통신망의 다중연결성을 이용한 결함허용 알고리즘

(Fault-Tolerant Algorithm using Multi-Connectivity of Communication Networks)

문 윤 호 † 김 병 기 ††
(Yun-Ho Moon) (Byung-Ki Kim)

요 약 본 논문의 목적은 네트워크상에서 한 시스템의 요소가 결함으로 인해 통신장애를 일으키는 경우에 대하여 새로운 회복알고리즘을 제안하는 것이다. 또한 인접행렬을 사용하면서 그 알고리즘을 시뮬레이션한다. 우리는 제안된 알고리즘의 한 번 수행시 하나의 결함노드를 복구하도록 하여 결국 결함시스템을 점진적으로 통신가능한 네트워크로 재구성 할 수 있다. 그렇게 하기위해 본문에서는 인접행렬을 이용하며 시뮬레이션된 회복과정을 연결한 MATRECO라는 새로운 회복알고리즘을 제시하였다.

Abstract The purpose of this paper is to propose new recovery algorithm for case of a system element raises communication obstacle due to faults in networks. Also we are simulate the algorithm using adjacency matrix. We recover one faulty node per each excution of proposed algorithm so that we can be reconstruct the faulty system gradually to communicatable network. For that, this paper propose a new recovery algorithm named MATRECO which connect the recovery process is simulated by use of adjacency matrix.

1. 서 론

컴퓨터의 발달에 따라 시스템의 보다 높은 신뢰도[1]가 중요관심사가 되었고 아울러 시스템의 신뢰도를 증가시킬 수 있는 구성방법이 활발히 연구되고 있으며 이제 시스템의 신뢰도는 그 시스템의 척도를 가늠하는데 가장 중요한 요소중 하나로 인식[2]되고 있다. 이와 맥락을 같이하여 신뢰도 향상을 위한 효율적 방법중 하나로 통신시스템의 구성요소에 이상이 생기는 경우 전체 시스템의 처리효율에 별 영향을 미치지 않도록 처리기능을 분배하며 정상화대처가 가능하게 하는 이른바 결함허용[3,4]에 대한 연구가 매우 활발하게 이루어지고 있는데 이는 결함발생시 시스템이 자체적으로 결함의 검사[5]와 위치확인[6] 및 회복을 수행하게 하여 어느 한계까지는 결함을 감당할 수 있도록 하려는 것[7]이다. 여기서 결함(faults)이란 시스템을 고장으로 인한 동작불능의 상태(failure or error)로 발전시킬 수 있는 원인이 되는 것을 의미하며 그 발생원인 및 정도가 일시적인가 어디에 기인되었는가에 따라 permanent physical fault, transient physical fault, design fault, specification fault등으로 구분되기도 한다[8]. 신뢰도가 절대적으로 중요한 실시간 처리분야에서의 시스템 유지나 통신기능 향상을 위해서는 네트워크상에서 어떠한 결함 발생시 그것을 극복 할 수 있는 시스템 구축[9,10]이 필요한데 본 연구는 그러한 시스템 구축시 수반될 수 있는 회복문제를 처리하기위한 과정에 대한 것이며 특수한 대상이 아닌 일반적인 네트워크에서 결함이 발생하더라도 전체적 통신처리 효율을 서서히 저하되게 하면서 어느정도 이상 저하되기 이전에 분산처리 개념[8]을 이용하여 결함을 복구할 수 있게하는 알고리즘을 제시하고자 한다. 이 알고리즘은 네트워크를 복구하며 연결상태를 재구성해준다. 여기에서는 중복(redundancy)개념[11]을 이용한다. 기존의 회복방식의 전형

† 비 회 원 : 전주기전여자대학 컴퓨터과 교수
yhmoon@kns.kijeon-c.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
bhkim@computing.soongsil.ac.kr
논문접수 : 1998년 5월 12일
심사완료 : 1999년 10월 1일

적인 진행단계를 살펴보면 다음과 같다. 우선 결함이 발생한 상태에서 첫째, 임의의 활성화되어있는 노드 Xi가 결함처리를 주관하는 국부적감시자(local supervisor)로서 자가진단프로그램[12]을 통하여 인접연결성을 갖는 노드 N(Xi)들중 결함노드 Xj를 찾아내며 둘째, Xi는 N(Xi) 중에서 대체가능(spare)노드 Xk를 찾는 과정을 반복실행하며 Xk는 결함노드 Xj의 결함이전상태로 간주한다. 본문에서 제안된 알고리즘은 이러한 기본개념을 확장하여 일반적인 네트워크에서 임의의 개수만큼 결함이 발생했을 경우 처리가능한 방식으로 고안되어있음을 볼 수 있다. 본 연구를 바탕으로 더 많은 자동분산 회복 알고리즘들에 대해 연구하게 될 것이며 한번에 보다 많은 노드를 복구할 수 있는지를 연구할 수 있는 기초가 될 것으로 기대한다.

2. 네트워크 모델링

분산회복 알고리즘을 적용해 보기 위한 통신망 모델을 설정함에 있어 최우선으로 고려한 것은 일반적으로 통신망 형태를 적용할 수 있게 하기 위함이었고 여기에서 설정된 모델은 임의로 연결성을 부여하여 구성한 네트워크이다[13]. 본문의 네트워크외의 다른 형태로 구성된 네트워크에도 물론 이 알고리즘은 적용 가능하다. 여기서는 임의의 형태로 된 열 여섯개의 요소를 갖는 통신망을 기본 모델로 설정하였으며 그림 1과 같다.

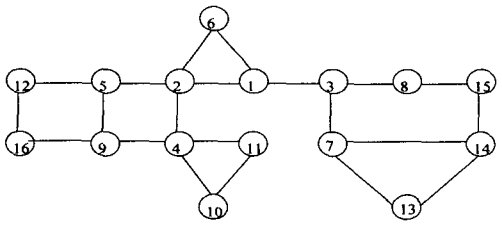

그림 1 기본시스템

2.1 다중연결을 고려하기 위한 프로시듀어

그림 1과같은 기본 모델에 다음과 같은 EXTRACT, SELECT, CONNECT 프로시듀어[12]를 적용하면 새로운 연결형태를 갖는 망모델이 구성되는데 EXTRACT란 비교적 오버헤드가 적으면서 최대직경거리(diameter)가 작은 스패닝(spanning)트리를 구하기 위한 알고리즘이며 SELECT란 스패닝트리에서 깊이(depth)가 최소인 루트를 갖는 트리형태를 만들기위해 노드를 선택하는 알고리즘이고 CONNECT란 루트를 갖는 트리에 변형이전의 초기모델에 존재하는 연결성을 모두 고려하여 연결했을 경우 그에대해 임의의 k개의 결함을 극복할 수 있는 연결성을 부여하기위한 알고리즘이다. 다음 그림 2는 결함 고려를 위한 첫 번째 수행과정인 EXTRACT이다.

<procedure EXTRACT> : Minimal Diameter Spanning Tree Extraction

1. E(i)는 노드 i가 갖는 간선들의 전체 개수, N(i)는 노드 i의 인접 노드들의 집합, D(i)는 내림차순에서의 i번째 노드, FIRST(i)는 큐 FIRST에서의 i번째 위치, SECOND(i)는 큐 SECOND에서의 i번째 위치로 각각 정의하며 E의 초기치는 0으로 한다.
2. 전체 노드를 N(i)에 대해 내림차순으로 정렬하여 D(1)의 E(i)가 D(2)의 E(i)보다 크면 D(1)을 중심노드(center)로 하며, 그렇지 않으면 D(1)과 간선수가 같은 D(i) 중에서 N(Xi)에 터미널(terminal)노드가 없는 것중 최소오더(minimum order[9])의 D(i)가 중심노드가 된다(first-selection). 만일 모든 D(i)가 N(Xi)에 터미널노드를 가지면 D(1)이 중심노드가 된다(art-selection).
3. 중심노드를 FIRST(rear)로 넣어준다.
4. 다음의 일련의 처리를 <조건>이 만족되어 루프를 벗어날 수 있을 때까지 반복 수행한다. N(center)를 E(i)에 따라 내림차순으로 정렬한 후 FIRST와 SECOND에 정렬순서대로 입력하고 <SECOND (front)와 SECOND(rear)가 같으면(즉 큐 SECOND가 비어 있으면) 5번으로 분기> 하며 SECOND (front)를 center가 되게 한다.
5. 다음의 일련의 처리를 <조건>이 만족되어 루프를 벗어날 수 있을 때까지 반복 수행한다. FIRST(front)를 center로 주고 N(center)중 center와 연결해도 사이클(cycle)이 형성되지 않는 노드들을 center와 연결하며 연결시마다 E를 1씩 증가시키고 만일 E가 21이면 루프를 벗어난다.

그림 2 스패닝트리 프로시듀어

그림 2의 EXTRACT를 수행하여 산출되는 그래프는 그림 3과 같은 스패닝 트리 형태이다.

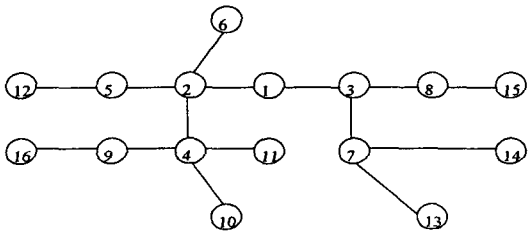

그림 3 EXTRACT에 의한 스패닝 트리

결함 고려를 위한 다중연결성을 구성하는 두 번째 과정은 그림 3을 바탕으로 SELECT프로시듀어를 실행하는 것이다. 다음 그림 4는 SELECT의 수행 과정을 보여주고 있다.

<procedure SELECT> : Centralized Rooted Tree Selection

1. DP(i)는 노드 i가 그래프에서 도달 가능한 최대 거리, I(i)는 오름차순에서의 I번째 노드로 각각 정의하며 각 간선의 길이는 모두 1로 간주한다.
2. 각 노드의 D(i)를 비교하여 오름차순으로 정렬하여 I(1)의 D(i)가 I(2)의 D(i)보다 크면 I(1)을 루트로 하고 그렇지 않으면 N(Xi)에 터미널노드가 없는 노드 중 정렬 순서가 가장 빠른 노드가 루트가 된다. 만일 모든 I(i)가 N(Xi)에 터미널노드를 가진다면 I(1)이 루트가 된다.
3. 계층구조의 구성을 위해 루트를 레벨 k로 하고 레벨 i의 자노드(children)는 레벨을 i+1로 정의 하면서 터미널노드까지 레벨을 할당한다(k<=i<=k+r-1). 이때 최하위 터미널노드의 레벨은 k+r 이 되며 이미 연결되어 있는 노드들은 제외한다.

그림 4 최소깊이 프로시듀어

그림 4의 SELECT를 수행하여 산출되는 그래프는 그림 5와 같은 최소깊이 트리 형태이다.

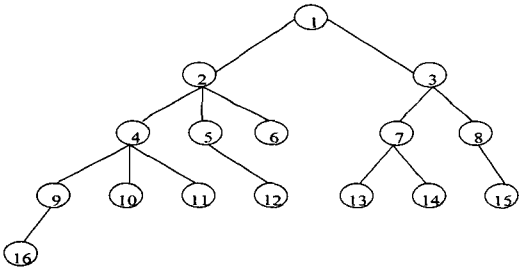

그림 5 SELECT에 의한 최소깊이 트리

이러한 최소깊이 트리가 구해진 후 원래 시스템 초기 모델이 가지고 있던 연결성을 모두 재고하기 위해 그림 5에 추가적으로 초기 시스템의 연결성을 나타낸 것이 그림 6이다.

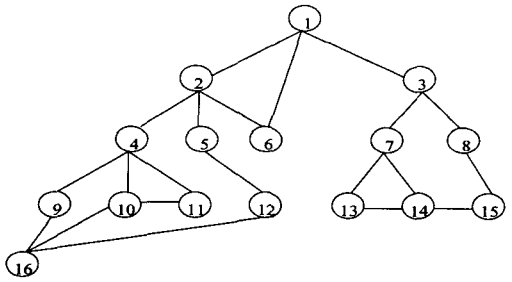

그림 6 기초 모델의 연결성을 고려한 시스템

결함발생에 대비한 다중연결성을 구성하는 마지막 단계의 프로시듀어가 CONNECT이다. 다음 그림 7은 CONNECT의 수행과정을 보여주고 있다.

<procedure CONNECT> : Node and Spare Connection

1. 루트노드를 레벨 k로 시작하여 프로시듀어 Tr[8]과 같은 방법으로 각 노드에 레벨을 부과하고 0부터 k-1까지의 k개 레벨 각각에 대하여 하나씩의 예비(spare)노드를 할당한다.
2. I(0<=i<=k+r-2)에 대하여 J(j<=i+k+1)까지의 모든 후손노드들(decendants)과 연결한다.
3. 싸이클을 형성하고 있는 노드들에 대하여 다음을 수행한다. 첫째, 내부에 싸이클이 존재하지 않는 최소의 싸이클(unit cycle)상의 노드들에 대해 연결의 중복성이 나타날 때까지 현재노드로 부터 거리 k+1이내에 존재하는 노드들 중 현재노드와 동일 부노드(parent)를 갖지 않는 노드들을 현재노드와 연결한다. 둘째, 자노드를 공유하는 부노드들을 서로 연결한다(동일 부노드를 공유하는 자노드들의 연결포함). 셋째, 노드 i(i는 "상위레벨에서 하위레벨로, 왼쪽에서 오른쪽으로" 진행)로부터 거리 k만큼의 하위레벨에 있는 노드에 직접 연결된 노드가 i로부터 거리 k+1 보다도 멀리있는 하위레벨에 존재하는 경우는 노드 i와 직접 연결한다.

그림 7 연결회복 프로시듀어

그림 7의 CONNECT를 수행하여 산출되는 그래프는 그림 8과 같은 트리형태이다.

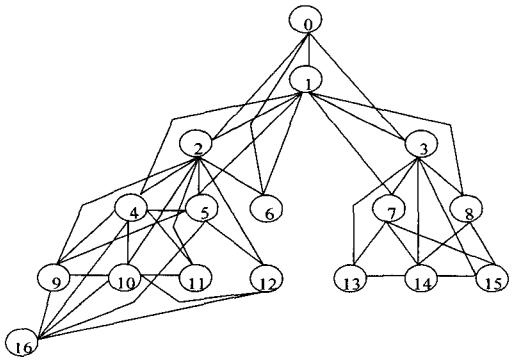

그림 8 결함을 고려한 연결성을 갖는 시스템

2.2 회복 알고리즘 구현을 위한 모델 설정

앞절에서 제시된 알고리즘을 차례대로 수행하면 특수한 형태(트리나 루프등)에 국한되지 않는 일반적인 망모델들에 대해 일련의 변환을 거치면서 다중연결을 설정할 수 있음을 보았다. 이러한 다중연결은 단순히 통신경

로만을 확장하는 측면이 아니라 시스템 구성요소중 결함이 발생하는 경우에 대비한 것이기도 하다. 본논문에 나타나 있는 모델 외에도 여러가지의 경우를 더 적용하여 실험해본 결과로는 본문과 마찬가지의 결과를 유도할 수 있었다. 따라서 2.1절에서와 같은 일련의 처리과정을 거쳐서 구성된 시스템을 초기 모델의 기본형태를 감안하여 연결성을 유지한채로 재구성할 수 있는데 그것은 일반적 형태의 모델이면서 결함을 대비한 다중연결이 내포된 시스템이라 할 수 있고 구성결과는 그림 9와 같다.

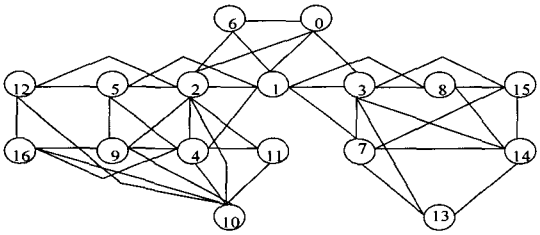

그림 9 연결성이 고려된 망모델(F-T 시스템)

3. F-T 시스템에서의 회복 알고리즘

본절에서는 앞서 설정, 구현된 모델망을 이용하여 시스템에서 결함이 발생하였을 경우 이를 분산방식으로 회복하는 알고리즘의 구현방식을 사례를 가정하여 알아보기로 한다. 이를 위해 우선 기존의 시스템을 연결성이 고려된 인접행렬형태로 변환하는 것이 필요하고 그 인접행렬을 바탕으로 하여 본문에서 제안되는 알고리즘을 단계별로 수행해 가는 것을 반복해야 하며 그 수행결과에 의해 최종적으로 산출되는 인접행렬의 관계성을 토대로 초기모델 망형태의 그래프로 환원해주는 과정이 순차적으로 진행된다.

3.1 시스템의 행렬식 변환

지금까지의 과정을 통하여 구성된 시스템을 기초모델로 이용하면서 구성요소간의 연결상태를 가시적인 방법으로 표현하기 위해 여기서는 시스템의 유형을 인접행렬로 표시하는데 각 요소와 요소간의 연결 관계성(간선:edge)이 있는 경우는 1로 표기하고 요소간에 연결이 없는 경우는 공란으로 나타내었다. 또 이미 설정된 모델망이 루트노드를 갖는 계층형 구조로도 변환이 가능하므로 레벨에 관한 정보를 표현해 주기 위해 별도의 배열레벨을 이용하였다. 그림 9의 시스템을 이러한 원리에 적용하여 인접행렬로 표현하고 행렬의 이름을 CONNECTIVITY(X,Y)라 명명하였으며 그 결과는 그림 10과 같다.

CONNECTIVITY(X,Y)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	NO	LE
0		1	1	1			1											0	0
1	1		1	1	1	1	1	1	1									1	1
2	1	1			1	1	1			1	1	1	1					2	2
3	1	1						1	1					1	1	1		3	2
4		1	1			1				1	1	1					1	4	3
5		1	1		1					1			1				1	5	3
6	1	1	1															6	3
7		1		1										1	1	1		7	3
8		1		1											1	1		8	3
9			1		1	1					1		1				1	9	4
10			1		1					1		1	1				1	10	4
11			1		1						1							11	4
12			1			1				1	1						1	12	4
13				1				1							1			13	4
14				1				1	1					1		1		14	4
15				1				1	1						1			15	4
16					1					1	1		1					16	5

그림 10 모델의 열결성을 나타내는 인접행렬

3.2 회복 알고리즘

앞절에서 기본적인 모델망의 연결성을 모두 고려한 행렬구성방식을 소개하였으며 이 절에서는 앞에서 구성된 인접행렬을 이용하면서 원래의 그래프(모델망)에 대해 결함이 발생했을 경우 회복작업을 하는 과정을 수행할 수 있는 프로시듀어를 제안하고자 한다. 이것을 MATRECO라 명명하였으며 알고리즘의 응용프로그램은 C 언어로 구현되어 시뮬레이션 되었고 본문의 예에서는 상세한 흐름의 이해를 돕기 위해 서술식 알고리즘의 형태로 설명과 함께 각 단계마다 일련번호를 붙여서 표기하였다. 이 알고리즘은 한 번 수행시마다 하나의 노드가 회복되어지는데 시스템 구성요소간의 연결상태에 따라 인접한 연결요소들의 관계를 행렬로 표현하는 경우 그 인접행렬을 CON[ ] 으로 표기하면 알고리즘이 매번 수행시마다 그 노드의 연결성이 변화되고 행렬 CON[ ] 내에서 해당 노드에 대응되는 행렬요소가 변환되어 결국 최종적으로 구성되는 행렬에서는 복구된 망의 연결성이 나타나게 되는데 그림 보여주게 된다. 그리고 본문에서 제시된 구조와 같은 일반적인 망의 형태들은 앞서 소개한 SELECT 프로시듀어(그림 4)에 의해 그림 5와 같은 트리모양으로 변형가능하므로 망요소들의 표현시는 트리구조의 용어를 사용하기로 하는데 본문에서 설정된 모델에 SELECT를 적용한 그림 5는 예비노드를 부가하였을 때 다음 그림 11과 같은 형태로 재구성이 가능하며 본문의 알고리즘에서 부노드나 자노드의 관계는 이 시스템을 근거로 하고 있다.

이 절에서 소개되는 알고리즘은 임의의 노드 X에서

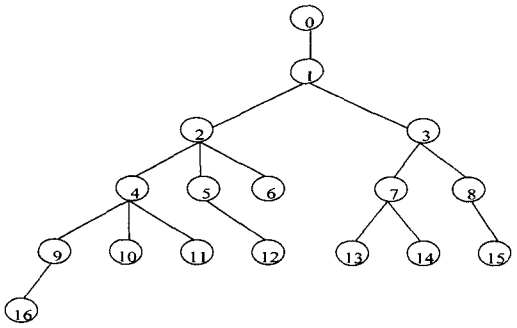

그림 11 예비노드가 고려된 트리형태의 초기모델

결함이 발생되었을 경우 X의 인접노드중 결함이 없는 노드 N(Xi)(트리형태로 변형시 parent 노드)가 국부적감시자 역할을 맡게되며 인접노드들을 조사하여 임의의 대체가능한 하나의 노드를 선택하여 처리하고 다시 과정을 시스템 전체가 안정될때까지 반복하게 된다. 만일 적당한 대체가능노드를 찾지 못한경우는 N(Xi)가 국부적감시자의 역할을 제대로 수행할 수 없는 연결상태로 판단할 수 있으므로 다른 노드에 역할을 넘겨주고 다시 반복하게 한다. 이러한 원리를 근거로 제안된 MATRECO의 처리절차는 그림 12와 같다.

```
< procedure MATRECO >
1.  define  N      /* 망을 구성하는 노드들의 개수 */
            X      /* 결함발생노드 */
            Z,v,i  /* 초기화 */
            QUE    /* 크기가 k+1 인 큐 */
            NEW    /* [0..17] 인 일차원 배열 */
2.  los = X
3.  old = los
4.  los= parent(los)
5.  NEW[i] = CON[los,i]
6.  만일 NEW[i]( 0<=i<=n) = 1 인 노드 i 의 LE[n,1] 이
    현재노드(los)의 레벨보다 k+1 이상의 상위레벨에 존재할
    때는 NEW[i] = 0        /* 현재결함노드가 local
    supervisor 의 위치로 갈 때의 연결성 변화를 고려 */
7.  NEW[los] = 1     /* 현재결함노드의 parent 노드와의
    연결성 고려 */
8.  NEW[old] = 0     /* 결함노드 자체와의 연결성 */
9.  Z = Z+1
10. 만일 Z > k+1 이면  v = QUE[front], NEW[v] = 1
11. 만일 (0<=i<=n) CON[old,i] != NEW[i] 이면  CON[
    i,old] = NEW[i]
12. CON[old,i] = NEW[i]
13. 만일 los = 0 이면 exit
14. QUE[rear] = old goto 3
```

그림 12 MATRECO알고리즘

3.3 분산회복 알고리즘 수행 및 연결성 변화

이제 앞서 구축된 시스템을 대상으로 MATRECO 알고리즘이 어떻게 수행되는지를 살펴보자. 본문에서는 결함이 발생한 요소를 설정함에 있어 가능한 경우중 가장 복잡한 사례를 고려하기위해 루트에서 가장 멀리 떨어져 있는 16번 요소를 결함발생노드로 간주하였으며 따라서 이후의 모든 알고리즘의 진행이 16번 요소의 결함발생시에 근거하여 이루어진다. 다음은 그로인한 알고리즘의 첫 번째 수행시 각 단계에 따른 전개상황은 다음 그림 13과 같이 진행된다.

```
2.  los = 16
3.  old = 16
4.  los = 9
6.  NEW[2] = 0
7.  NEW[9] = 1
8.  NEW[16] = 0
9.  Z = 1  ( Z < k+1 )
12. CON[16,i] = NEW[i]
14. QUE[rear] = 16 goto 3
```

그림 13 첫 번째 수행단계

또한 알고리즘 수행에 따른 연결성의 변화를 쉽게 파악할 수 있게 하기위해 노드의 생성, 삭제 가 발생한 노드에 대하여는 모두 *를 이용하여 구분하였다. 그에 따라 진행된 해당노드에 대한 인접행렬의 변화과정이 그림 14에 나타나 있다.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
old(16)					1	1				1	1		1				
Los(9)			1		1	1					1		1				1
6 수행			*		1	1					1		1				1
7 수행					1	1				*1	1		1				1
8 수행					1	1				1	1		1				*
NEW[16]					1	1				1	1		1				

그림 14 첫 번째 수행시 인접행렬의 변화과정

이와같은 방식으로 두 번째부터 마지막인 다섯번째까지의 수행과정 및 각 수행단계에 따른 연결성 변화를 나타내는 인접행렬 표현을 그림 15부터 그림 22까지 나타내었다.

```
3.  old = 9
4.  los = 4
6.  NEW[1] = 0
7.  NEW[4] = 1
8.  NEW[9] = 0
```

9. Z = 2
11. CON[11,9] = 1
12. CON[9,i] = NEW[i]
14. QUE[rear] = 9 goto 3

그림 15 두 번째 수행단계

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
old(9)			1		1	1					1		1				1
los(4)		1	1			1				1	1	1					1
6수행		*	1			1				1	1	1					1
7수행			1		*1	1				1	1	1					1
8수행			1		1	1				*	1	1					1
NEW[9]			1		1	1					1	1					1
CON[11]			1		1					*1							

그림 16 두 번째 수행시 인접행렬 변화

3. old = 4
4. los = 2
6. NEW[0] = 0
7. NEW[2] = 1
8. NEW[4] = 0
9. Z = 3 (Z = k+1)
10. V = 16 , NEW[16] = 1
11. CON[i,4] = 1 (i = 6,12)
12. CON[4,i] = NEW[i]
14. QUE[rear] = 4 goto 3

그림 17 세 번째 수행단계

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
old(4)		1	1			1				1	1	1					1
los(2)	1	1			1	1	1			1	1	1	1				
5수행	*	1			1	1	1			1	1	1	1				
6수행		1	*1		1	1	1			1	1	1	1				
7수행		1	1		*	1	1			1	1	1	1				
8수행		1	1			1	1			1	1	1	1				*1
NEW[4]		1	1			1	1			1	1	1	1				1
CON[6]	1	1	1		*1												
CON[12]			1		*1	1					1						1

그림 18 세 번째 수행에따른 인접행렬 변화

3. old = 2
4. los = 1
7. NEW[1] = 1
8. NEW[2] = 0
9. Z = 4
10. V = 9 , NEW[9] = 1
11. CON[i,2] = 0 (i = 0,10,11,12)
    CON[j,2] = 1 (j = 3,7,8)
12. CON[2,i] = NEW[i]
14. QUE[rear] = 2 goto 3

그림 19 네 번째 수행단계

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
old(2)	1	1			1	1	1			1	1	1	1				
los(1)	1		1	1	1	1	1	1	1								
6수행	1	*1	1	1	1	1	1	1	1								
7수행	1	1	*	1	1	1	1	1	1								
8수행		1		1	1	1	1	1	1	*1							
NEW[2]		1		1	1	1	1	1	1	1							
CON[0]		1	*	1			1										
CON[3]	1	1	*1					1	1					1	1	1	
CON[7]		1	*1	1										1	1	1	
CON[8]		1	*1	1											1	1	
CON[10]			*		1					1		1	1				1

그림 20 네 번째 수행에따른 인접행렬 변화

3. old = 1
4. los = 0
7. NEW[0] = 1
8. Z = 5
10. v = 4 , NEW[4] = 1
11. CON[i,1] = 0 (i = 0,5,7,8)
12. CON[2,i] = NEW[i]
13. los = 0 이므로 loop exit

그림 21 다섯 번째 수행단계

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
old(1)	1		1	1	1	1	1	1	1								
los(0)		1	1				1										
6수행	*1	1	1				1										
7수행	1	*	1				1										
8수행			1		*1		1										
NEW[1]			1				1										
CON[0]		*	1				1										
CON[5]		*	1		1					1			1				1
CON[7]		*		1										1	1	1	
CON[8]		*		1											1	1	

그림 22 다섯 번째 수행에따른 인접행렬 변화

위에서 볼 수 있었던 바와 같이 시스템에서 한 요소가 결함을 일으켰을 경우 알고리즘의 회복단계의 수행이 루트노드에서 결함노드까지의 거리에 비례하여 필요한 횟수만큼 반복되는데 본문의 경우는 결함요소인 16번 노드가 중심요소인 루트노드로부터 다섯단계(깊이 5)의 거리를 가지므로 다섯번 수행되었다.

이상의 과정들에대한 변화를 모두 반영하여 최종적으로 완성된 연결성을 갖는 인접행렬 표현은 RESULT(X,Y)로 명명하였으며 그림 23에 나타나 있다.

RESULT(X,Y)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		1	1	1		1										
2	1		1	1	1	1	1	1	1							
3	1	1					1	1					1	1	1	
4	1	1			1	1			1	1	1	1				1
5		1		1		1			1			1				1
6	1	1		1	1											
7		1	1										1	1	1	
8		1	1											1	1	
9		1		1	1					1	1					1
10				1					1		1	1				1
11				1					1	1						
12				1	1					1						1
13			1				1							1		
14			1				1	1					1		1	
15			1				1	1						1		
16				1	1				1	1		1				

그림 23 완성된 연결성의 행렬표현

3.4 회복된 시스템 표현

앞절에서 제안된 MATRECO의 수행으로 기존의 시스템이 결함요소를 발생시켰을 경우 단계적절차를 거쳐 최종적으로 그림 19와 같은 인접행렬을 산출할 수 있음을 보았으므로 이절에서는 인접행렬의 표현을 토대로 노드간의 연결성을 재구성하여 궁극적으로 결함발생후 분산적 회복 과정을 통하여 재구성 되어진 회복시스템을 완성한다. 초기 시스템 모델에 인접행렬의 연결성을 적용한 결과가 그림 24에 나타나 있다. 이렇게 구성된 시스템은 결함발생시 분산회복이 가능한 것이므로 결함을 허용할 수 있는 시스템이라 볼 수 있다.

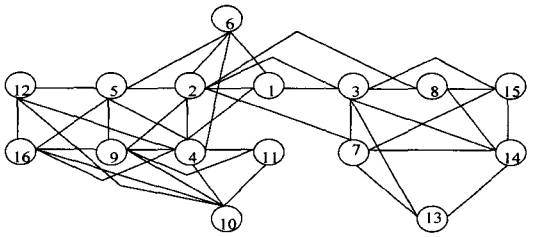

그림 24 회복된 시스템 구성

4. 결 론

시스템의 구성요소가 손상되면 그에 따르는 작업량도 줄어들게 되며 전체적인 시스템의 효율적 운영도 지장을 초래하게 된다. 따라서 구성요소가 결함을 발생시킨 경우 그에 대한 신속한 조치를 취하여야만 전체적 처리 효율에 별 무리를 주지 않는정도[14]에서 자연스러운 시스템 회복이 가능할 수 있게 된다. 본문에서는 결함이 발생했을 경우 결함발생에 대비한 망의 연결성을 고려한 시스템에서의 단계적 분산회복을 가능하게 하는 알고리즘을 제시했는데 이것은 기존의 트리나 루프형태의 특정모델에만 국한되어 적용가능했던 방식과는 달리 일반적인 망모델을 적용대상으로
만들어진 알고리즘이다. 그러므로 이제까지의 연구들이 특수형태들을 대상으로 하고 있었다는 점에 비하여 본 논문은 보다 포괄적인 유형의 망들에 대한 새로운 분산회복 방식을 설정하였으며 그로인해 적용대상에 대한 응용성도 높였다고 할 수 있다.

본문에서는 한 순간에 하나의 결함이 발생하는 것을 기본적인 입장으로 고려하였으나 동시에 여러개의 결함이 발생하는 경우에 대해 우선순위를 부여하며 배타적인 처리를 하는 방안과 병렬처리가 가능한 알고리즘을 적용하는 연구가 진행중에 있으며 더 나아가 병렬처리중 실시간 적응처리가 가능한 알고리즘도 연구되어야 할 것이다. 아울러 이러한 알고리즘들은 논문에서와 같이 포괄적인 상태가 아닌 구성요소들의 세부적 상황이 주어진 여건하에서는 보다 단순화 될 수도 있으리라고 기대된다.

참 고 문 헌

[1] Robert Geist and Kishor Trivedi, "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques," IEEE Trans. on Conputer, pp 52-56, 1990.

[2] Ernst J.Schmitter and Peter Baues, "The Basic Fault-Tolerant System," IEEE Micro, pp. 66-74, 1984.

[3] Victor P.Nelson, "Fault-Tolerant Computing: Fundamental Concepts," IEEE Trans. on Computer, pp. 19-25, 1990.

[4] David A. Rennels, "Fault-Tolerant Computing - Concepts and Examples," IEEE Trans. on Computer, vol.c-33, no.12, december 1984.

[5] D.M. Blough and A. Pelc, " Almost Certain Fault Diagnosis Through Algorithm-Based Fault Tolerance, " IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 5, pp 532-539, May 1994.

[6] Ghassem Miremadi,Johan Karlsson, "Two Software Techniques for On-line Error Detection , " FTCS International Symposium on Fault-Tolerant Computing, pp328-335, July 1992.

[7] Tim Olson, " Fault-Tolerant Chips Increase System Reliability, " Computer Design, March 15, 1986.

[8] Alsirdas Avizienis and John P.J.Kelly, " Fault-Tolerance by Design Diversity : Concepts and Experiments, " IEEE Computer, August 1984.

[9] Danial P.Siewiorek, "Architecture of Fault-tolerant Computers:An Historical Perspective," proceedings of the IEEE,vol.79,no.12,december 1991.
[10] Danial P.Siewiorek, " Architecture of Fault-Tolerant Computers, " IEEE Computer, August 1984.
[11] B. Vinnakota and N. K. jah, " Diagnosability and Diagnosis of Algorithm-Based Fault Tolerant Systems, " IEEE Trans. on Computers, Vol. 42, No. 8, pp 924-937, Aug. 1993.
[12] A.S.Tenenbaum, "Computer Networks," Englewood Cliffs, NJ:Prentice-Hall, 1981.
[13] Y. H. Moon, and B. K. Kim, " A Study of Fault-Tolerant System construction Algorithm in General Network topology, " The journal of Korean Institute of Communication Sciences, Vol. 23, No. 6, June 1998.
[14] Shalini Yajnik, Niraj K. Jah, " Graceful Degradation in Algorithm-Based Fault Tolerant Multiprocessor Systems, " IEEE Trans. on Parallel and Distributed Systems, Vol. 8, No. 2, February 1997.
[15] S. Yalnik. N. K. Jah, " Analysis and Randomized Design of Algorithm-Based Fault Tolerant Multiprocessor Systems Under the Extended Graph-Theoretic Model, " Proc. Isca Parallel and Distributed Computing & Systems, pp. 52-27, Louisville, Ky., Oct. 1993.
[16] Raif M. Yanney and John P. Hayes, " Distributed Recovery in Fault-Tolerant Multiprocessor Networks, " IEEE Trans. on Computer, Vol. C-35, No. 10, October 1986.

문 윤 호
1985년 전북대학교 전산통계학과 학사. 1987년 숭실대학교 전산학과 석사. 1998년 숭실대학교 전산학과 박사과정. 1998년 전주기전여자대학 컴퓨터과 부교수. 관심분야는 분산처리, 결함허용시스템, 컴퓨터구조

김 병 기
1977년 서울대학교 전자공학과 공학사. 1979년 KAIST 전산학과 이학석사. 1979년 경북대학교 전산학과 전임강사. 1998년 KAIST 전산학과 박사. 1998년 숭실대학교 정보과학대학 교수. 관심분야 ATM, 이동데이타통신, 컴퓨터구조