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

Chang Chen
An Feng
Tohru Kikuno
Koji Torii

RECONFIGURATION ALGORITHM FOR FAULT-TOLENTANT 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 \( \mathcal{O}(n^2) \) 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
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 $\tau$.

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 $OP_1(n) = \{ c_{ij} \mid 1 \leq i, j \leq n \}$ is a set of $n \times n$ ordinary processors, $S_1(n) = \{ a_j, b_j \mid 1 \leq j \leq n \}$ is a set of $2n$ spare processors with $OP_1(n) \cap S_1(n) = \emptyset$, and $L_1(n) = \{ (a_j, c_{ij}), (b_i, c_{ij}) \mid 1 \leq i, j \leq n \}$ is a set of $2n^2$ links.

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

(B) Cover $\tau_1$ 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 $\tau_1 : F_1 \rightarrow S_1(n)$ satisfying the conditions (1) and (2) is called a cover $\tau_1$ 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 $b_i$.

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 $\tau_1$ 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 $\tau_2$ for a given $F_2$ is an injection $\tau_2 : F_2 \rightarrow S_2(n) - F_2$ that satisfies conditions (1) and (2) mentioned in 2.1.

![Figure 1 Array processor $AP_1$](image1)

![Figure 2 Cover $\tau_1$](image2)

![Figure 3 Cover $\tau_2$](image3)
(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 \( \tau_2 \) is shown by the arcs in Figure 3.

### 3. Spare Processor Assignment Problem

#### 3.1 Definition of problem

For the given cover \( \tau_\mu \) and \( F_\mu (\mu = 1, 2) \), any ordinary processor \( d \) satisfying the conditions (3) and (4) is called a **dangerous processor**.

1. \( d \in OP_\mu(n) \setminus (F_\mu \setminus OP_\mu(n)) \)
2. \( d = cij \), then all of spare processors \( b_i \) and \( a_j \) are assigned to faulty processors with respect to \( \tau_\mu \) or are faulty spare processors.

We represent a set of dangerous processors with respect to \( \tau_\mu \) 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 \( cij \in D(\tau_\mu) \) fails afterwards, one of \( b_i \) and \( a_j \) is surely assigned to \( cij \), and thus the system can survive. On the other hand, if an ordinary processor \( cij \in D(\tau_\mu) \) fails afterwards, then there remain no spare processors to take over \( cij \).

**Example 4** Consider the cover \( \tau_1 \) 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 \( \mu (\mu = 1, 2) \) is defined as follows: When an array processor \( AP_\mu \) and a set of faulty processors \( F_\mu \) are given as the input of the problem, we should find a cover \( \tau_\mu^* \) for \( F_\mu \) such that \#(\( D(\tau_\mu^*) \)) is minimum among all possible covers \( \tau_\mu^* \)'s for \( F_\mu \).

**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 \( \mu \) 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 \( \mu (\mu = 1, 2) \), 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 \( \mu \), a single spare processor is individually assigned to a faulty processor \( cij \). 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 \( \tau_1 \)

In this subsection, we will present a necessary and sufficient condition that assures the existence of the cover \( \tau_1 \) for a given \( F_1 \). For the given \( AP_1 \) and \( F_1 \), we construct a bipartite graph \( G_{AP_1} = (V, E) \) such that

\[
V = V_a \cup V_b, \quad E = \{(b_i, a_j) \mid a_j \in S_1(n), \exists k \in \{1, \ldots, n\} \text{ and } E_{b_i} \in F_1\}
\]

where \( V_a = \{a_j \mid a_j \in S_1(n)\} \) and \( V_b = \{b_i \mid b_i \in S_1(n)\} \). We call nodes \( a_j \in V_a \) and \( b_i \in V_b \) an a-node and b-node, respectively. Let \( C_{AP_1} = \{g_1, g_2, \ldots, g_s\} \) be a set of connected components of the bipartite graph \( G_{AP_1} \).

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

For \( G_{AP_1} = (V_a \cup V_b, E) \), 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 \( \rho(e_1) \neq \rho(e_2) \) (that is, \( \rho \) is an injection).

Lemma 1 Consider an array processor \( AP_1 \) and a set of faulty processors \( F_1 \). Then a cover \( \tau_1 \) exists for \( F_1 \) if and only if a node assignment \( \rho \) exists for a bipartite graph \( G_{AP_1} \).

Theorem 1 Consider an array processor \( AP_1 \) and a set of faulty processor \( F_1 \). Then, a cover \( \tau_1 \) exists for \( F_1 \) if and only if for each connected component \( g_i = (V_i, E_i) \) in \( C_{AP_1} \) either \( |V_i| = |E_i| + 1 \) or \( |V_i| = |E_i| + 1 + 1 \) holds.

(Proof) Only if . Assume that a cover \( \tau_1 \) exists. Then we can conclude that \( |V_i| = |E_i| + 1 \) holds. Additionally, for each connected component \( g_i = (V_i, E_i) \), the relation \( |E_i| + 1 \geq |V_i| \) holds. These two facts imply that \( |V_i| = |E_i| + 1 \) or \( |V_i| = |E_i| + 1 + 1 \) holds. We can construct a node assignment \( \rho_i \) for each connected component.

Example 7 Consider the set of connected components \( C_{AP_1} = \{g_1, g_2, g_3\} \) in Figure 6(b). For \( i = 2, 3 \), \( V_i = E_i + 1 \) holds. For \( i = 1 \), \( |V_1| = |E_1| + 1 \) holds. Thus, at least one cover \( \tau_1 \) exists. Examples of such covers are shown in Figures 4 and 5.

4.2 Necessary and sufficient condition

Assume that \( C_{AP_1} = \{g_1, g_2, \ldots, g_s\} \) is rearranged into \( \tilde{C}_{AP_1} = \{\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_k\} \) as follows:

- Each \( \tilde{g}_i = (V_i, E_i) \) satisifies \( |V_i| = |E_i| + 1 \), and each \( g_i = (V_i, E_i) \) satisifies \( |V_i| = |E_i| + 1 + 1 \).

Lemma 2 Define \( I_A = \{x_1, x_2, \ldots, x_k \mid x_i \in V\} \) for a bipartite graph \( G_{AP_1} = (V, E) \). Then there exists a node assignment \( \rho: E \rightarrow V \) for \( G_{AP_1} = (V, E) \) with \( \rho(V) = V \).
IA if and only if $x_i$ in IA is a node in $\hat{G}_i \in \vec{G}_{AP_1}$ ($1 \leq i \leq k$).

Lemma 3 Let $\nu: E \rightarrow V$ be the node assignment for $G_{AP_1} = (V,E)$, $V = V_a \cup V_b$ that satisfies $\nu(E) = V \setminus IA$ 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 $\tau_1$ that corresponds to $p$.

(Proof) See Reference (5).

Example 8 Consider the connected components $\vec{G}_{AP_1} = (G_2, G_3 : G_1)$ 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 $\tau_1$ 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 = $(x_1, x_2, ..., x_k)$, for which the value of $p_1(IA)$ is minimum. For each connected component $G_i = (V_i, E_i)$ ($1 \leq i \leq k$) in $\vec{G}_{AP_1}$, we define the minimum degrees of a-nodes and b-nodes as follows:

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

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

(C1) $\deg(x_i) = \{ \min_{deg}(a,i) \text{ if } x_i \in V_a \cap V_i \}$ \text{and} $\min_{deg}(b,i) \text{ if } x_i \in V_b \cap V_i \}$. 

(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 $G_i = (V_i, E_i)$ ($1 \leq i \leq k$) in $\vec{G}_{AP_1}$, we define the difference $\Delta(i) = \min_{deg}(a,i) - \min_{deg}(b,i)$. Let $Ind(a) = \{ i | x \in IA \cap V_a \text{ and } x \in V_i \}$ and $Ind(b) = \{ i | x \in IA \cap V_b \text{ and } x \in V_i \}$ for $G_i$. Then $\vec{G}_{AP_1}$ is the set of indices of nodes in IA. Note that $Ind(a) \cup Ind(b) = \{1, 2, ..., k\}$ for $\vec{G}_{AP_1}$.

Proposition 2 Let $Ind(a) = \{ i | x \in IA_{min} \cap V_a \text{ and } x \in V_i \}$ for $G_i$ and $Ind(b) = \{ i | x \in IA_{min} \cap V_b \text{ and } x \in V_i \}$ for $G_i$ and $n = \min_{deg}$ and $\Delta(i)$.

Consider a sequence $(\Delta(1), \Delta(2), ..., \Delta(k))$ and sort it in a nondecreasing order of values $\Delta(i)$'s. Let $\Delta = (\Delta(1), \Delta(2), ..., \Delta(k))$. Let $IA' = (x_1, x_2, ..., x_k)$ such that the result sequence, for which $\Delta(1) \leq \Delta(2) \leq ... \leq \Delta(k)$ holds. Let $IA = (x_1, x_2, ..., x_k)$, where $x_i$ is a node in $G_i$ $(1 \leq i \leq k)$ such that the number of a-nodes in IA is $\gamma$.

Proposition 3 Let $\gamma$ be the number of a-nodes in a given $IA_{min}$, and let $\min_{p_1}(\gamma) = (|V_a| - \gamma) (|V_b| - (k - \gamma)) - |E| + \sum_{i=1}^{k} \min_{deg}(b,i) + \sum_{i=1}^{k} \sum_{i=1}^{\gamma} \deg(x_i).$ Let $\Delta = (\Delta(1), \Delta(2), ..., \Delta(k))$. Then the value $p_1(IA_{min})$ must satisfy the following condition:

(C3) $p_1(IA_{min}) = \min \{ \min_{p_1}(\gamma) | 1 \leq \gamma \leq k \}$. 

(Proof) By the definitions of $p_1$, Ind(a) and Ind(b),

$$p_1(IA) = D + \sum_{i=1}^{k} \deg(x_i) + \sum_{i=1}^{\gamma} \deg(x_i)$$

is derived where $D = (|V_a| - \gamma) (|V_b| - (k - \gamma)) - |E|$. Then, $p_1(IA) \geq D + \sum_{i=1}^{k} \min_{deg}(b,i) + \sum_{i=1}^{\gamma} \Delta(i)$. By the property of $\Delta$,

$$p_1(IA) \geq D + \sum_{i=1}^{k} \min_{deg}(b,i) + \sum_{i=1}^{\gamma} \Delta(i) = \min_{p_1}(\gamma).$$

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

Next, assume that the minimum value of $\min_{p_1}(\gamma)$ is $\min_{p_1}(m)$. Then by definition of $IA_{min}$, the number of a-nodes in $IA_{min}$ must be $m$. Thus, $p_1(IA_{min}) = \min \{ \min_{p_1}(\gamma) | 1 \leq \gamma \leq k \}$. 

Theorem 2 Let $IA$ be a set IA = $(x_1, x_2, ..., x_k)$ where $x_i$ ($1 \leq i \leq k$) is a node in $G_i$ and $G_{AP_1} = \vec{G}_{AP_1}$ for $G_1$, $G_2$, ..., $G_k$. Then $IA = IA_{min}$ if and only if it satisfies the conditions C1, C2 and C3.

(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. Then let $m$ be the number of a-nodes in IA. On the other hand, let $IA'$ be any sequence $(x'_1, x'_2, ..., x'_k)$ such that $x'_i$ is a node in $G_i$ $(1 \leq i \leq k)$. Let $Ind'(a) = \{ i | x \in IA' \cap V_a \text{ and } x \in V_i \}$ and $Ind'(b) = \{ i | x \in IA' \cap V_b \text{ and } x \in V_i \}$ for $G_i$. Then $m$ is the number of a-nodes in $IA'$. 

Then, by the definition of $\min_{deg}$ and $\Delta(i)$ and $\Delta$. 

456
\[ p_1(IA') = D_m + \sum_{i \in Ind(a)} \deg(x_i) + \sum_{i \in Ind(b)} \deg(x_i) \]
\[ \geq D_m + \sum_{i \in Ind(a)} \min_{deg}(a,i) + \sum_{i \in Ind(b)} \min_{deg}(b,i) \]
\[ = D_m + \sum_{i \in Ind(a)} \min_{deg}(a,i) + \sum_{i \in Ind(b)} \Delta(i) \]
\[ \geq D_m + \sum_{i \in Ind(a)} \min_{deg}(a,i) + \sum_{i \in Ind(b)} \min_{deg}(b,i) + \sum_{i = 1}^{m'} \Delta(i) = \min_{p_1}(m'). \]

Furthermore, since IA satisfies the condition C3, \( \min_{p_1}(m') \geq \min_{p_1}(m) \) is derived. Next, consider the derivation of \( p_1(IA') \geq \min_{p_1}(y) \) 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 \( C_{AP:1} = \{g_2, g_3 : g_1 \} \) 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 \( \tilde{\Delta}(1) = (-1, 0) \). Now, we compute \( \min_{p_1}(0) = 6 * (4 - 2) = 4 + 1 - 2 = 7, \min_{p_1}(1) = (6 - 1) * (4 - 1 - 8 + 1 + 2 + 1 - 9) = 0, \) and \( \min_{p_1}(2) = (6 - 2) * 4 - 8 + 1 + 2 - 1 = 10 \). Finally, we get \( m = 0, IAm_{min} = \{b_4, b_5\} \).

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 \( \tau_1 \))

1.1 Construct the bipartite graph \( GA_{PI} \) and the set \( C_{AP:1} = \{g_i : V_i, E_i\} \).

1.2 Check whether each \( g_i \in C_{AP:1} \) satisfies the condition \( \{V_i = E_i, (1 \leq i \leq k) \} \) or not. If it is satisfied, rearrange the sequence \( C_{AP:1} = \{g_{i1}, g_{i2}, \ldots, g_{ik}\} \) in nondecreasing order.

Step 2 (Computation of \( IAm_{min} \))

2.1 Get the values \( \min_{deg}(a,i), \min_{deg}(b,i) \) for each \( g_i \) \( (1 \leq i \leq k) \).

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

2.3 Get the set \( \{p_1(y) | \Delta(1), \Delta(2), \ldots, \Delta(k) \} \) and find the value \( \gamma \) giving the minimum value of \( \min_{p_1}(y) \). Let \( m \) be the resultant value.

2.4 Finally, let \( IAm_{min} = \{x_{s_1}, x_{s_2}, \ldots, x_{s_m}, x_{s_{m+1}}, \ldots, x_{s_k}\} \) such that each \( x_{s_i} \) \( (1 \leq s_i \leq m) \) is a-node in \( g_{si} \) with \( \deg(x_{s_i}) = \min_{deg}(a, s_i) \), and each \( x_{s_i} \) \( (m+1 \leq i \leq k) \) is b-node in \( g_{si} \) with \( \deg(x_{s_i}) = \min_{deg}(b, s_i) \).

Step 3 (Construction of \( \tau_1 \))

3.1 Determine an assignment \( \rho_i, (1 \leq i \leq k) \) as follows:

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

\( \text{Case 2} \) For each \( g_{kl} \) \( (k+1 \leq i \leq l) \), define \( \rho_{kl} \) by the procedure of Case 2 in Theorem 1's proof.

3.2 Compose an assignment \( \rho \) for \( GA_{PI} \) from \( \rho_1, \rho_2, \ldots, \rho_k \).

3.3 Define a cover \( \tau_1 \) 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 \( GA_{PI} \) shown in Figure 6, and get \( C_{AP:1} = \{g_2, g_3 : g_1\} \). At Step 2, we get \( IAm_{min} \) described in Example 9. Finally at Step 3, we can construct an optimal cover \( \tau_1 \) shown in Figure 5.

Theorem 3 Algorithm SPA-1 solves a given SPA problem on Model 1 in \( O(n^2) \) time, where \( n \) is the number of outer 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 \( C_{AP:1} \) 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 \( (\Delta(1), \Delta(2), \ldots, \Delta(k)) \) to get the sequence \( \tilde{\Delta} \), then at Substep 2.3, we compute \( \min_{p_1}(y) (0 \leq \gamma \leq k) \).

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 \( g_{kl} \) \( (k+1 \leq i \leq l) \). 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 \( IAm_{min} \) 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 \( \tau_1 \) exists,
the number of connected component in $C_{AP2}$ 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 $\tau_1$ 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 $\tau_2$

For the given $AP2$ 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 $C_{AP2} = (g_1, g_2, ..., g_s)$, $g_i = (V_i, E_i)$ (1 ≤ i ≤ s) be a set of connected components of the bipartite graph $G_{AP2}$. Then, we divide $V_i$ (1 ≤ i ≤ 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 $V_{in} = \{ b_i, a_j \mid b_i \in F_2, a_j \notin F_2 \}$. Let $V_f = V_{if} \cup V_{2f} \cup ... \cup V_{sf}$ and $V_n = V_{in} \cup V_{2n} \cup ... \cup V_{sn}$.

Next, we define a node assignment where $\rho: E \to V_n$ 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 $\tau_2$ exists for $F_2$ if and only if a node assignment $\rho$ exists for a bipartite graph $G_{AP2}$.

**Theorem 4** Consider an array processor $AP2$ and a set of faulty processors $F_2$. Then a cover $\tau_2$ exists for $F_2$ if and only if each connected component $g_i = (V_i, E_i)$ in $C_{AP2}$ satisfies either $V_{in} = 0$ or $V_{in} = 1$.

**Example 11** Consider $AP2$ 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, $g_1 = (V_1, E_1)$, $g_2 = (V_2, E_2)$, $g_3 = (V_3, E_3)$, $g_4 = (V_4, E_4)$ produces $V_{in} = 0$, $V_{in} = 1$, $V_{in} = 1$, $V_{in} = 1$. On the other hand, $g_5 = (V_5, E_5)$, $g_6 = (V_6, E_6)$, $g_7 = (V_7, E_7)$, $g_8 = (V_8, E_8)$ produces $V_{in} = 1$, $V_{in} = 1$, $V_{in} = 1$, $V_{in} = 1$. Clearly, for $i = 1, 2, V_{in} = 1$ holds and for $i = 3, 4, V_{in} = 1$ holds. Thus, at least one cover $\tau_2$ (shown in Figure 3) exists.

5.2 Necessary and sufficient condition

Assume that $C_{AP2} = (g_1, g_2, ..., g_s)$ is rearranged into $\tilde{C}_{AP2} = (\tilde{g}_1, \tilde{g}_2, ..., \tilde{g}_k : g_{k+1}, ..., g_s)$ as follows:

For $\tilde{g}_i = (V_i, E_i)$ (1 ≤ i ≤ k), $V_{in} = 1$, $V_{in} = 1$ hold, for $g_i = (V_i, E_i)$ (k+1 ≤ i ≤ s), $V_{in} = 0$, $V_{in} = 1$ hold.

Additionally, we define two sets $T_a = \{ a_j : a_j \in S_2(n) \cap F_2 \}$ and $\forall k (1 \leq k \leq \lceil \log_2 n \rceil) c_{kk} \notin F_2)$, $T_b = \{ b_i : b_i \in S_2(n) \cap F_2 \}$ and $\forall k (1 \leq k \leq \lceil \log_2 n \rceil) c_{kk} \notin F_2$).

**Lemma 4** Define $IA = \{ x_1, x_2, ..., x_k \mid x_i \in V \}$ for a bipartite graph $G_{AP2} = (V, E)$. Then a node assignment $\rho: E \to V_n$ for $G_{AP2} = (V, E)$ with $\rho(E) = V \cdot IA$ exists if and only if $x_i$ in $IA$ is a node in $\tilde{g}_i, \in \tilde{C}_{AP2} (1 \leq i \leq k)$.

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

$$p_2(IA) = \left( |V_a| + |T_a| + 1 \right) \cdot \left( |V_b| + 1 \right) - |E| - k \Sigma\text{deg}(x_i)$$

where $i = V_a \cap IA \lor k - i = V_b \cap IA \lor \text{deg}(x_i)$ 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 $\tau_2$ that corresponds to $\rho$. 

458
For each connected component \( \hat{g}_i = (V_i, E_i) \) \((1 \leq i \leq k)\), 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 \( I_{A_{\text{min}}} \) to be an \( I_A = \{x_1, x_2, \ldots, x_k\} \) for which the value of \( p_2(I_A) \) is minimum.

**Theorem 5** Let \( I_A \) be a set \( I_A = \{x_1, x_2, \ldots, x_k\} \) where \( x_i \) \((1 \leq i \leq k)\) is a node in \( \hat{g}_i \); for \( C_{A_{\text{min}}} = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_k\} \), then \( I_A = I_{A_{\text{min}}} \) if and only if all the conditions C4, C5 and C6 are satisfied by the set \( I_A \).

(C4) is the same as C1 in Proposition 1.

(C5) is the same as C2 in Proposition 2.

(C6) \( p_2(I_{A_{\text{min}}}) = \min\{\min_{p_2}(\gamma) \mid 0 \leq \gamma \leq k\} \), where

\[
\min_{p_2}(\gamma) = (\forall a_1 + \Gamma_2 \cdot \gamma) (\forall b_1 + \Gamma_3 \cdot (k - \gamma)) \\
- \sum_{i=1}^{k} \min_{deg}(b_i) \cdot \gamma + \sum_{i=1}^{\gamma} \Delta(s_i).
\]

**Example 12** Consider \( A_{\text{P}2}, F_2 \) in Example 11 again. We rearrange \( C_{A_{\text{P}2}} \) into \( C_{A_{\text{P}2}} = \{g_3, g_4, g_1, g_2\} \). We get \( \Gamma_2 = 0, \Gamma_3 = 1, \Gamma_4 = 0 \). For \( g_3, g_4 \), we get \( \min_{deg}(a, b) = 1, \min_{deg}(b, b) = 2, \min_{deg}(a, b) = 1, \min_{deg}(b, b) = 1, \) and thus \( \Delta(3) = 1, \Delta(4) = 0 \). Thus we compute \( \min_{p_2}(0) = 6 \cdot (6-2) = 18, \min_{p_2}(1) = 6 \cdot (6-1) = 18, \min_{p_2}(2) = 6 \cdot (6-2) = 18 \). As the result, we get \( m = 2 \), \( I_{A_{\text{min}}} = \{a_5, a_6\} \) is obtained and the cover \( C_2 \) in Figure 3 is constructed from \( I_{A_{\text{min}}} = \{a_5, a_6\} \).

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


