하이퍼큐브를 이용한 결함 허용 라우팅 알고리즘

정회원 최 병 환*, 강 성 수**, 이 충 세***

Fault-Tolerant Routing Algorithm in Hypercube Multicomputers

Byung-whan Choi*, Sung-soo Kang**, Chung-sei Rhee*** ***Regular Members***

요 약

하이퍼 큐브는 정규적이며 결함 허용 능력을 갖고 있기 때문에 결함 허용 알고리즘을 구현하기 쉽다. 하이퍼 큐브를 이용한 결함 허용 알고리즘들이 많이 개발되었다. 이러한 알고리즘들 중에 안전(safe)과 불안정(unsafe) 개념을 이용한 알고리즘이 Masuyama등에 의해 개발되었다. 이 논문에서는 Masuyama의 알고리즘을 개선한 새로운 알고리즘을 제안하고 시뮬레이션을 통하여 성능을 비교한다.

Key Words : Hypercube, Fault Tolerant, Multocomputer, Routing.

ABSTRACT

Hypercube has a capability of fault-tolerance and regularity, which is easy to develop an algorithm. Many algorithms have been developed as an efficient fault-tolerance routing algorithm using hypercube. Among these algorithms, a method which use safe and unsafe concept was developed by Masuyama. Masuyama suggested an enhanced algorithm that take advantage of unsafe-safe concept. In this paper, we propose an algorithm that uses the unsafe, safe concept and modify Masuyama's algorithm. Using simulator we compare the performance of the proposed algorithm with existing algorithms.

Ⅰ. 서 론

네트워크에서의 주요 문제 중 하나는 라우팅을 얼마나 효율적으로 하는가이다. 그러나 라우팅이 효율적이라고 하더라도 네트워크상에 결함이 발생할 경우 라우팅이 실패하는 경우가 있다. 따라서 네트워크상에 이러한 결함이 발생하더라도 다른 정상적인 노드사이의 통신은 지속될 수 있는 라우팅 방법이 필요하다. 이러한 라우팅을 결함허용 라우팅이라고 한다.

결함의 발생빈도는 노드 수, 즉 네트워크의 크기에 비례한다. 따라서 네트워크의 크기가 커질수록 결함허용 능력이 중요한 문제가 된다. 여러 네트워크 구조 중 이러한 결함 허용 능력이 뛰어난 구조는 하이퍼큐브 네트워크이다[10].

이러한 하이퍼큐브 네트워크에서 좀 더 효율적인 결함허용 라우팅을 위한 알고리즘이 필요하다. 또한 하이퍼큐브는 계층적인 구조의 특징을 가지고 있어 알고리즘 개발에 유리하다는 점도 있다. 이 논문에서는 이러한 하이퍼큐브에서의 결함허용라우팅 알고리즘을 제안한다.

하이퍼큐브에서의 라우팅 알고리즘은 지역정보기반(local-information-based)([1],[4],[6]),전역정보기반(global-information-based)[9],제한전역정보기반(limited-global-information-based) ([2],[3],[5],[7],[8],[11],[12],[13])으로 나눌 수 있다.

* 충북대학교 컴퓨터과학과 알고리즘 연구실 (cbworld@kornet.net)
** 충북대학교 컴퓨터과학과 알고리즘 연구실 (kendo100kr@naver.com)
*** 충북대학교 컴퓨터과학과 교수 (csrhee@cbu.ac.kr)
논문번호 : KICS2005-01-041 접수일자: 2005년 1월 21일

전역정보기반은 각 노드가 모든 노드의 정보를 가지고 라우팅을 하는 방법이고 지역정보기반은 이웃노드의 정보만을 가지고 라우팅하는 방법이다.

제한정보기반 라우팅방법은 지역정보와 전역정보의 장, 단점을 절충한 라우팅 방법으로 각 노드는 이웃노드의 정보만을 유지하지만 이웃노드의 정보는 지역기반의 이웃노드 정보보다 더 많은 정보를 가지고 있다. 각 노드가 더 많은 정보를 갖기 위해 일정 라운드 동안 이웃노드의 상태를 조사하고 이러한 정보를 유지한다. 최적라우팅은 얻은 정보를 기반으로 수행되며 그렇지 않을 경우 우회하여 라우팅을 수행하여 라우팅 실패를 최대한 피하게 된다.

이 논문에서는 하이퍼큐브의 결함허용 라우팅 알고리즘 중에서 제한전역정보기반의 알고리즘을 사용하고 기존의 방법을 개선하여 최단경로를 더 많이 찾는 방법을 제안한다. 제한정보기반 알고리즘의 성능을 알아보기 위해 기존의 방법과 제안된 방법의 라우팅 성공 횟수와 실패 횟수를 비교하기위한 시뮬레이션을 수행하였다. 여기서 경로를 찾는 속도는 고려하지 않는다. 또한 유니캐스트(unicast) 라우팅만 고려하였다.

Ⅱ. 관련 연구

n차 하이퍼큐브는 0번부터 2^n-1까지의 숫자로 번호가 부여된 2^n개의 노드가 자신의 노드 번호와 서로 1비트만 다른 노드끼리만 연결되어있는 그래프이다. 즉, n차의 하이퍼큐브가 있을 때 노드의 수 $N=2^n$이다. 하이퍼큐브에서 각 노드를 이진수로 표현한다. 즉, 노드를 $u=u_{n-1}u_{n-2}...u_0$로 표현하고 여기서 $u_i \in \{0,1\}$이고 $i \in \{0,1,...,n-1\}$이다. u_i는 i번째 비트를 나타낸다. 노드 u의 이웃노드를 u^i로 표현하는데 이것은 현재 노드의 i번째 비트를 변화시킨 노드를 의미한다. 또한 u^{ij}는 u^i노드에서 j번째 비트를 변화시킨 노드를 의미한다.

예를 들면, 4차원 하이퍼큐브에서 임의의 노드 u의 번호를 13이라고 하면 이진수로 $1101(1_3 1_2 0_1 1_0)$로 표현된다. 각 비트는 괄호 안에 표기한 것처럼 가장 오른쪽을 0번째 비트로 하고 왼쪽으로 가면서 첫 번째 비트 두 번째 비트라고 부른다. 1101의 이웃노드는 비트 자릿수만큼 존재하므로 이 예에서는 4개의 이웃노드가 존재하며 1101의 이웃노드는 각각 1101^0,1101^1, 1101^2, 1101^3으로 표현하고 각 표현이 의미하는 노드는 1100, 1111, 1011, 0101로써 1101과 각각 한 비트씩 반전된 노드들이다. 1101^{13}가 의미하는 것은 1111^3, 즉 노드 0111을 의미한다.

현재노드를 u라 하고 목적지 노드를 d라고 하면 두 노드사이의 최단거리는 $u \oplus d$의 결과의 1의 개수와 같다. 이를 H(u,d)로 표현하며 해밍거리(hamming distance)라고 부른다. $u \oplus d$의결과 1에 해당되는 비트의 위치를 p라 하면 u^p를 선호하는(preferred) 이웃노드라고 정의하고 나머지 노드는 여분의(spare) 이웃노드라고 정의하고 u^s로 표현한다.

예를 들면, 소스 노드를 1101, 목적지 노드를 0110이라 할 경우 1101⊕0110= 1011이 되고 1011의 1의 숫자는 3개이다 따라서 H(1101,0110)= 3이 되고 두 노드 사이의 최단 거리가 3이 됨을 의미한다. 즉 3번의 비트를 변화시켜 도달할 수 있음을 의미한다. 1101과 0110의 XOR연산 결과인 1011에서 1의 위치는 0번째, 1번째, 3번째이므로 1100(1101^0), 1111(1101^1), 0101(1101^3)번 노드가 선호하는 이웃노드가 되고 나머지 노드 1001(1101^2)은 여분의 이웃노드가 된다.

선호하는 이웃노드를 거쳐 라우팅을 하게 되면 그 경로는 최단 경로(shortest path)가 되고 여분의 이웃노드를 거쳐 라우팅을 하게 되면 우회하게 되는데 한번 우회할 때마다 거리는 H+2 가 된다. 즉, 최단경로의 2만큼 거리가 멀어지게 된다.

그림 1은 3차원 하이퍼큐브를 보여준다. 각 노드는 3비트로 구성되어 있고 이웃노드는 각각 한 비트씩 다른 것을 보여주고 있다. 노드 000의 이웃노드는 100,010,001 같이 한 비트씩 다른 노드가 되고 그 개수는 3개가 된다. 000에서 011로 라우팅할 경우 H(000,011)=2가 되고 u^p={010,001}이 되고 u^s={100}이 된다.

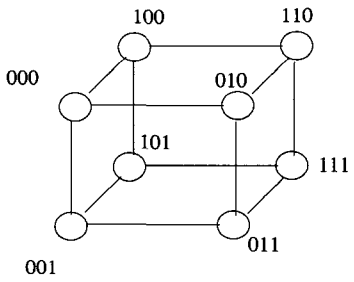


그림 1. 3차원 하이퍼큐브

2.1 안전, 불안전 노드

제한전역정보 기반 알고리즘으로 불안전(unsafe),

안전(safe)개념을 이용한 알고리즘들이 제안되었다([2],[5]). 이 방법은 n차의 하이퍼큐브일 경우 n개의 이웃노드의 상태를 조사하여 자신의 노드의 상태를 결정하는 방법이다.

각 노드는 안전(safe), 보통-불안전(ordinary unsafe), 강-불안전(strongly unsafe), 결함(fault)이라는 네 가지의 상태를 갖고 안전한 노드로 우선적인 라우팅을 하게 된다. 정의 1과 정의 2는 불안전한 노드와 안전한 노드에 관한 정의를 나타낸다.

정의 **1.** 이웃에 두개 이상의 결함 노드가 존재하거나 세 개 이상의 불안전 노드나 결함 노드가 존재할 경우 불안전한 노드라고 정의한다.

정의 **2.** 이웃노드 중에 안전 노드가 존재하지 않으면 강-불안전(strongly unsafe) 노드라고 하고 그렇지 않을 경우 보통-불안전(ordinarily unsafe) 노드라고 정의한다.

알고리즘으로 표현하기 위해 용어를 다음과 같이 정의한다.

SAFE: 안전 노드 목록
O_UNSAFE: 보통-불안전 노드의 목록
S_UNSAFE: 강-불안전 노드의 목록
l_{ou}: 보통-불안전 노드의 수
lsu: 강-불안전 노드의 수
l_f: 결함 노드의 수

알고리즘 Iden_Unsafe()

```
begin
l_ou = 0; l_su = 0;
while (system status is not stable) begin
    for j=1 to n begin
        Receive status Sx from neighbor X
        connected by dimension j;
        if( Sx= ordinary unsafe and X is not in
          O_UNSAFE) then
        begin
            Add X to O_UNSAFE;
            Delete X from SAFE;
            l_ou := l_ou+1;
        end
        if( Sx= strongly unsafe and X is not in
          S_UNSAE) then
        begin
            Add X to STRONGLY UNSAFE;
            Delete X from SAFE or UNSAFE
            l_su := l_su +1
        end
        end
    end
    if( l_f ≧ 2 or l_f+l_u ≧3) then begin
        if( l_f+l_u = n ) then begin
            Mark current node as strongly unsafe;
        else
            Mark current node as ordinary unsafe;
        end
    end
end
```

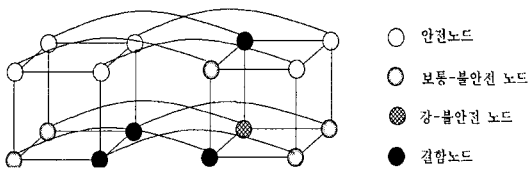


그림 2. 안전, 불안전 노드의 표시

그림 2는 4차원 하이퍼큐브에서 각 노드에서 정의 1과 정의 2를 적용한 결과를 보여준다. 이런 결과를 얻기 위하여 4회의 라운드가 요구된다. 라운드는 라우팅 전에 모든 노드가 이웃노드의 정보를 얻는 횟수를 의미한다. 즉 1라운드는 각 노드가 이웃의 모든 정보를 얻는 것을 의미한다.

노드의 안정도는 안전 노드, 보통-불안전 노드, 강-불안전 노드, 결함 노드 순으로 낮아진다. 라우팅은 안전도가 높은 노드를 우선적으로 택하여 라우팅하게 된다. 라우팅 방법을 간략히 정리하면 다음과 같다.

2.2 안전, 불안전 노드를 이용한 라우팅방법

다음과 같이 용어들을 정의한다.

i, j: 하이퍼큐브상의 노드,
u: 현재노드,
d: 목적지노드,
s: 소스노드,
H(i,j): i, j사이의 해밍거리
$N(i) = \{j | H(i,j) = 1\}$ (노드 i의 이웃노드들의 집합)

$D(i,d) = \{j | H(j,d) = H(i,d) - 1, j \in N(i)$
(노드 i에서 목적지 t로 가기 위한 선호하는 노드들의 집합)
l = H(u,d) (해밍거리)
N= N(s) (이웃노드들의 집합)
D= D(s,d) (선호하는 이웃노드들의 집합)
S= 안전 노드의 집합
O= 보통-불안전 노드의 집합
U= 강-불안전 노드의 집합

위의 정의를 이용하여 라우팅 알고리즘은 다음과 같이 작성한다.

알고리즘 ROUTE(s, t):

---

begin
if $l = 0$ then deliver the message to s and exit
else if there exists $j \in D \cap S$ then take node j
else if there exists $j \in D \cap O$ then take node j
else if there exists $j \in D \cap U$ and
($s \in U$ *or* $l \leq 2$)
then take node j
else if there exists $j \in (N - D) \cap S$ then take node j
else if there exists $j \in (N - D) \cap O$ then take node j
else there exist no route
ROUTE(j, t)
end

그림 3은 이 방법을 이용한 라우팅을 보여주고 있다.

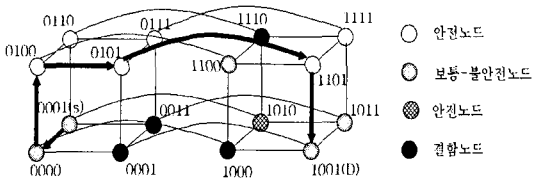


그림 3. 안전, 불안전을 이용한 라우팅

2.3 Masuyama가 제안한 알고리즘

불안전, 안전 개념을 이용한 라우팅의 단점은 최단경로가 존재함에도 불구하고 경로를 우회할 수 있는 데에 있다. 따라서 더 많은 최단경로를 발견하기 위해 Masuyama는 소스와 목적지를 포함하는 가장 작은 서브큐브를 구성하여 그 서브큐브에서 안전, 불안전노드를 다시 계산하여 라우팅하는 방법을 제안하였다[8].

만약 소스와 목적지를 포함하는 가장 작은 서브큐브를 구성하지 못하면 기존의 방법대로 라우팅한다. 이 방법은 소스와 목적지 이외의 결함은 라우팅하는데 아무런 영향을 미치지 않음을 이용한 방법이다. 즉 많은 수가 노드가 분포해 있을 때 특정지역에서의 라우팅은 멀리 떨어진 곳의 노드의 상태와는 아무런 관련이 없기 때문이다.

이 방법은 소스에서 목적지를 포함하는 서브큐브 내부의 결함만 고려하기 때문에 결함 수를 상대적으로 줄일 수 있기 때문에 확실히 노드의 안전도는 높아지고 더 많은 수의 최단경로를 구할 수 있다.

알고리즘으로 표현하면 다음과 같다.

---

begin
ROUTE(s, t) for the smallest sub-cube which contains s and t
if there exist no route, then ROUTE(s, t) for the fully hypercube
end

그림 2와 같이 각 노드의 상태가 분포해 있을 경우 안전, 불안전 노드를 구별하는 방법으로 라우팅을 할 경우 그림 3과 같은 결과를 얻는다. 최단경로 0001 --> 1010 --> 1011 --> 1001이 존재하지만 1010이 강-불안전노드이기 때문에 보통-불안전노드로 우회하게 된다.

이를 해결하기위해 소스와 목적지를 포함하는 최소 서브큐브를 구성하여 안전, 불안전노드를 그 서브큐브 내에서만 다시 결정하면 그림 4처럼 된다. 그 다음 다시 기존의 라우팅 방법(안전, 불안전에서의 라우팅 방법)을 적용하면 최단경로 라우팅이 가능하다.

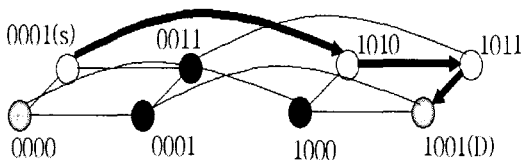


그림 4. 소스와 목적지를 포함한 최소하이퍼큐브

Ⅲ. 제안하는 알고리즘

3.1 Masuyama방법의 문제점

Masuyama가 제안한 방법은 소스나 목적지가 바뀔 때마다 매번 안전, 불안전 노드를 다시 판별해야 하는 단점이 있다. 따라서 매 라우팅 시 계산시간이 많이 걸리게 된다. 이를 해결하기 위해 기존의 방법(안전, 불안전 노드를 사용하는 방법)과 Masuyama의 서브큐브를 이용하는 방법을 결합하여 보다 개선된 라우팅 방법을 제안한다. 제안하는 방법은 소스나 목적지가 변하더라도 다시 안전, 불안전 노드의 구별을 할 필요가 없다.

3.2 이웃노드 정보관리

새로운 알고리즘에서의 정보수집방법은 기존의 안전, 불안전노드 방식을 약간 수정한다. 수정한 방법은 다음과 같다.

i. 기존의 방법대로 노드의 상태를 불안전, 안전 노드로 구분한다.
ii. 결정된 불안전, 안전 상태를 수치로 표현한다. (안전 노드: 0, 보통-불안전 노드: 1, 강-불안전 노드: 2, 결함 노드: 3)
iii. 이웃노드의 정보를 n비트의 배열에 저장한다. 예) (0,1,3,0)은 $u^0 = u^3$= 안전 노드, u^2=보통-불안전 노드, u^1= 결함 노드를 의미한다.
iv. 정보를 이웃노드와 교환한다.

알고리즘 STATE()

```
---------------------------------------------------------------
begin
  Iden_Unsafe()
  for i=0 to number of node begin
    if ( Sx == SAFE) then Sx= 0
    else if ( Sx == O_UNSAFE) then Sx= 1;
    else if ( Sx == S_UNSAFE) then Sx= 2;
    else Sx= 3;
  end
  Save the information of neighbor_Sx in
    array and make a status_table
  Exchange the status table
end
---------------------------------------------------------------
```

3.3 라우팅 방법

2절에서 저장된 이웃노드들의 정보를 가지고 라우팅하는데, 라우팅은 최단경로를 발견할 경우와 그렇지 못할 경우 우회해서 갈 수 있게 하였다.
u를 현재 노드, d를 목적지 노드라고 가정하고 다음과 같이 알고리즘을 작성한다.

(a) 최단경로

노드 u에서 $u^i \in$ {preferred neighbor nodes}와 $u^{i,j} \in$ {preferred neighbor nodes}를 만족시키는 $u^{i,j}$에 해당되는 노드들의 상태정보를 수집하여 더한다. 그 합을 sum이라 하고, sum<=2 이면 바로 u^i로 라우팅을 한다. 그렇지 않다면 2<sum<H(u^i,d)×3인 u^i의 노드 중에서 sum의 값이 최소인 노드로 라우팅한다. 이 조건을 만족하지 못하면 우회하여 라우팅한다.

(b) 우회경로

(a)의 최단경로 조건을 만족하지 못하면 u^i에 해당되는 모든 노드의 sum값을 조사해서 최소값을 갖는 노드로 라우팅한다.

(c) 라우팅 실패

(a), (b) 두 라우팅 조건을 만족 못하면 라우팅은 실패하게 된다.

예) 소스-노드(S): 0010,
목적지-노드(D): 1001

그림 5는 기존의 방법을, 그리고 그림 6은 제안한 방식을 나타낸다. 같은 소스 노드와 목적지 노드를 가지고 있지만 제안한 방식은 최단 거리를 이용한 라우팅을 하고 있음을 알 수 있다. 그림 2의 라우팅순서는 다음과 같다.

0010의 선호하는 이웃노드는 {1010,0000}이 된다. 1010⊕1001 = 0011 이고 여기서 11에 해당되는 비트 즉 0번째 비트, 1번째 비트가 되는데 (1,3,3,1)에서 해당 비트의 합을 구한다. 여기선 3+1= 4가 된다. (a)의 최단경로 조건에서 2보다 크므로 다음 노드도 계산하게 된다. 0000⊕1001=1001이 되므로 {3,0,1,3}에서 0번째와 3번째에 해당되는 비트를 더하면 3+3= 6이 된다. 이것 또한 2보다 크므로 바로 라우팅이 안 되고 1010의 값 4와 비교해서도 큼으로 무시된다. 1010의 4의 값은 (a)의 최단경로 조건 2보다 크고 H(1010,1001)×3=6 보다 작으므로 라우팅이 가능하다.

같은 방법을 목적지 노드까지 적용한다. 이렇게 함으로써 기존의 알고리즘에서 우회하던 것을 제안

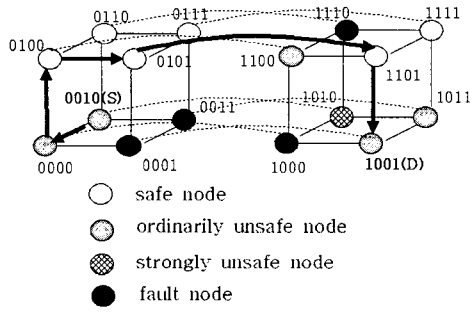


그림 5. 기존의 방법(불안전, 안전방식)

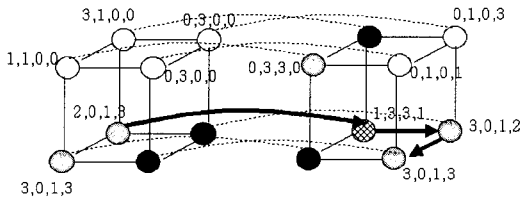


그림 6. 제안한 방법

한 알고리즘에서는 최단경로로 라우팅이 가능하게 한다.

알고리즘 Routing()

```
---------------------------------------------------------------------
begin
    for i= 0 to n begin   //n: dimension
        if (u^i == preferred neighbor) then begin
            xor =  u^i  XOR target
            sum  =  sum  of  information  value(bit
location corresponding '1' of xor) in table at u^i
            if sum <=2 then begin routing to the u^i
        end
        end
    end
    for i = 0 to n begin
      if sum< H(u^i,d)×3 then begin routing to the u^i
    end
    end
    detour()    //same method using spare neighbor
end                 //route to the node which has the
smallest value
---------------------------------------------------------------------
```

Ⅳ. 실험 및 평가

4.1 실험 개요

이 실험은 라우팅의 속도 즉 계산과정에 걸리는 시간은 고려하지 않고 단지 최단 경로를 얼마나 많이 찾는가에 관해 알아본다. 소스와 목적지의 위치를 랜덤하게 바꾸며 실험을 수행했고 같은 소스와 목적지가 나오지 않도록 고려하였으며 소스와 목적지 사이의 거리는 제약을 두지 않았다. 즉 소스와 목적지 사이의 거리가 먼 경우와 가까운 경우를 따로 나누워 시뮬레이션하지 않고 랜덤하게 정했다.

4.2 실험 방법

실험은 두 가지 방법으로 행했다. 우선 차수에 따른 우회경로 감소율을 조사했다. n차원 하이퍼큐브에서 항상 n-1개의 결함이 발생한다고 가정하고 하이퍼큐브의 차수를 증가시키면서 실험했다. 그 다음으로 차원을 변화시키지 않고 결함 수를 증가시킴으로써 목적지까지 우회하는 횟수를 조사했다. 위의 두 가지 방법을 소스와 목적지를 바꾸어 가며 각각 10만회씩 수행을 해서 우회 없이 목적지에 도달한 횟수를 조사했다.

4.3 실험 결과 및 평가

기존의 방법과 제안한 방법의 차원의 증가에 따른 우회경로의 감소율과 임의의 차원에서 결함수의 증가에 따른 우회경로의 감소율을 시뮬레이션 하였다.

표 1. 차수에 따른 우회경로 감소율

방법 \ 차원	4	5	6	7	8	9	10
기존 방법	1304	571	173	54	10	1	0
제안한 방법	1304	549	157	45	10	1	0
감소율	0%	3.9%	9.3%	16.7%	0%	0%	0%

표 1은 차수의 증가에 따른 우회경로 감소율을 보인 것이고 이때 결함 수는 n-1개로 한다. 이유는 이웃노드가 모두 결함이면 그 노드는 네트워크로부터 단절되기 때문이다. 결과는 5~7차 까지는 감소율이 증가했으나 4차원과 8~10차원은 변화가 없었다. 이유는 네트워크상에서 결함노드가 차지하는 비율과 관련이 있다.

4차원에서 3개의 결함이 발생하는 경우와 10차원에서 9개의 결함 발생하는 경우 결함이 차지하는 비율을 비교해보면 전자의 경우는 37.5%, 후자의 경우는 0.88%를 차지한다.

따라서 전자의 경우 즉 4차원인 경우 라우팅 알고리즘과 상관없이 선택할 경로가 적기 때문이고

후자의 경우인 10차원의 경우는 결함이 인접해서 발생할 확률이 적기 때문에 우회할 가능성이 거의 없다.

이 결과는 여러 라우팅 알고리즘에서 n-1개로 제한된 상태에서 라우팅 방법을 제시한 알고리즘들이 고차로 갈수록 비효율적임을 보여준다.

표 2. 7차원에서 결함 수 증가에 따른 우회경로 감소율

차원 / 방법	6 (4.69%)	7 (5.47%)	8 (6.25%)	9 (7.03%)
기존 방법	54	64	76	156
제안한 방법	45	52	63	106
감소율	16.7%	18.8%	17.11%	32.05%

표 3. 8차원에서 결함 수 증가에 따른 우회경로 감소율

차원 / 방법	7 (2.73%)	8 (4.3%)	9 (4.6%9)	10 (5.08%)
기존 방법	10	36	40	73
제안한 방법	10	26	31	44
감소율	0%	27.8%	22.5%	39.73%

표 2와 표 3은 각각 7차원과 8차원에서 결함수를 증가시키면서 시뮬레이션한 결과이다. 결과는 결함이 많이 발생할수록 우회를 많이 하고 그에 따라 새로 제안된 방법의 우회경로를 감소시키는 비율도 증가함을 보여준다.

표에서 나타낸 경우보다 더 많은 결함을 발생시키면 기존방법에서 라우팅 실패하는 경우가 발생하며 비교를 하지 않았다. 즉, 기존의 방법에서 라우팅의 실패가 발생할 경우에도 새로 제안된 방법에서는 라우팅 실패가 발생하지 않았다. 따라서 제안된 방법은 많은 결함이 발생할수록 효율성이 좋아짐을 보이고 있다.

그림 7은 기존의 방법에서 우회하는 경우에도 제안한 방법에서는 최단 경로로 라우팅하는 예를 두 개 보여준다.

첫 번째 예에서 소스 노드(10001)에서 다음으로 라우팅할 노드를 선택해야하는데 라우팅 가능한 두 개의 선호하는 이웃노드의 상태가 보통-불안전 노드이다. 이렇게 상태가 동일할 경우(모든 노드가 안전할 경우도 같음)는 알고리즘 설계 시에 미리 설정한 라우팅 원칙을 따라야한다.

이 논문 전체에서는 가장 오른쪽 비트를 0비트로 정의했다. 이러한 정의에 맞추어, 이런 경우와 같이 상태가 동일한 경우는 가장 오른쪽 비트를 선택하게끔 정했다. 이런 원칙에 의해 기존의 방법에서는 우회하게 된 것이다. 만약 가장 왼쪽부터 선택하게 했다면 기존의 방법도 최단 경로를 선택했을 것이다. 그러나 중요한 사실은 왼쪽부터 선택하게 원칙을 세워서 이번경우에 최단경로를 찾는다고 해도 또 다른 상황에서는 왼쪽부터 선택하는 원칙 때문에 우회하는 경우가 발생할 수 있는데 있다.

그러나 제안한 방법에서는 왼쪽부터 선택하건 또는 오른쪽부터 선택하건 또는 임의의 순서대로 선택하게 설정하더라도 항상 최단 경로를 찾는 데 있다.

두 번째 예의 경우는 소스 노드(10100)에서 다음 노드 10110노드까지는 같은 경로를 택하지만 10110 노드에서 다음 노드로 향할 때 기존방법은 10111노드가 강-불안전 노드이므로 보통-불안전 노드인 11110을 선택했다. 11110도 선호하는 이웃노드이기 때문에 최단 경로이긴 하지만 그 다음 노드를 선택할 때 우회를 하게 된다.

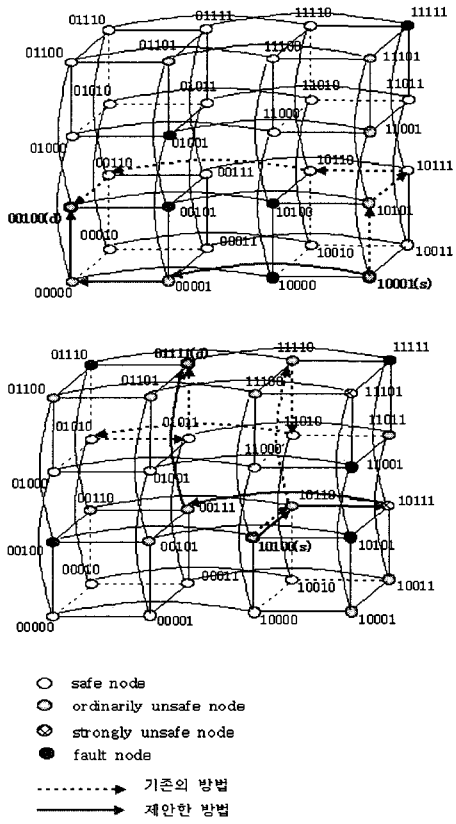


그림 7. 기존 방법과 제안한 방법의 라우팅 경로 비교

반면에 제안한 방법에서는 10110노드에서 강-불안전 노드를 선택하여 라우팅을 하게 된다. 이는 다음으로 라우팅할 노드를 선택함에 있어서 불필요한 정보를 무시하고 계산하기 때문이다.

V. 결 론

멀티컴퓨터의 내부 프로세서 사이의 통신이나 분산 네트워크상에서의 통신이나 통신 주체가 되는 노드들은 계속적으로 증가하고 있다. 그러나 이런 환경에서 중요한 것은 하나의 노드에 결함이 발생할 경우 전 통신과정이 마비되는 데에 있다. 이런 상황을 처리하기 위해 결함이 발생하더라도 전체 시스템의 통신이 유지되도록 하는 방법이 필요하다.

특히 네트워크의 크기가 커질수록 그에 따른 결함의 발생 확률 또한 커지므로 대규모 네트워크에서는 결함허용 라우팅이 필수적인 요소가 되어가고 있다. 이러한 결함허용라우팅의 가장 중요한 점은 대체 경로를 찾고 가능한 한 가장 빠른 경로를 찾아야 한다.

이 논문에서는 기존의 방법에 비해 더 많은 최단 경로로 라우팅을 가능하게 할 뿐만 아니라 많은 결함이 발생하더라도 라우팅 실패확률을 줄여주는 방법을 제시하였다. 이 방법의 단점은 기존의 방식에 비해 n차원에 대해 n배 만큼의 메모리를 더 요구한다는 것이다. 그러나 계산속도면에서는 제안한 방법이 복잡해 보이지만 기존방법에서도 매 라우팅 시 선호하는 이웃노드를 찾고 또한 각 노드의 상태를 살피기 때문에 제안한 방법에서 추가된 연산 부분은 시간에 거의 영향을 주지 않는다.

앞으로 수집된 데이터의 효과적인 저장방법을 연구하여 요구되는 메모리의 크기를 줄이는 연구와 함께 라우팅의 최단거리와 우회를 결정하는 sum값 등의 제한 조건을 변화시키며 실험을 행해 최적의 경계조건을 연구하려고 한다.

참 고 문 헌

[1] M. S. Chen and K.G. Shin, "Depth-First Search Approach for Fault-Tolerant Routing in Hypercube Multicomputers", *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 2 pp. 143-156, Feb 1996

[2] G. M. Chiu and S.P. Wu, "A Fault-Tolerant Routing Strategy in Hypercube Multicomputers," *IEEE Trans. Computers*, Vol. 45, no.2, pp. 143-156, Feb 1996

[3] Ge-Ming Chiu, Shui-Pao Wu, "A Fault-Tolerant Routing Strategy in Hypercube Multicomputers," *IEEE Trans. Computers*, Vol 45, Issue 2, page 143-155, Feb 1996

[4] Y. Lan, "A Fault-Tolerant Routing Algorithm in Hypercubes," *Proc. 1994 Int'l Conf. Parallel Processing*, pp. II 163-II 166, Aug 1994

[5] T. C. Lee and J. P. Hayes, "A Fault-Tolerant Communication Scheme for Hypercube Computers" *IEEE Trans. Computers*, vol. 41, no. 10, pp.1242-1256, Oct 1992

[6] J. M. Gordon and Q. F. Stout, "Hypercube Message Routing in the Presence of Faults", *proc.Third Conf. Hypercube Concureent Computers and Applications*, page 251-263, Jan 1988

[7] Zhen Jian, Jie Wu, "A Limited-Global Information Model for Dynamic Fault-Tolerant Routing in Cube-Based Multicomputers," *Network Computing And Applications, 2003 NCA*, Second IEEE International Symposium on, page 330-340, 16-18 April 2003

[8] Masuyama, H. Kawasaki, T. Kawamura, T. "A Fault-Tolerant Routing Algorithm for Hypercube Multi-Computers Systems", *Man and Cybemetics, 2002 IEEE International Conference On*, Vol 2, page 629-633, Oct. 6-9, 2002

[9] C. S. Raghavendra, P. J. Yang "Free Dimension An Effective Approach to Achiveing Fault Tolerance in Hypercubes", *Proc. 22nd Int'l Symp. Fault-Tolerant Computing*, pp.170-177, 1992

[10] Y. Saad and M.H.Schultz, "Topological Properties of Hypercubes," *IEEE Trans. Computers*, Vol 37, No 7, page 867-872, July 1988

[11] J. Wu, "Unicasting in Faulty Hypercubes Using Safety Levels," *IEEE Trans. Computers*, vol. 46, no.2, pp. 241-247, Feb 1997

[12] J. Wu. and E.B. Fernandez, "Broadcasting

in Faulty Hypercubes," *Proc. 11th Symp. Reliable Distributed Systems*, pp. 122-129, Oct 1992

[13] Jie Wu, "Adaptive Fault-Tolerant Routing in Cube-Based Multicomputers Using Safety Vectors," *Parallel and Distributed Systems, IEEE Transactions on*, Vol 9, Issue 4, page 321-334, April 1998

최 병 환 (Byung-whan Choi) 정회원

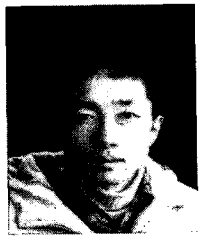
1987년 2월 공주사범대학교 수학교육과 학사

1998년 8월 충북대학교 전자계산학과 석사

1999년 3월~현재 충북대학교 전자계산학과 박사과정

<관심분야> 알고리즘, 결함허용, 통신이론

강 성 수 (Sung-soo Kang) 정회원



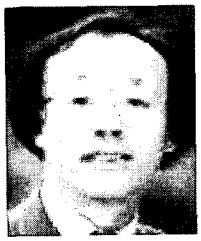
1992년 2월 충북대학교 전기전자공학부 학사

2005년 2월 충북대학교 전자계산학과 석사

현재 충북대학교 전자계산학과 박사과정

<관심분야> 분산컴퓨팅, 라우팅

이 충 세 (Chung-sei Rhee) 정회원



1979년 8월 Univ. of South Carolina 컴퓨터과학과 석사

1990년 12월 Univ. of South Carolina 컴퓨터 과학과 박사

1998년 9월~1991년 9월 동아대학교 경영정보학과 부교수

1991년 9월~현재 충북대학교 컴퓨터과학과 교수

<관심분야> 결함허용, 알고리즘, 전문가시스템, 정보보안