem and the SPA problem.

(a) In the SPA problem on Model μ (μ = 1, 2), spare processors $S_\mu(n)$ are $1 \times n$ and $n \times 1$ arrays. However, in the spare allocation problem spare cells are placed in $SR \times N$ columns and $M \times SC$ rows, respectively.

(b) In the SPA problem on model μ, a single spare processor is individually assigned to a faulty processor c_{ij}. On the other hand, in the spare allocation problem, either a single row or a single column of spare cells is collectively allocated to faulty processors.

(c) In ordinary reconfiguration problems including the spare allocation problem, the goal is to relieve only the faulty processors. But the SPA problem is to minimize the number of dangerous processors that will arise in the system in the future.

Kuo and Fuchs have already shown that the complexity of optimal spare allocation problem is NP-complete, and presented good heuristic algorithms to solve the problem[9].

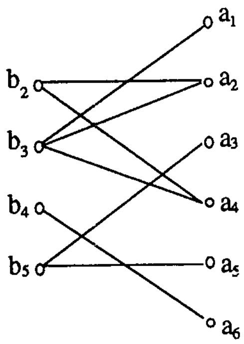
4. SPA Problem on Model 1

4.1 Existence of cover τ_1

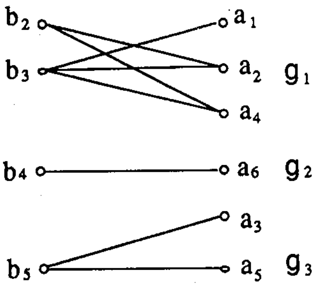
In this subsection, we will present a necessary and sufficient condition that assures the existence of the cover τ_1 for a given F_1. For the given AP_1 and F_1, we construct a bipartite graph[6] $G_{AP1} = (V,E)$ such that $V=V_a \cup V_b$, where $V_a = \{a_j \mid a_j \in S_1(n)$ and $\exists k\ (1 \le k \le n)\ [c_{kj} \in F_1]\}$, $V_b = \{b_i \mid b_i \in S_1(n)$ and $\exists k\ (1 \le k \le n)\ [c_{ik} \in F_1]\}$ and $E=\{(b_i,a_j) \mid c_{ij} \in F_1\}$. We call nodes $a_j \in V_a$ and $b_i \in V_b$ an *a-node* and *b-node*, respectively. Let $C_{AP1} = \{g_1, g_2, \ldots, g_s\}$ be a set of connected components[6] of the bipartite graph G_{AP1}.

Example 6 Consider the array processor AP_1 and faulty processors F_1 shown in Figure 2. A bipartite graph G_{AP1} shown in Figure 6(a) is constructed. There exist three connected components $C_{AP1} = \{g_1, g_2, g_3\}$ which are shown in Figure 6(b).

For $G_{AP1}=(V_a \cup V_b, E)$, we define a function $\rho: E \rightarrow V_a \cup V_b$, called a *node assignment*, as follows:
(1) For any edge $e = (b_i, a_j)$ in E, the value $\rho(e)$ is either b_i or a_j.
(2) For any pair of edges e_1, e_2 in E, if $e_1 \neq e_2$ then



(a) Bipartite graph G_{AP1}



(b) Connected components of G_{AP1}

Figure 6 Bipartite graph G_{AP1} and its components

$\rho(e_1) \neq \rho(e_2)$ (that is, ρ is an injection).

Lemma 1 Consider an array processor AP_1 and a set of faulty processors F_1. Then a cover τ_1 exists for F_1 if and only if a node assignment ρ exists for a bipartite graph G_{AP1}.

Theorem 1 Consider an array processor AP_1 and a set of faulty processor F_1. Then, a cover τ_1 exists for F_1 if and only if for each connected component $g_i=(V_i,E_i)$ in C_{AP1} either $|V_i| = |E_i|$ or $|V_i| = |E_i| + 1$ holds.

(Proof) Only if ... Assume that a cover τ_1 exists. Then we can conclude that $|V_i| \ge |E_i|$ holds. Additionally, for each connected component $g_i=(V_i,E_i)$, the relation $|E_i| + 1 \ge |V_i|$ holds. These two facts imply that $|V_i| = |E_i| + 1$ or $|V_i| = |E_i|$.

If ...Assume that for each $g_i=(V_i,E_i)$, either $|V_i| = |E_i|+1$ or $|V_i| = |E_i|$ holds. We can construct a node assignment ρ_i for each connected component.

(Case 1) $|V_i| = |E_i|+1$ ··· In this case, g_i is a tree. Select any node z in g_i as the root node. Assume that (x,y) is any edge in g_i, and the length of the path between the root and x is less than the length of the path between the root and y. Then, we define $\rho_i((x,y))=y$.

(Case 2) $|V_i| = |E_i|$ ··· In this case, $g_i = (V_i,E_i)$ contains exactly one loop. So we construct a rooted spanning tree[6] $T_i=(V_i,E_i')$ of g_i with root z. Assume that (z,w) is an edge such that $(z,w) \in E_i - E_i'$. Then, for each edge (x,y) in T_i, we define $\rho_i((x,y))=y$ in the same way as in Case 1. Finally, $\rho_i((z,w))=z$.

Then we can easily compose a node assignment ρ from $\rho_1, \rho_2, \ldots, \rho_s$, since $E=E_1 \cup E_2 \cup \ldots \cup E_s$ and $E_i \cap E_j = \phi\ (i \neq j)$. From Lemma 1 we can conclude that a cover τ_1 exists. ∎

Example 7 Consider the set of connected components $C_{AP1} = \{g_1, g_2, g_3\}$ in Figure 6(b). For $i = 2,3$, $|V_i| = |E_i| + 1$ holds. For $i = 1$, $|V_1| = |E_1|$ holds. Thus, at least one cover τ_1 exists. Examples of such covers are shown in Figures 4 and 5.

4.2 Necessary and sufficient condition

Assume that $C_{AP1} = \{g_1, g_2, \ldots, g_s\}$ is rearranged into $\overline{C_{AP1}} = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_k : g_{k+1}, \ldots, g_s\}$ as follows: each $\hat{g}_i = (V_i, E_i)$ $(1 \le i \le k)$ satisfies $|V_i| = |E_i|+1$, and each $g_i = (V_i, E_i)$ $(k+1 \le i \le s)$ satisfies $|V_i| = |E_i|$.

Lemma 2 Define $IA = \{x_1, x_2, \ldots, x_k \mid x_i \in V\}$ for a bipartite graph $G_{AP1} = (V,E)$. Then there exists a node assignment $\rho: E \rightarrow V$ for $G_{AP1} = (V,E)$ with $\rho(E) = V$ -

IA if and only if x_i in IA is a node in $\hat{g}_i \in \overline{C_{AP1}}$ $(1 \le i \le k)$.

Lemma 3 Let $\rho: E \to V$ be the node assignment for $G_{AP1} = (V,E)$, $V = V_a \cup V_b$ that satisfies $\rho(E) = V - IA$ mentioned in Lemma 2.Next, define a function $p_1(IA)$ as follows:

$$p_1(IA) = (|V_a| - \alpha) * (|V_b| - \beta) - |E| + \sum_{i=1}^{k} deg(x_i),$$

where $\alpha = |V_a \cap IA|$, $\beta = |V_b \cap IA|$ and $deg(x_i)$ is the degree of the node x_i. Then the value $p_1(IA)$ represents the number of dangerous processors with respect to a cover τ_1 that corresponds to ρ.
(Proof) See Reference (5). ∎

Example 8 Consider the connected components $\overline{C_{AP1}} = \{g_2, g_3 : g_1\}$ in Figure 6(b) and $IA = \{a_3, b_4\}$. Then, we can compute $p_1(IA) = (4-1)*(6-1)-8+(1+1) = 9$. On the other hand, the cover τ_1, in Figure 2 is obtained by applying the procedure mentioned in Proof of Theorem 1. Then, the number of dangerous processors, which are shown with dark circles in Figure 4, is also 9.

We define IA_{min} to be an $IA = \{x_1, x_2, \ldots, x_k\}$, for which the value of $p_1(IA)$ is minimum. For each connected component $\hat{g}_i = (V_i, E_i)$ $(1 \le i \le k)$ in $\overline{C_{AP1}}$, we define the minimum degrees of a-nodes and b-nodes as follows:

$$min\_deg(\mathrm{a},i) = \min\{ deg(x) \mid x \in V_a \cap V_i \},$$
$$min\_deg(\mathrm{b},i) = \min\{ deg(x) \mid x \in V_b \cap V_i \}.$$

Proposition 1 Each node x_i $(1 \le i \le k)$ in IA_{min} satisfies the following condition C1.

(C1) $$deg(x_i) = \begin{cases} min\_deg(\mathrm{a},i) & \text{if } x_i \in V_a \cap V_i \\ min\_deg(\mathrm{b},i) & \text{if } x_i \in V_b \cap V_i. \end{cases}$$

(Proof) Let $D = (|V_a| - \alpha)(|V_b| - \beta) - |E|$. Then, by the definition, $p_1(IA) = D + \sum_{i=1}^{k} deg(x_i)$. Since D is a constant and $p_1(IA)$ is minimum, the condition C1 must be held for each x_i in IA_{min}. ∎

For each connected component $\hat{g}_i = (V_i, E_i)$ $(1 \le i \le k)$ in $\overline{C_{AP1}}$, we define the difference $\Delta(i) = min\_deg(\mathrm{a},i) - min\_deg(\mathrm{b},i)$. Let $Ind(\mathrm{a}) = \{ i \mid x \in IA \cap V_a$ and $x \in V_i$ for $\hat{g}_i \}$ and $Ind(\mathrm{b}) = \{ i \mid x \in IA \cap V_b$ and $x \in V_i$ for $\hat{g}_i \}$ be set of indices of nodes in IA. Note that $Ind(\mathrm{a}) \cup Ind(\mathrm{b}) = \{1,2,\ldots,k\}$ for $\overline{C_{AP1}}$.

Proposition 2 Let $Ind(\mathrm{a}) = \{ i \mid x \in IA_{min} \cap V_a$ and $x \in V_i$ for $\hat{g}_i \}$ and $Ind(\mathrm{b}) = \{ i \mid x \in IA_{min} \cap V_b$ and $x \in V_i$ for $\hat{g}_i \}$ for a given IA_{min}. Then, the set IA_{min} satisfies the following condition C2 for $\Delta(i)$.

(C2) For any l, m with $l \in Ind(\mathrm{a})$ and $m \in Ind(\mathrm{b})$,
$$\Delta(l) \le \Delta(m).$$

(Proof) See Reference (5). ∎

Consider a sequence $(\Delta(1), \Delta(2), \ldots, \Delta(k))$ and sort it in a nondecreasing order of values $\Delta(i)$'s. Let $\bar{\Delta} = (\Delta(s_1), \Delta(s_2), \ldots, \Delta(s_k))$, $1 \le s_1, s_2, \ldots, s_k \le k$, be the resultant sequence, for which $\Delta(s_1) \le \Delta(s_2) \le \ldots \le \Delta(s_k)$ holds. Let $IA\gamma$ be $IA = \{x_1, x_2, \ldots, x_k\}$ where x_i is a node in $\hat{g}_i$ $(1 \le i \le k)$ such that the number of a-nodes in IA is γ.

Proposition 3 Let γ be the number of a-nodes in a given IA_{min}, and let $min\_p_1(\gamma) = (|V_a| - \gamma)(|V_b| - (k - \gamma)) - |E| + \sum_{i=1}^{k} min\_deg(\mathrm{b},i) + \sum_{i=1}^{\gamma} \Delta(s_i)$ for the sequence $\bar{\Delta} = (\Delta(s_1), \Delta(s_2), \ldots, \Delta(s_k))$. Then the value $p_1(IA_{min})$ must satisfy the following condition.

(C3) $p_1(IA_{min}) = \min\{ min\_p_1(\gamma) \mid 0 \le \gamma \le k \}$.

(Proof) By the definitions of p_1, $Ind(\mathrm{a})$ and $Ind(\mathrm{b})$,
$$p_1(IA_\gamma) = D_\gamma + \sum_{i \in Ind(\mathrm{a})} deg(x_i) + \sum_{i \in Ind(\mathrm{b})} deg(x_i)$$
is derived where $D_\gamma = (|V_a| - \gamma)(|V_b| - (k-\gamma)) - |E|$. Then,
$$p_1(IA_\gamma) \ge D_\gamma + \sum_{i=1}^{k} min\_deg(\mathrm{b},i) + \sum_{i \in Ind(\mathrm{a})} \Delta(i).$$
By the property of $\bar{\Delta}$,
$$p_1(IA_\gamma) \ge D_\gamma + \sum_{i=1}^{k} min\_deg(\mathrm{b},i) + \sum_{i=1}^{\gamma} \Delta(s_i) = min\_p_1(\gamma).$$

If we select a special IA_γ satisfying the conditions C1 and C2, then $p_1(IA_\gamma) = min\_p_1(\gamma)$ is derived.

Next, assume that the minimum value of $min\_p_1(\gamma)$'s is $min\_p_1(m)$. Then by definition of IA_{min}, the number of a-nodes in IA_{min} must be m. Thus, $p_1(IA_{min}) = \min\{ min\_p_1(\gamma) \mid 0 \le \gamma \le k \}$ is derived. ∎

Theorem 2 Let IA be a set $IA = (x_1, x_2, \ldots, x_k)$ where x_i $(1 \le i \le k)$ is a node in $\hat{g}_i$ for $\overline{C_{AP1}} = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_k : g_{k+1}, \ldots, g_s\}$. Then $IA = IA_{min}$ if and only if all the conditions C1, C2 and C3 are satisfied by the set IA.

(Proof) Only if ... This part is clearly proved by Propositions 1, 2 and 3.

If ... Assume that IA satisfies the conditions C1, C2 and C3. Let m be the number of a-nodes in IA. On the other hand, let IA' be any sequence $(x'_1, x'_2, \ldots, x'_k)$ such that x'_i is a node in $\hat{g}_i$ $(1 \le i \le k)$. Let $Ind'(\mathrm{a}) = \{ i \mid x \in IA' \cap V_a$ and $x \in V_i$ for $\hat{g}_i \}$ and $Ind'(\mathrm{b}) = \{ i \mid x \in IA' \cap V_b$ and $x \in V_i$ for $\hat{g}_i \}$. Let m' be the number of a-nodes in IA'.

Then, by the definition of min_deg and $\Delta(i)$ and $\bar{\Delta}$,

$$p_1(IA') = D_{m'} + \sum_{i \in Ind'(a)} deg(x'_i) + \sum_{i \in Ind'(b)} deg(x'_i)$$
$$\geq D_{m'} + \sum_{i \in Ind'(a)} min\_deg(a,i) + \sum_{i \in Ind'(b)} min\_deg(b,i)$$
$$= D_{m'} + \sum_{i=1}^{k} min\_deg(b,i) + \sum_{i \in Ind'(a)} \Delta(i)$$
$$\geq D_{m'} + \sum_{i=1}^{k} min\_deg(b,i) + \sum_{i=1}^{m'} \Delta(s_i) = min\_p_1(m').$$

Furthermore, since IA satisfies the condition C3, $min\_p_1(m') \geq min\_p_1(m)$ is derived. Next, consider the derivation of $p_1(IA\gamma) \geq min\_p_1(\gamma)$ in the proof of Proposition 3. In this case, since IA satisfies the conditions C1 and C2, two inequalities can be replaced by equalities. Thus we can get $p_1(IA) = min\_p_1(m)$. As the result, $p_1(IA') \geq p_1(IA)$ is derived for any IA'. ■

Example 9 Consider $\overline{C_{AP1}} = \{g_2, g_3 : g_1\}$ in Example 8, again. Then min_deg(a,2)=1, min_deg(b,2)=1, min_deg(a,3)=1, min_deg(b,3)=2, and thus we get $\Delta(2) = 0$, $\Delta(3) = -1$, and $\bar{\Delta} = (-1, 0)$. Now, we compute $min\_p_1(0)$=6*(4-2)-8+1+2=7, $min\_p_1(1)$=(6-1)*(4-1)-8+1+2+(-1)=9, and $min\_p_1(2)$=(6-2)*4-8+1+2-1=10. Finally, we get m=0, $IAmin = \{b_4, b_5\}$.

4.3 Algorithm for SPA problem on Model 1

Based on the conditions C1, C2 and C3 in Theorem 2, an algorithm, called Algorithm SPA-1, for the SPA problem can be developed.

[Algorithm SPA-1]

Step 1 (Checking the existence of τ_1)

1.1 Construct the bipartite graph G_{AP1} and the set $C_{AP1} = \{g_i = (V_i, E_i)\}$.

1.2 Check whether each $g_i \in C_{AP1}$ satisfies the condition ($|V_i| = |E_i|+1$ or $|V_i| = |E_i|$) or not. If it is satisfied, rearrange into $\overline{C_{AP1}} = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_k : g_{k+1}, \ldots, g_s\}$.

Step 2 (Computation of IA_{min})

2.1 Get the values $min\_deg(a,i)$, $min\_deg(b,i)$ for each $\hat{g}_i$ ($1 \leq i \leq k$).

2.2 Get the value $\Delta(i)$ for each $\hat{g}_i$ ($1 \leq i \leq k$) and sort the sequence ($\Delta(1)$, $\Delta(2)$, ..., $\Delta(k)$) in nondecreasing order. Let $\bar{\Delta} = (\Delta(s_1), \Delta(s_2), \ldots, \Delta(s_k))$ be the resultant sequence.

2.3 Get the set $\{min\_p_1(\gamma) \mid 0 \leq \gamma \leq k\}$, and find the value γ giving the minimum value of $min\_p_1(\gamma)$. Let m be the resultant value.

2.4 Finally, let $IA_{min} = (x_{s1}, x_{s2}, \ldots, x_{sm}, x_{sm+1}, \ldots, x_{sk})$ such that each x_{si} ($1 \leq i \leq m$) is a-node in $\hat{g}_{si}$ with $deg(x_{si}) = min\_deg(a, s_i)$, and each x_{si} ($m+1 \leq i \leq k$) is b-node in $\hat{g}_{si}$ with $deg(x_{si}) = min\_deg(b, s_i)$.

Step 3 (Construction of τ_1)

3.1 Determine an assignment ρ_i ($1 \leq i \leq s$) as follows:

(Case 1) For each $\hat{g}_{si}$ ($1 \leq i \leq m$), define ρ_{si} by the procedure of Case 1 in Theorem 1's proof with selecting a-node x_{si} as the root. Then, for each $\hat{g}_{si}$ ($m+1 \leq i \leq k$), define ρ_{si} by the same procedure with selecting b-node x_{si} as the root.

(Case 2) For each g_i ($k+1 \leq i \leq s$), define ρ_i by the procedure of Case 2 in Theorem 1's proof.

3.2 Compose an assignment ρ for G_{AP1} from $\rho_1, \rho_2, \ldots, \rho_s$.

3.3 Define a cover τ_1 by applying Lemma 2.

Example 10 Consider AP_1 in Figure 1 and F_1 in Figure 2 as an input of the SPA problem. Then at Step 1, we construct a bipartite graph G_{AP1} shown in Figure 6, and get $\overline{C_{AP1}} = \{g_2, g_3 : g_1\}$. At Step 2, we get IA_{min} described in Example 9. Finally at Step 3, we can construct an optimal cover τ_1 shown in Figure 5.

Theorem 3 Algorithm SPA-1 solves a given SPA problem on Model 1 in $O(n^2)$ time, where n^2 is the number of ordinary processors.

(*Proof*) At first, we explain the correctness of Algorithm SPA-1. Step 1 is based on the necessary and sufficient condition mentioned in Theorem 1. If the condition is not satisfied, there exists no solution of the given SPA problem. The set C_{AP1} can be easily constructed by applying a depth-first search algorithm[2].

Step 2 is the most essential part of Algorithm SPA-1, and utilizes the conditions C1, C2 and C3 in Theorem 2. At Substep 2.2, a sorting algorithm, for instance heap sort algorithm[2], is applied to the sequence ($\Delta(1)$, $\Delta(2)$, ..., $\Delta(k)$) to get the sequence $\bar{\Delta}$, then at Substep 2.3, we compute $min\_p_1(\gamma)$ ($0 \leq \gamma \leq k$).

Step 3, based on the result of Theorem 1, gives a concrete solution for the given SPA problem. At Substep 3.1, we must construct a spanning tree for each connected component g_i ($k+1 \leq i \leq s$). The spanning tree is also found by applying a depth-first search algorithm[2].

As mentioned above, Steps 1, 2 and 3 are based on the results in Theorems 1 and 2. Theorem 2 proves that IA_{min} optimally satisfies the conditions C1,C2 and C3. Thus our algorithm SPA-1 is correct.

Next, we briefly mention the time complexity of Algorithm SPA-1. Since the size of F_1 is generally $O(n^2)$, the number of edges in the bipartite graph becomes $O(n^2)$. Thus, the depth-first search algorithm requires $O(n^2)$ time at Substep 1.1 with respect to the worst-case time complexity. Furthermore, when a cover τ_1 exists, the number of connected component in C_{AP1} is O(n). Thus, both k and s have the value of O(n). The sorting at Substep 2.2 takes O(n log$n$) time. Each of Substeps 2.1, 2.2 and 2.4 takes O(n) time.

The size of F_1 is bounded by $2n$ when a cover τ_1 exists for the SPA problem. The depth-first search algorithm requires O(n) time at Substep 3.1. Substeps 3.2 and 3.3 take O(n) time also. Thus, Algorithm SPA-1 takes O(n^2) time with respect to the worst-case time complexity. ■

5. SPA Problem on Model 2

5.1 Existence of cover τ_2

For the given AP_2 and F_2, we construct a bipartite graph $G_{AP2} = (V,E)$, $V = V_a \cup V_b$ in the same way as mentioned in Subsection 4.1. Let $C_{AP2} = \{g_1, g_2, \ldots, g_s\}$, $g_i = (V_i, E_i)$ $(1 \le i \le s)$ be a set of connected components of the bipartite graph G_{AP2}. Then, we divide V_i $(1 \le i \le s)$ into two disjoint subsets $V_i = V_{if} \cup V_{in}$, where $V_{if} = \{b_i, a_j \mid b_i, a_j \in F_2\}$ and $V_{in} = \{b_i, a_j \mid b_i \notin F_2, a_j \notin F_2\}$. Let $V_f = V_{1f} \cup V_{2f} \cup \ldots \cup V_{sf}$ and $V_n = V_{1n} \cup V_{2n} \cup \ldots \cup V_{sn}$.

Next, we define a node assignment where $\rho: E \rightarrow V_n$ as follows:

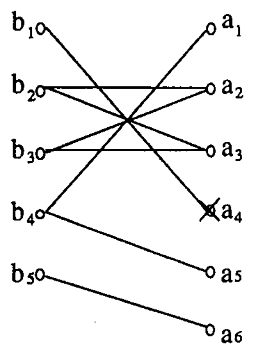
(1) For any edge $e = (b_i, a_j)$ in E, the value $\rho(e)$ is either b_i or a_j. (Note that $\rho(e) \notin F_2$ is assured.)

(2) For any pair of edges e_1, e_2 in E, if $e_1 \neq e_2$ then $\rho(e_1) \neq \rho(e_2)$.

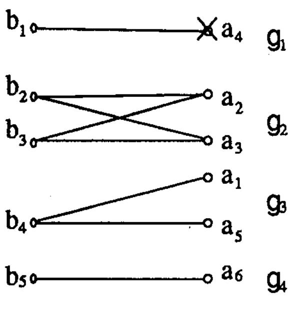
Then a cover τ_2 exists for F_2 if and only if a node assignment ρ exists for a bipartite graph G_{AP2}.

Theorem 4 Consider an array processor AP_2 and a set of faulty processors F_2. Then a cover τ_2 exists for F_2 if and only if each connected component $g_i = (V_i, E_i)$ in C_{AP2} satisfies either $|V_{in}| = |E_i|$ or $|V_{in}| = |E_i| + 1$.

Example 11 Consider AP_2 and F_2 shown in Figure 3. Then, a bipartite graph G_{AP2} shown in Figure 7(a) is constructed. There exist four connected components $C_{AP2} = \{g_1, g_2, g_3, g_4\}$ shown in Figure 7(b). In the figure, b_i or a_j with a cross represents a faulty spare processor. Thus, $g_1 = (V_1, E_1)$, $V_1 = \{b_1, a_4\}$, $E_1 = \{(b_1, a_4)\}$ produces $V_{1f} = \{a_4\}$ and $V_{1n} = \{b_1\}$. On the other hand, $g_3 = (V_3, E_3)$, $V_3 = \{b_4, a_1, a_5\}$, $E_3 = \{(b_4, a_1), (b_4, a_5)\}$ produces $V_{3f} = \phi$, $V_{3n} = V_3$. Clearly, for $i = 1, 2$, $|V_{in}| = |E_i|$ holds and for $i = 3, 4$, $|V_{in}| = |E_i| + 1$ holds. Thus, at least one cover τ_2 (shown in Figure 3) exists.



(a) Bipartite graph G_{AP2}



(b) Connected components of G_{AP2}

Figure 7 Bipartite graph G_{AP2} and its components

5.2 Necessary and sufficient condition

Assume that $C_{AP2} = \{g_1, g_2, \ldots, g_s\}$ is rearranged into $\overline{C_{AP2}} = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_k : g_{k+1}, \ldots, g_s\}$ as follows: For $\hat{g}_i = (V_i, E_i)$ $(1 \le i \le k)$, $|V_{in}| = |E_i| + 1$ hold, for $g_i = (V_i, E_i)$ $(k+1 \le i \le s)$, $|V_{in}| = |E_i|$ holds.

Additionally, we define two sets $T_a = \{a_j \mid a_j \in S_2(n) \cap F_2$ and $\forall k\ (1 \le k \le n)\ [c_{kj} \notin F_2]\}$, $T_b = \{b_i \mid b_i \in S_2(n) \cap F_2$ and $\forall k\ (1 \le k \le n)\ [c_{ik} \notin F_2]\}$.

Lemma 4 Define $IA = \{x_1, x_2, \ldots, x_k \mid x_i \in V\}$ for a bipartite graph $G_{AP2} = (V,E)$. Then a node assignment $\rho: E \rightarrow V_n$ for $G_{AP2} = (V,E)$ with $\rho(E) = V - IA$ exists if and only if x_i in IA is a node in $\hat{g}_i \in \overline{C_{AP2}}$ $(1 \le i \le k)$.

Lemma 5 Let $\rho: E \rightarrow V_n$ be the node assignment for $G_{AP2} = (V,E)$ that satisfies $\rho(E) = V - IA$ mentioned in Lemma 4. Next, define a function $p_2(IA)$ as follows:

$$p_2(IA) = (|V_a| + |T_a| - t) * (|V_b| + |T_b| - (k - t)) - |E| + \sum_{i=1}^{k} deg(x_i)$$

where $t = |V_a \cap IA|$, $k - t = |V_b \cap IA|$, $deg(x_i)$ is the degree of the node x_i. Then the value $p_2(IA)$ represents the number of dangerous processors $p_2(IA)$ with respect to a cover τ_2 that corresponds to ρ.

For each connected component $\hat{g}_i = (V_i, E_i)$ $(1 \le i \le k)$, we define $min\_deg(a,i)$, $min\_deg(b,i)$ and $\Delta(i)$ in the same way as mentioned in Subsection 4.1. Then we also define IA_{min} to be an $IA = \{x_1, x_2, ..., x_k\}$ for which the value of $p_2(IA)$ is minimum.

Theorem 5 Let IA be a set $IA = (x_1, x_2, ..., x_k)$ where x_i $(1 \le i \le k)$ is a node in $\hat{g}_i$ for $\overline{C_{AP2}} = \{\hat{g}_1, \hat{g}_2, ..., \hat{g}_k : g_{k+1}, ..., g_s\}$. Then $IA = IA_{min}$ if and only if all the conditions C4, C5 and C6 are satisfied by the set IA.
(C4) is the same as C1 in Proposition 1.
(C5) is the same as C2 in Proposition 2.
(C5) $p_2(IA_{min}) = \min\{min\_p_2(\gamma) \mid 0 \le \gamma \le k\}$, where

$$min\_p_2(\gamma) = (|Va| + |T_a| - \gamma)(|Vb| + |T_b| - (k - \gamma)) - |E| + \sum_{i=1}^{k} min\_deg(b,i) + \sum_{i=1}^{\gamma} \Delta(s_i).$$

Example 12 Consider AP_2, F_2 in Example 11 again. We rearrange C_{AP2} into $\overline{C_{AP2}} = \{g_3, g_4 : g_1, g_2\}$. We get $|T_a|=0$, $|T_b|=1$. For g_3, g_4, we get $min\_deg(a,3)=1$, $min\_deg(b,3)=2$, $min\_deg(a,4)=1$, $min\_deg(b,4)=1$, and thus $\Delta(3)=-1$, $\Delta(4)=0$. Thus we compute $min\_p_2(0)=6*(6-2)-8+1+2=19$, $min\_p_2(1)=(6-1)*(6-1)-8+2+1-1=19$, and $min\_p_2(2)= 6*(6-2)-8+2+1-1 =18$. As the result, we get $m=2$. $IA_{min}=\{a_5, a_6\}$ is obtained and the cover τ_2 in Figure 3 is constructed from $IA_{min}=\{a_5, a_6\}$.

6. Conclusion

We have presented two kinds of models (Models 1 and 2) to formulate reconfiguration for fault-tolerant array with spare processors. For the SPA problem on Models 1 and 2, we have developed the algorithms that find an optimal solution in $O(n^2)$ time where n^2 is number of ordinary processors in the array.

The future research work includes the following:

(1) Another extension of the SPA problem ... To develop new applications, we should extend the definitions to n-dimensional hypercubes, and develop efficient algorithms to solve the extended problem.

(2) Application to spare allocation problem (1)(9) ... There exist many similarities between the spare allocation problem and the SPA problem. Thus, the ideas in this paper may possibly be applied to develop a new approach to the spare allocation problem.

References

(1) J.A.Abraham, P.Banerjee, C.Y.Chen, W.K.Fuchs, S.Y.Kuo and A.L.N.Reddy: "Fault tolerance techniques for systolic arrays", IEEE Computer, Vol.20, No.7, pp.56-74(1987).

(2) A.V.Aho, J.E.Hopcroft and J.D.Ullman: "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading Mass.(1974).

(3) P.Banerjee: "Strategies for reconfiguring hypercubes under faults", Proc. 20th International Symposium on Fault-Tolerant Computing, pp.210-217(1990).

(4) P.Banerjee, J.T.Rahmch, C.B.Stunkel, V.S.S.Nair, K.Roy and J.A.Abraham:"An evaluation of system-level fault tolerance in the Intel hypercube multiprocessor", Proc. 18th International Symposium on Fault-Tolerant Computing,pp.362-367(1988).

(5) C.Chen, A.Feng, Y.Takada, T.Kikuno and K.Torii: "Spare processor assignment for reconfiguration of fault-tolerant arrays", Trans. IEICE, Vol. 73E, No.8, pp.1247-1255(1990).

(6) W.K.Chen: "Applied Graph Theory", North-Holland(1971).

(7) N.Hasan and C.L.Liu: "Minimum fault coverage in reconfigurable arrays", Proc. 18th International Symposium on Fault-Tolerant Computing, pp.348-353(1988).

(8) S.Y.Kung, S.N.Jean and C.W.Chang: "Fabrication-time and run-time fault-tolerant array processors using single-track switches", Proc. of International Workshop on Defect and Fault Tolerance in VLSI Systems, pp.281-294(1988).

(9) S.-Y.Kuo and W.K.Fuchs: "Efficient spare allocation in reconfigurable arrays", Proc. 23rd ACM/IEEE Design Automation Conference, pp.385-390(1986).

(10) F.Lombard, R.Nagrini, M.G. Sami and K.Stefanelli: "Reconfiguration of VLSI arrays: a covering approach", Proc. 17th International Symposium on Fault-Tolerant Computing, pp.251-256(1987).

(11) R.G.Melhem: "Bi-level reconfigurations of fault tolerant arrays in bi-modal computational environments", Proc. 19th International Symposium on Fault-Tolerant Computing, pp.488-495(1989).

(12) A.Nayak, N.Santoro and R.Tan: "Fault-intolerance of reconfigurable systolic arrays", Proc. 20th International Symposium on Fault-Tolerant Computing, pp.202-209(1990).

(13) R.Negrini, M.G.Sami and K.Stefanelli: "Fault tolerance techniques for arrays structures used in supercomputing", IEEE Computer, pp.78-87(Feb. 1986).

(14) V.P.Roychowdhury, J.Bruck and F.Kailath: "Efficient algorithm for reconfiguration in VLSI/WSI arrays", IEEE Trans. Computer, Vol.39, No.4, pp. 480-489 (1990).

(15) Y.Suzuki, T.Hirata, M.Imai, M.Yamashita and T.Ibaraki: "Reconfiguration of a fault-tolerant rectangular systolic array", Trans. IEICE Japan, Vol.J70-D, No.3, pp.534-542(1987) (in Japanese).