분산 깊이 우선 탐색 프로토콜의 복잡도 개선을 위한 연구

최 종 원†

요 약

그래프 트래버설(traversal) 기법은 그래프의 노드들을 '방문(visiting)'하는 임의의 패턴이라고 할 수 있으며, 그래프 트래버설 방법 중 하나가 깊이 우선 탐색법이다. 이 깊이 우선 탐색 기법은 유향 그래프의 강결합 요소나 일반 그래프의 이중 결합 요소를 찾는데 이용된다. 이러한 깊이 우선 탐색 기법을 분산 네트워크 상에서 구현하기 위한 분산 프로토콜은 통신망의 위상 변화가 없는 고정위상 문제와 시간의 지남에 따라 위상의 변화가 있는 동적 위상 문제로 나누어 볼 수 있다. 본 논문에서는 먼저 고정 위상에서의 개선된 분산 깊이 우선 탐색 프로토콜을 설계하고 다음으로 이 프로토콜을 동적 위상에 적용하여 링크/노드의 고장/복구에 대처할 수 있는 레질리언트 프로토콜을 설계하였다. 또한, 이들 프로토콜의 메시지와 시간 복잡도를 각각 분석하고, 기존 프로토콜의 복잡도 보다 개선된 결과를 보였다.

Improvement on The Complexity of Distributed Depth First Search Protocol

Jongwon Choe†

ABSTRACT

A graph traversal technique is a certain pattern of 'visiting' nodes of a graph. Many special traversal techniques have been applied to solve graph related problems. For example, the depth first search technique has been used for finding strongly connected components of a directed graph or biconnected components of a general graph. The distributed protocol to implement this depth first search technique on the distributed network can be divided into a fixed topology problem where there is no topological change and a dynamic topology problem which has some topological changes. Therefore, in this paper, we present a more efficient distributed depth first search protocol with fixed topology and a resilient distributed depth first search protocol where there are topological changes for the distributed network. Also, we analysed the message and time complexity of the presented protocols and showed the improved results than the complexities of the other distributed depth first search protocols.

1. 서 론

지난 십여 년간 우리는 연산과 통신 기술의 급속한 발전에 따른 데이타 통신 분야에서의 커다란 발전을 보아왔다. 이들 기술적인 향상과 네트워크 서비스에 대한 증대하는 수요는 보다 정교하고 효율적인 분산 네트워크 시스템의 설계를 필요로 하게 되었으며, 분산 시스템은 중앙 집중 시스템 보다 고장허용(fault-

※이 논문은 1994년도 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음.

† 정 회 원:숙명여자 대학교 전산학과 조교수

논문접수:1996년 1월 22일, 심사완료:1996년 4월 16일

tolerant), 자원 공유, 그리고 병렬성 등에 있어서 장점을 갖고 있다. 그리고, 분산 네트워크 시스템을 운용하기 위한 분산 네트워크 프로토콜을 설계함에 있어서 각 노드들이 주고받는 메시지 수와 통신 지연을 한 단위로 할 때의 수행 시간을 줄임으로써 분산 네트워크 통신 성능을 향상시키려는 연구가 활발히 진행되고 있다[1-8, 10, 12, 14].

그래프 트래버설(traversal) 기법은 그래프(네트워크)의 노드들을 '방문(visiting)'하는 임의의 패턴이라고 할 수 있으며, 그래프 트래버설 방법 중 하나가 깊이 우선 탐색법이다. 이 깊이 우선 탐색법은 유향 그래프의 강결합 요소나 일반 그래프의 이중 결합 요소를 찾는데 이용이 된다. 강결합 요소는 통신 경로에서 유향 사이클을 찾는데 이용되는데 이 유향 사이클은 교착상태(deadlock)를 야기할 수 있다. 이중 결합 요소는 임의의 고장 노드가 통신망을 단절(disconnect)하는지 아닌지를 조사하는데 이용될 수 있다[3, 14, 17].

여러 가지의 깊이 우선 탐색 프로토콜이 있는데 이들을 'centralized'와 'distributed'의 두 가지 유형으로 나누어 볼 수 있다. 본 논문에서는 분산(distributed) 프로토콜에 초점을 맞추고자 한다. 그리고 지금까지의 많은 논문들이 고정위상의 문제를 다루고 있는데, 즉 이들은 프로토콜이 수행되고 있는 동안, 네트워크의 위상 변화가 없다고 가정하고 있다[3, 7, 14, 17].

대부분의 경우, 분산 프로토콜은 위상이 변화하는 통신 네트워크에서 수행이 된다. 위상 변화는 시간이 지남에 따라 노드 그리고/혹은 링크의 추가나 삭제를 나타낸다. 따라서, 네트워크에서의 위상 변화에도 불구하고 올바로 재기능을 할 수 있는 프로토콜이 필요하다. 이러한 프로토콜을 레질리언트(resilient, failsafe, 그리고 reliable) 프로토콜이라고 한다[2, 8, 11].

본 논문에서는 먼저 고정 위상에서의 개선된 분산 깊이 우선 탐색 프로토콜을 설계하고 다음으로 이 프로토콜을 동적 위상에 적용하여 링크/노드의 고장/복구에 대처할 수 있는 레질리언트 프로토콜을 설계한다. 본 논문에서 제안한 고정 위상에서의 개선된 분산 깊이 우선 탐색 프로토콜은 통신망의 각 노드가 통신하기 위해 교환하는 메시지에 '미방문 노드 집합(UNS)'와 '방문 노드 집합(VNS)'를 사용한다. 이 프로토콜은 최악의 경우, 최대 사용된 메시지 수와 단위시간 수가 $2|N|-3$으로 한정지어진다. 최선의 경우에는, 단지 $|N|-1$의 메시지와 단위 시간만이 요구된다.

또한 동적 위상에서의 분산 깊이 우선 탐색 프로토콜은 최악의 경우, $k(2|N|+|R_i|-5)$ 메시지와 단위시간을 필요로 하며, 여기서 $|N|$은 노드의 수이고 $|R_i|$는 고장을 탐지한 노드와 루트 노드 사이의 노드 수이다. 또한, k는 링크의 고장회수이고, 링크의 회복에는 무관하다. 즉, 이 프로토콜의 복잡도는 링크의 복구수에 의존하지 않는다. 한편, 최선의 경우는 $k(2|N|+|R_i|-3-|N_v|)$의 메시지와 단위시간을 필요로 한다.

2절에서는 본 논문의 네트워크 모델을 기술하고 3절에서는 고정위상에서의 개선된 분산 깊이 우선 탐색 프로토콜을 설명한다. 그리고, 4절에서는 동적 위상에서의 분산 깊이 우선 탐색 프로토콜을 설명하고, 마지막으로 5절에서 결론을 맺는다.

2. 네트워크 모델

이 절에서는 본 연구의 기반이 되는 통신 네트워크의 2 가지 모델을 간략히 설명한다.

2.1 고정 위상 모델

분산 프로토콜에 대한 환경은 점대점 비동기 통신 네트워크이며, 이는 무방향, 연결 통신 그래프 $G(N, E)$로 나타낼 수 있는데, 여기서 N은 유한 집합의 노드 수이고 $E \subset N \times N$은 유한 집합의 링크 수이다.

유일한 식별 번호를 갖는 각 노드는 통신을 담당하는 컴퓨팅 단위로서 프로세서, 로칼 메모리 그리고 무한 용량의 입출력 큐를 갖는다. 각 노드는 초기에 국지적 환경에 대한 정보만을 갖으며(즉, 자신의 식별 번호, 이웃 노드들의 식별 번호, 그리고 이웃 링크의 상태 등), 무한의 프로세싱 능력과 메모리 기능을 갖는다.

각 링크 $l=(u, v) \in G$는 노드 u와 v를 연결하는 양방향 통신 회선이다. 링크의 양끝에 그 링크와 연관된 입력 큐와 출력 큐가 있다. 링크를 통해 수신(송신)된 메시지는 선입선출 방식으로 입력(출력) 큐에 놓여지며, 그 순서대로 처리된다.

노드들이 공유하는 공통 메모리(common memory)가 없다. 메시지는 작동하는 링크를 통해 송수신될 수 있다. 링크가 작동할 때, 메시지는 독립적으로 양

방향 전송될 수 있고, 유한 시간 후에 오류없이 그리고 보내진 순서대로 링크의 종단 노드에 도착한다.

분산 프로토콜(분산 알고리즘이라고도 함)은 $|N|$개의 프로그램으로 구성되며, 각각은 G에 속하는 임의의 노드에 할당된다. G에 있는 각 노드는 그 프로그램에 따라 작동한다. 프로토콜은 그들 프로그램을 수행하는 임의의 노드 집합에 의해 시작된다. 한 메시지의 수신은 로칼 계산을 수행하게 하고 그 결과로 나타나는 새로운 메시지의 전송을 야기한다. 따라서, 분산 프로토콜에서의 기본 오퍼레이션들은 다음과 같다: 링크를 따라 이웃 노드에게 메세지 전송, 이웃 노드로부터 메세지 수신, 그리고 로칼 메모리에 있는 정보 처리. 이러한 오퍼레이션들이 아주 빠른 시간에 처리(전송 시간과 비교하여) 된다고 가정한다. 전송 중에 있는 메시지가 없을 때 이 프로토콜의 전체적인 종료에 도달했다고 말한다.

2.2 동적 위상 모델

분산환경에서, 위상변화의 수, 순서, 혹은 시간에 대한 가정을 미리 정할 수 없다. 따라서, 위상 변화가 있는 통신망에서, 통신 링크의 특성에 대해 자세한 명시를 해야만 한다. 본 연구에서는 [13, 18]에서 제안된 비신뢰적인 통신 네트워크를 위한 모델을 따르기로 한다.

링크의 양끝에 있는 노드는 링크가 "up(작동중)" 혹은 "down(고장)" 인지를 탐지할 수 있는 어떤 방법을 갖고 있다고 가정한다. 따라서, 링크가 고장나고 복구될 때마다, 양끝 노드들은 유한 시간(그러나 반드시 동시일 필요는 없음) 안에 이 사실을 알게 된다. 메시지는 작동중인 링크를 통해서 송수신될 수 있다. 그러나, 메시지 전송 중에 링크의 고장이 발생할 수 있으므로, 메세지는 반드시 수신자에 도달하지는 않는다. 고정위상 모델에서와 마찬가지로, 링크가 작동할 때, 메세지는 양방향으로 전송될 수 있고, 이들은 반대편에 유한 시간 후에 오류없이 그리고 순서대로 도착할 것이다. 링크가 복구되었을 때, 그 링크로 전송되고 있는 메세지는 없고, 또한 그 링크로 전송되기를 기다리는 메세지도 없다(즉, 링크로 전송되고자 큐에 대기하고 있던 모든 메세지는 링크가 고장나면 모두 큐에서 지워진다).

노드의 고장/복구는 그 노드에 이웃하는 모든 링크의 고장/복구로 간주될 수 있다. 이상 제시한 네트워크 모델 특성은 [3, 13, 19]의 논문에서 공통으로 사용되고 있다.

3. 고정 위상에서의 개선된 분산 깊이 우선 탐색 프로토콜

이 절에서는 고정 위상을 갖는 일반적인 무방향 통신 네트워크에서 깊이 우선 탐색 트리를 구성하기 위한 개선된 분산 깊이 우선 탐색 프로토콜을 보인다. 본 논문에서 제안하는 분산 깊이 우선 탐색 프로토콜은 통신망의 각 노드들이 통신을 위해 주고받는 메시지에 'UNS(Unvisited Node Set)'과 'VNS(Visited Node Set)'을 포함시킨다. 최악의 경우, 이 프로토콜은 많아야 $2|N|-3$ 메시지와 단위 시간을 필요로 한다. 최선의 경우, 단지 $|N|-1$ 메시지와 단위 시간을 필요로 한다. 그리고, 이 프로토콜은 시작 노드(루트 노드라 부름) 대신에 통신망의 임의의 노드에서 종료될 수 있다.

표 1은 여러 분산 깊이 우선 탐색 프로토콜과 통신망 그래프 $G(N, E)$에 대한 고정 위상에서의 메시지와 시간 복잡도를 나타내고 있다.

〈표 1〉 여러 분산 깊이 우선 탐색 프로토콜의 복잡도

저 자	시간 복잡도	메시지 복잡도
T. Cheung [4]	$2\|E\|$	$2\|E\|$
B. Awerbuch [3]	$<4\|N\|$	$4\|E\|$
.B. Lakshmanan [10]	$2\|N\|-2$	$<4\|E\|-(\|N\|-1)$
I. Cidon [7]	$\leq 2\|N\|$	$\leq 3\|E\|$
M. B. Sharma [14]	$2\|N\|-2$	$2\|N\|-2$

3.1 개선된 분산 깊이 우선 탐색 프로토콜 개요

이 프로토콜에서, 복잡도를 줄이기 위해 [14]에서와 달리 'UNS'를 통신 메시지에 추가로 포함시킨다. 따라서, 이 프로토콜은 수행 중에 노드들이 송수신하는 메시지에 'UNS'와 'VNS'을 포함하게 된다. 이 UNS를 조사함으로써, 이 프로토콜은 임의의 노드에서 종료될 수 있다.

이 프로토콜의 수행은 외부로부터 초기 메시지를

받는 루트 노드에서 시작을 하게 된다. 이때 초기 값으로 UNS와 VNS는 공집합을 갖는다. 루트 노드는 자신의 이웃 노드들 중에서 '미방문(unvisited)' 노드 하나를 선택하고, 그 노드를 'son'으로 표시한 후에, 자신의 식별번호를 VNS에 추가하고 또한, 이웃 노드 중 나머지 미방문 노드들은 UNS에 포함시킨다.

그리고, UNS와 VNS를 이용하여 지금까지 '방문(visited)'과 '미방문(unvisited)'된 노드들에 대한 정보를 갖는 FIND(VNS, UNS) 메시지를 선택된 노드로 보낸다[그림 1 참조]. 이 메시지를 수신하는 노드는 송신 노드를 'father' 노드로 표시하고 자신의 이웃 노드중에 '미방문' 노드가 있는가를 조사한다. 즉, 자신의 이웃 노드들의 식별 번호가 VNS에 포함되어 있는가를 검사한다. 한 이웃 노드가 VNS에 포함되어 있지 않다면 그 노드는 미방문 노드이다.

모든 이웃 노드들이 '방문' 되었고 수신된 메시지에 있는 UNS가 공집합이면, 이 프로토콜은 종료하게 된다. 즉, 더이상 '방문'해야 될 노드가 없다.

또한, 모든 이웃 노드들이 '방문' 되었고 수신된 메시지에 있는 UNS가 공집합이 아니면, 'father' 노드에게 FIND(VNS, UNS) 메시지를 보낸다. 즉, 백트랙킹을 하게 된다.

그리고 만약에 '미방문' 노드가 남아 있다면, 이들 '미방문' 노드들중에서 하나를 선택한다. 자신의 식별번호를 VNS에 추가하고 나머지 '미방문' 노드들의 식별번호를 UNS에 추가한다. 그리고 나서, 선택된 노드로 VNS와 UNS를 갖는 FIND 메시지를 보낸다. 마지막으로, 메시지를 전송하기 전에 선택된 노드의 식별번호가 UNS에 있다면 UNS에서 삭제시킨다.

이러한 방식으로, UNS가 공집합이고 이웃 노드들 중에 '미방문' 노드가 남아있지 않을 때, 이 프로토콜의 수행은 임의의 노드에서 종료될 수 있다. 이와는

```
. 프로토콜에서 사용된 메시지와 변수:
FIND(VNS, UNS) : 임의의 노드가 처음으로 '방문'되거나 백트랙킹될 때
                 그 노드에 수신되는 메시지
VNS : 지금 까지 '방문'된 노드들의 집합
UNS : 지금 까지 '미방문'된 노드들의 집합
Neighbors(i) : 노드 i의 이웃 노드 집합
Father(i) : 깊이 우선 탐색 트리에서 노드 i의 'father' 노드
Son(i) : 깊이 우선 탐색 트리에서 노드 i의 'son' 노드 집합

. 프로토콜의 초기화
임의의 노드가 루트 노드로 선택되어, 그 노드로부터 탐색이 시작된다.
이 프로토콜을 수행하기 위해, 이 루트 노드는 외부로부터 메시지를
수신한다고 가정한다.

. 노드 i 의 프로토콜
Response to receiving FIND(VNS, UNS) message from node j

Father(i) <- j
case :
 1. There is no unvisited node in Neighbors(i) and UNS is empty : stop
 2. There is no unvisited node in Neighbors(i) and UNS is not empty :
      send FIND(VNS, UNS) to Father(i) /* 백트랙킹 */
 3. There is an unvisited node :
    a. select an unvisited node k ∈ Neighbors(i)
    b. Son(i) <- Son(i) + { k }
    c. VNS <- UNS + { i }
    d. if k ∈ UNS then UNS <- UNS - k
    e. UNS <- UNS + (unvisited nodes ∈ Neighbors(i) except k)
    f. send FIND(VNS, UNS) to node k
```

(그림 1) 개선된 분산 깊이 우선 탐색 프로토콜 의사 코드

(Fig. 1) An improved distributed depth first search protocol pseudo code

달리 지금까지 제안된 다른 프로토콜들의 종료는 항상 시작 노드인 루트 노드에서 발생한다[3, 4, 7, 14].

위에서 설명한 통신망의 각 노드에서 수행되는 분산 깊이 우선 탐색 프로토콜을 의사 언어 형식으로 나타내면 그림 1 과 같다.

3.2 프로토콜의 복잡도 분석

이 절에서는 3.1에서 제안한 개선된 분산 깊이 우선 탐색 프로토콜의 시간과 메시지 복잡도를 살펴본다.

정리 3.1 최악의 경우, 제안된 분산 깊이 우선 탐색 프로토콜은 각각 최대 $2|N|-3$ 메시지와 단위시간을 필요로 한다.

증명: 통신망의 각 노드는 루트노드를 제외하고 부모 노드로부터 하나의 메시지만을 받는다. 그리고 각 노드가 수신한 메시지에 따라, 부모 노드에게 메시지를 보내거나 아무런 메시지도 보내지 않는다. 즉, UNS가 공집합이고 '미방문' 이웃 노드를 갖고 있지 않으면, 이 순간에 통신망에 '미방문' 노드가 없기 때문에 어떠한 메시지도 보내지 않는다.

최악의 경우, 루트노드와 마지막으로 '방문' 된 노드를 제외하고, 통신망의 각 노드는 부모 노드로부터 (혹은 부모 노드로) 메시지를 수신(혹은 송신) 하는데, 이에 $2|N|-4$ 메시지 와 단위시간이 요구된다. 즉, $|N|-2$개의 노드는 부모 노드로부터 하나의 메시지를 받고 백트랙킹을 위해 부모 노드로 하나의 메시지를 보낸다. 따라서, $2(|N|-2)$개의 메시지가 필요하다. 그리고 마지막으로 '방문'된 노드는 부모 노드(최악의 경우, 부모 노드는 루트 노드이다)로부터 하나의 메시지를 수신하며, 한 단위 시간이 요구된다. 그리고 이 노드는 백트랙킹을 위한 메시지가 필요 없다. 따라서, 이 프로토콜의 수행에 필요한 전체 메시지 수와 단위시간 수는 $2|N|-3$ 이다. □

정리 3.2 최선의 경우, 제안된 분산 깊이 우선 탐색 프로토콜은 각각 $|N|-1$ 메시지와 단위시간을 필요로 한다.

증명: 이 경우, 통신망의 각 노드는 루트 노드를 제외하고 부모 노드로부터 하나의 메시지를 받는다. 따라서 $|N|-1$ 메시지와 단위 시간이 요구된다. 마지막으로 '방문'된 노드가 부모 노드로부터 메시지를 받을 때, '미방문' 노드가 없고 UNS가 공집합이므로 이 노드는 프로토콜 수행을 종료한다. 즉, 이 경우에는 전혀 백트랙킹을 위한 메시지가 필요 없다. 따라서, 이 프로토콜의 수행에 필요한 전체 메시지 수와 단위시간 수는 $|N|-1$ 이다. □

따라서, 항상 $2|N|-2$ 메시지와 단위 시간을 필요로 하는 [14] 프로토콜과 비교할 때, 본 논문에서 제안한 프로토콜이 성능 면에서 우수하다고 말할 수 있다.

4. 동적 위상에서의 분산 깊이 우선 탐색 프로토콜

[12]에서, 최대 $k(2|N|+|R_i|-3)$ 메시지와 단위시간을 필요로 하는 레질리언트 분산 깊이 우선 탐색 프로토콜을 제안하였다. 여기서 $|N|$은 노드의 총수를 나타내며, $|R_i|$는 고장 발생노드와 루트 노드사이에 존재하는 노드의 수이다. 통신망에서 k개의 링크 고장과 m개의 링크회복이 있다는 가정을 한다.

이 프로토콜의 복잡도가 링크회복과 관계없음을 알 수 있다. 동적 위상에서의 분산 깊이 우선 탐색 프로토콜의 복잡도를 개선하기 위하여 앞 절에서 제안된 프로토콜을 이용하기로 한다. 따라서 이 절에서는 통신망에 위상 변화가 있을 때 그 통신망에 대한 보다 개선된 레질리언트 분산 깊이 우선 탐색 프로토콜를 보이고자 한다. 초기 깊이 우선탐색 트리가 주어지면, 레질리언트 프로토콜은 위상 변화에 대응되는 깊이 우선 탐색 트리를 갱신하게 된다. 유한의 위상 변화가 임의의 기간 동안에 발생할 수 있으며 이 시간이후에는 위상 변화가 없다고 가정하면, 이 프로토콜은 마지막 위상 변화후 유한 시간 내에 그 통신망에 대한 깊이 우선 탐색 트리를 구하게 된다.

4.1 그래프 이론적 특성

분산 환경에서, 링크가 고장/복구될 때, 위상 변화를 탐지하는 양단 노드들은 고장/복구 프로세스를 시작한다. 이 프로세스는 일반적으로 깊이 우선 탐색 트리를 갱신하기 위해 여러 노드들의 협력을 필요로

한다. 개선된 프로토콜을 제안하기에 앞서 깊이 우선 탐색 트리 구성과 관련된 그래프 이론적 특성을 살펴보기로 한다. 이 특성들은 고장/복구 문제에 관련된 노드들의 동작을 위한 기초가 될 것이다.

편의를 위하여 다음과 같은 표기법을 사용하기로 한다. 초기 깊이 우선 탐색 트리 T가 주어지면, $T-\{l\}$과 $T+\{l\}$ 각각이 T에 링크 $l=(u, v)$의 삭제와 추가를 나타낸다고 하자.

고장에 대한 반응으로, 단일 링크 $l=(u, v)$의 고장을 고려해보자. 노드 u가 노드 v의 부모 노드라고 가정하자. 또한 T_b는 링크 고장 전에 그래프 $G(N, E)$의 깊이 우선 탐색 트리이고 T_a는 링크 고장 후 그래프 $G(N, E-\{l\})$의 깊이 우선 탐색 트리라고 정의하자. 여기서 $l \in E$은 고장난 링크이다. l이 비-트리 링크 ($l \notin T_b$)이면, 깊이 우선 탐색 트리 T_b에 아무런 영향을 미치지 않는다($T_a=T_b$). 따라서 깊이 우선 탐색 트리를 갱신할 필요가 없다. 그러나, 만약에 l이 트리-링크 ($l \in T_b$)이면, T_b는 두개의 서브-트리 T_u와 T_v(T_u가 루트 노드를 포함한다)로 나뉘어진다. 각 종단 노드가 링크 고장을 탐지한 후에, 노드 u와 v는 이 두개의 서브-트리를 연결하기 위해 레질리언트 분산 깊이 우선 탐색 프로토콜을 시작한다. 그러나 실제에 있어서, T_v 트리는 새로운 깊이 우선 탐색 트리를 구성하기 위해 T_u로부터 메시지를 수신하여야 하기 때문에 노드 v는 아무 작업도 하지 않는다. 이 프로토콜을 수행한 후에, $T_a=T_u+\{l'\}+T_{v'}$를 얻게 되며, 여기서 l'는 T_u와 $T_{v'}$의 이웃 링크들 중하나이다. 그리고 $T_{v'}$는 T_v의 새로 구성된 깊이 우선 탐색 트리이다.

명제 4.1 깊이 우선 탐색 트리 T_b가 주어지고, 트리-링크 $l=(u, v)$가 고장이 난 후에, T_b는 T_u와 T_v 두개의 서브-트리로 나뉘어진다. T_a를 구한 후에 T_u는 위상이 변하지 않는다.

이제 여러 개의 고장이 발생하는 경우(복구는 없다고 가정)을 생각해보자. 초기 깊이 우선 탐색 트리는 여러 개의 서브-트리로 나뉘어진다. 명제 4.1에 의해 재구성 프로세스를 마친 후, 위상이 변하지 않는 루트 노드를 포함하는 하나의 서브-트리가 생성된다. 그래프 $G(N, E-l_f)$의 마지막 깊이 우선 탐색 트리는 $T_{f_u}+\{l'_1+\cdots+l'_f\}+T_{v_1'}+\cdots+T_{v_f'}$의 형태이다($l_f$는 고장 링크의 집합임). 여기서 l'_i는 T_{f_u}와 $T_{v_i'}$의 동작 가능한 이웃 링크들 중하나이다. 그리고 $T_{v_i'}$는 T_{v_i}의 새로 구성된 깊이 우선 탐색 트리이고 T_{f_u}는 마지막 링크 고장 후, 루트 노드를 포함하는 서브-트리이다.

복구에 대한 반응으로 단일 복구 경우를 생각해 보자. 복구된 링크의 종단 노드들 중 하나가 초기 그래프에 포함되어있지 않았다면, $T_a=T_b+\{l\}$이 될 것이다($l=(u, v)$은 복구된 링크이다). 종단 노드들이 모두 초기 그래프에 없었다면, 이 복구에 의해서는 새로운 트리가 생성되지 않는다. 즉, $T_a=T_b$, l로서 그래프가 단절(disconnected)된다. 두 종단 노드 모두가 초기 그래프에 있었다면, 복구된 링크에 의한 깊이 우선 탐색 트리 재구성 프로세스는 만들어지지 않는다. 이 프로세스는 복구된 링크의 상태를 동작가능한 링크(즉: 링크 상태는 'up'이다)로 바꾼다. 제안된 분산 깊이 우선 탐색 프로토콜은 망 그래프에서 구성될 수 있는 여러 깊이 우선 탐색 트리들 중 하나를 찾기 때문이다.

따라서, 복구된 링크에 대한 링크 고장 프로세스에 의해(즉: 이 링크가 복구되기 전의 고장 프로세스가 있어야한다), 다른 깊이 우선 탐색 트리가 구성될 것이다. 이것이 최소 신장 트리 구성의 경우[8]와 다른 점이다. 다중 복구의 경우, 위에 설명된 개념을 적용할 수 있다.

4.2 개선된 레질리언트 분산 깊이 우선 탐색 프로토콜

이 절에서는 링크 고장과 회복에 관한 프로시저를 설명한다.

4.2.1 링크 고장 프로시저

한 노드가 자신의 이웃 링크중 비-트리 링크가 고장인 것을 감지하면, 그 노드는 다시 그 링크의 상태만을 바꾼다(즉, 링크상태가 'down'으로 변경되며, 이는 그 링크가 작동을 하지 않는다는 것을 의미한다).

임의의 트리-링크 $l=(u, v)$의 고장을 고려해 보자. 여기서 노드 u는 노드 v의 부모 노드이다. 이 경우, 노드 u는 깊이 우선 탐색 트리를 재구성하기 위해 레질리언트 분산 깊이 우선 탐색 프로토콜을 수행하며 Failure 프로세스를 만든다. 이 Failure 프로세스에서 T_u에 있는 노드들은 방문(visited) 노드로 취급되기 때문에 T_u에 있는 노드의 식별 번호를 알아야 하며, 이를 위해 Collect 메시지를 이용한다.

따라서, 노드 u가 자식(child) 노드를 갖고 있으면, 이들에게 Collect(failed_link, VNS, UNS) 메시지를 보낸다. failed_link는 서로 다른 Failure 프로세스를 구별하기 위해 이용된다. 그러나, 만약에 자식 노드들을 갖고 있지 않다면, 자신의 부모 노드에게 Collect 메시지를 보낸다. 여기서, 노드 u가 고장을 감지하였을 때, VNS와 UNS는 초기 값으로 공집합을 갖는다. 그리고 노드 u가 자식 노드들중 하나로 Collect 메시지를 전송하기 전에 노드 u는 자신의 식별 번호를 VNS에 추가하여야 한다. 부모 노드로부터 이 Collect 메시지를 수신하는 각 노드는 VNS에 자신의 식별번호를 추가하고 이웃노드들 중 비-트리 링크 노드들의 식별번호가 UNS에 포함되어 있지 않으면 UNS에 추가한다. 그리고 자식 노드들에게 그 메시지를 순방향 전송한다. 또한, 자신의 식별번호가 UNS에 포함되어 있다면 그 번호를 UNS에서 제거한다.

Collect 메시지를 수신한 노드가 자식 노드를 갖고 있지 않다면, 부모 노드에게 Collect_Ack(failed_link, VNS, UNS) 메시지를 보낸다. 부모 노드에게 Collect_Ack 메시지를 보낼 때, 임의의 노드는 이 고장 프로세스을 위해 자식 노드들에게 Collect 메시지를 전송한 경우에, 그 노드는 자신의 모든 자식 노드들로부터 Collect_Ack 메시지를 수신하였는가를 조사하여야 한다. 그리고 자식노드로부터 Collect 메시지를 처음으로 수신한 각 노드는 자신의 식별번호를 VNS에 추가하고 자식 노드들이 있다면 이 메시지를 전송한다. 그렇지 않으면, 부모 메시지에게 Collect 메시지를 전송한다.

이러한 방식으로, 루트 노드는 최종적으로 T_u에 속하는 모든 노드들의 식별 번호를 수집하게 된다. 그리고, 루트 노드는 루트 노드로 Collect 메시지가 전달된 역경로를 따라 노드 u에게 Find(failed_link, VNS, UNS) 메시지를 전송한다. 이를 위해, 자식노드로부터 Collect 메시지를 수신하는 각 노드는 Collect 메시지가 전달된 노드를 기억하여야 한다(이 목적을 위해 back_link 라는 변수를 사용한다). 한 노드가 Find 메시지를 수신하면, 메시지의 failed_link 변수를 조사한다. 노드 식별 번호가 failed_link의 부모 노드 식별 번호 u와 같으면, 노드 u는 방문되지 않은 이웃 링크중 작동하는 링크하나를 선택한다. 그리고 선택된 링크의 종단 노드를 'son'으로 표시하고 그 선택된 노드로 Reconstruction(failed_link, VNS, UNS) 메시지를 보낸다. 노드 u의 이웃 노드들중 나머지 미방문 노드들을 UNS에 넣는다. Reconstruction 메시지를 수신하는 노드는 이 메시지를 송신한 노드를 'father' 노드로 표시한다. 노드 u가 미방문 노드를 갖고 있지 않으면, 'father' 노드에게 Reconstruction 메시지를 보낸다. 그 후에 'father' 노드는 같은 재구성 프로시저를 수행한다. 임의의 노드가 Reconstruction 메시지를 수신했을 때, 모든 이웃 노드들이 방문되었고 수신 메시지 안에 있는 미방문 노드 집합(UNS)가 공집합이면, 수행을 중단한다(즉: 링크 고장에 대한 새로운 깊이 우선 탐색 트리를 발견하게 된다).

그렇지 않고, 모든 이웃 노드들이 방문되었고 UNS가 공집합이 아니면, Reconstruction 메시지를 부모 노드에게 보낸다. 그리고 모든 이웃 노드들이 방문되지 않았으면, 미방문 노드들 중에서 하나를 선택한다. 그리고 자신의 식별번호를 VNS에 추가하고 나머지 미방문 이웃 노드들의 식별 번호를 UNS에 추가한다. 그런 다음, 선택된 노드에게 Reconstruction메시지를 보낸다. 메시지를 보내기 전에, 선택된 노드의 식별번호가 UNS에 있으면 이 번호를 삭제한다. 이러한 방식으로, 이 프로토콜의 수행은 미방문 노드 집합이 공집합이고 임의의 노드에서 남아있는 이웃 노드들 중 미방문 노드들이 없을 때 임의의 노드에서 프로토콜의 수행이 종료될 수 있다. 이것이 다른 레질리언트 알고리즘들과 다른 점이다[6, 8, 19].

다음으로 다중 링크 고장의 경우를 고려하여 보자. 임의의 한 노드가 Collect(혹은 Collect_Ack) 메시지를 전송 혹은 수신한 후에 다른 링크 고장으로 인한 또다른 Collect(혹은 Collect_Ack) 메시지를 수신하게 되었을 때, 만약에 부모(혹은 자식) 노드의 이웃 트리 링크가 고장이라면, Collect(혹은 Collect_Ack) 메시지를 무시한다. 왜냐하면 이미 그 노드의 부모 노드(혹은 자신)가 이 링크 고장에 대한 Failure 프로세스를 시작하였기 때문이다. 그렇지 않으면, 앞서 기술한 바와 같이 부모(혹은 자식)에게 그 메시지를 전송한다. 이때 위상 변화에 대한 Failure 프로세스가 적어도 하나이상 존재하게 된다.

이와 같은 방식으로, 루트 노드가 마지막 Failure 프로세스에 대한 T_u에 포함된 VNS와 UNS를 성공적으로 수집하면, 루트 노드는 링크 고장을 탐지한 노

드에게 깊이 우선 탐색 트리를 재구성하기 위한 Find (failed_link, VNS, UNS) 메시지를 전송한다. 이 전파 과정동안, 재구성을 시작해야하는 노드로 가는 역 경로에 있는 링크가 고장이 나면, 이 Find 메시지는 고장이나 링크 종단에 있는 노드에 의해 무시된다. Find 메시지가 링크 고장을 탐지한 노드에게 올바로 전달된다면, 이웃 노드들 중 미방문 노드를 선택하고 위에서 기술한 대로 Reconstruction 메시지를 전송한다.

4.2.2 링크 복구 프로시저

링크 $l=(u, v)$의 복구를 고려하자. 두 종단 노드들이 링크 복구를 탐지하였을 때, 두 노드가 초기 그래프에 있었는지를 조사한다. 그렇다면, 각 노드는 링크 상태를 작동중인 링크(즉 링크상태='up') 로 변환한다.

한 노드만이 초기 그래프에 있었다면, 초기 그래프에 있던 노드는 다른 노드를 'son'으로 세트하고 초기 그래프에 없던 다른 노드는 이 노드를 'father'로 세트한다. 만약 두 노드 u와 v가 초기 통신망 그래프에 없었다면 이들 두 노드는 비연결 요소를 구성한다. 즉, 복구전의 분산 깊이 우선 트리와 단절된다.

위에서 설명한 프로토콜의 의사 언어 형식은 아래 그림 2와 같다.

```
Response of node u to a failure notification for link l=(u, v) where u is the parent of v.
    if l is a tree-link then
        if u has children then
            delete node v from the children nodes
            send Collect((u,v), (v)) to all children
            set k to number of children nodes
        endif
    else
        state of link l ← 'down'
    endif

    Response to receiving Collect(failed_link, (node id set)) on link l
    if Collect message is from parent node then
        node_id set := node_id set ∪ (own_id)
        if there are children then
            send Collect(failed_link, (node_id set) to all children
            set k to number of children to which it sends messages
        else
            send Collect_Ack(failed_link, (node_id set)) to parent
        endif

    else (* Collect message is from a child*)
        back_link ← sending node_id
        node_id set := node_id set ∪ (own_id)
        if there are children(except the child sent the Collect message) then
        send Collect(failed_link, (node_id set) to the children
        set k to number of children to which it sends messages
        else
            if it has parent then
            send Collect(failed_link, (node_id set)) to parent
            else Find(failed_link, (node_id set)) on back_link
            endif
        endif
    endif

    Response to receiving Collect_Ack(failed_link, (node_id set))
    collect_id set := collect_id set ∪ (node_id set)
    k := k - 1
```

```
if k=0 and it has parent and (own_id = 'u' or Collect message was from child) then
    send Collect_Ack(failed_link, {collect_id set}) to parent
endif
if k=0 and it has parent and own_id ≠ 'u' then
    send Collect_Ack(failed_link, {collect_id set}) to parent
endif
if k=0 and it has no parent then
    send Find(failed_link, {collect_id set})
endif

Response to receiving Find(failed_link, {node_id set})
if own_id = 'u' then
    for some node i of its neighbors,
    if it is ∉ node_id set and there is no parent
    and children relationship then
    mark(i) ← 'son'
    node_id set := node_id set ∪ {own_id}
    send Reconstruction({node_id set}) to node i
    endif
else
    send Find(failed_link, {node_id set}) on back_link
endif

Response to receiving Reconstruction({node_id set}) from node k
if own_id ∉ {node_id set} then
    reset current parent and children relationship
    mark(k) ← 'father'
endif
for some node i of its neighbors,
if i ∉ node_id set and there is no parent
and children relationship then
    mark(i) ← 'son'
    node_id set := node_id set ∪ {own_id}
    send Reconstruction({node_id set}) to node i
    endif

elseif own_id = 'root' then stop
    else send Reconstruction ({node_id set}) to parent
    endif
endif

Response of node u, v to a recovery notification for link l = (u, v)
if u and v was in initial network then
    state of link l ← 'up'
endif
if u was in initial network then
    mark(v) ← 'son'
else
    mark(v) ← 'father'
endif
if v was in initial network then
    mark(u) ← 'son'
else
    mark(u) ← 'father'
endif
```

(그림 2) 레질리언트 분산 깊이 우선 탐색 프로토콜 의사 코드

(Fig. 2) A resilient distributed depth first search protocol pseudo code

4.3 레질리언트 분산 깊이 우선 탐색 프로토콜의 복잡도 분석

이 절에서는 4.1에서 제안한 레질리언트 분산 깊이 우선 탐색 프로토콜의 시간과 메시지 복잡도를 보인다.

정리 4.1: 최악의 경우, 제안된 레질리언트 프로토콜에서 이용되는 메시지 수는 k개의 링크 고장의 경우 $k(2|N|+|R_i|-5)$로 한정된다. 여기서 $|N|$은 노드의 개수이고 $|R_i|$는 링크 고장을 탐지한 노드로부터 루트 노드까지의 노드 수이다.

증명: 링크 $l=(u, v)$가 고장이면, 깊이 우선 탐색 트리는 두개의 서브-트리로 나뉘어진다. 이들을 T_u와 T_v라 하자. $|N_u|$와 $|N_v|$를 각각 T_u와 T_v에 있는 노드의 수라고 정의한다. 따라서 $|N|=|N_u|+|N_v|$이다.

T_u에서 Collect, Collect_Ack와 Find를 위한 $2(|N_u|-1)$ 메시지가 필요하다. T_u를 T_v에 연결하기 위해, $|R_i|$ Reconstruction 메시지가 필요한데, 이는 T_u에 있는 노드에게 보내기 위한 하나의 메시지와 T_u에 있는 T_v로의 가지 노드를 찾기 위한 $|R_i|-1$ 메시지이다. 또한 T_v에서 서브 트리 재구성을 위해 $2(|N_v|-3)$ 메시지가 필요하다.

따라서, 하나의 링크 고장 후에 깊이 우선 탐색 트리를 재구성하기 위해 $2(|N_u|+|N_v|)-5+|R_i|$ 메시지가 필요하다. 여기서 $|N_u|+|N_v|$는 $|N|$으로 대치할 수 있다. 그 결과 최악의 경우 최대 $2|N|+|R_i|-5$ 메시지를 이용한 재구성 깊이 우선 탐색 트리를 얻을 수 있다.

k개의 링크 고장이 발생하면, 깊이 우선 탐색 트리를 재구성하는데 필요한 메시지의 수가 $k(2|N|+|R_i|-5)$로 제한된다. 최악의 경우는 이전 링크 고장에 대한 프로세스의 수행이 끝나기 바로 전에 링크 고장이 다시 발생하는 경우이다. □

정리 4.2: 최선의 경우, 제안된 레질리언트 프로토콜이 이용하는 메시지 수는 k개의 링크고장의 경우 $k(2|N|+|R_i|-3-|N_v|)$로 제한된다. 여기서 $|N|$은 노드의 수이고 $|R_i|$는 링크고장 탐지 노드로부터 루트 노드까지의 노드 수이다.

증명: 링크 $l=(u, v)$ 고장을 고려하자. 최선의 경우 T_u에서 필요한 메시지 수는 최악의 경우와 같다. 즉, Collect, Collect_Ack 그리고 Find 를 위해 $2(|N_u|-1)$ 메시지가 필요하다.

T_u를 T_v에 연결하기 위해, $|R_i|$개의 Reconstruction 메시지가 필요하다(즉, T_u로의 연결노드를 찾기 위해 백트랙킹을 위한 $|R_i|-1$ 개의 메시지가 필요하고 T_v에 있는 한 노드로 연결하기 위해 하나의 메시지가 필요하다).

또한, T_v에서 $|N_v|-1$개의 Reconstruction 메시지가 필요하다(정리 3.2). 따라서, 하나의 링크 고장 후, 깊이 우선 탐색 트리를 재구성하기 위해 $2(|N_u|-1)+|R_i|+|N_v|-1$ 개의 메시지가 필요하다. $|N|=|N_u|+|N_v|$를 이용하면, 최선의 경우에 많아야 $2|N|+|R_i|-3-|N_v|$ 개의 메시지를 사용하여 깊이 우선 탐색 트리를 재구성할 수 있다.

만약 k개의 링크 고장이 발생한다면, 깊이 우선 탐색 트리를 재구성하는데 필요한 메시지는 많아야 $k(2|N|+|R_i|-3-|N_v|)$가될 것이다. □

정리 4.3: 제안된 레질리언트 프로토콜의 시간 복잡도는 최악의 경우에 $k(2|N|+|R_i|-5)$의 단위 시간이 필요하며, 최선의 경우에는 최대 $k(2|N|+|R_i|-3-|N_v|)$의 단위 시간이 필요하다. 링크의 고장은 k개이고, 모든 메시지는 단위 시간 내에 링크의 종단 노드에게 전달이 된다고 가정한다.

증명: 프토토콜의 전체 수행 시간은 링크를 통해 메시지를 전송하는데 걸리는 시간이다. 최악의 경우, 메시지가 단위 시간에 종단 노드에게 전달된다는 가정과 정리 5.1로부터, 메시지를 전달하는데 필요한 전체 시간은 최대 $k(2|N|+|R_i|-3-|N_v|)$의 단위 시간으로 제한된다.

최선의 경우는, 최악의 경우와 마찬가지로 가정과 정리 5.2로부터, 메시지를 전달하는데 필요한 전체 시간은 최대 $k(2|N|+|R_i|-3-|N_v|)$의 단위 시간으로 제한된다. □

5. 결 론

분산 네트워크 프로토콜은 통신망에서 중요한 역할을 하고 있다. 그리고 분산 시스템은 고장-허용(fault tolerance), 자원 공유, 그리고 병행성 등을 포함하여

중앙집중 시스템보다 많은 장점을 제공하고 있다.

논문에서 살펴본 바와 같이, 분산 프로토콜은 두개의 기본 요소를 갖고 있다. 즉, 메시지를 수신, 처리, 변형, 그리고 송신하는 프로세스와 이 메시지가 전송되고 구조적이고 동적인 특성을 갖는 네트워크를 형성하는 링크이다.

분산 네트워크 시스템을 운용하기 위한 분산 네트워크 프로토콜을 설계함에 있어서 각 노드들이 주고받는 메시지 수와 링크상의 통신 지연을 한 단위로 할 때의 수행시간을 줄이고자 하는 연구로써 본 논문에서는 고정 위상에서의 분산 깊이 우선 탐색 프로토콜과 이를 기반으로 동적 위상에서의 레질리언트 분산 깊이 우선 탐색 프로토콜을 제안하였다.

본 논문에서 제안된 결과는 유향 그래프의 강결합 요소를 찾거나 일반 그래프의 이중 결합 요소를 찾는 분산 프로토콜 설계에 이용될 수 있으며 앞으로의 연구과제로 남아 있다.

또한, 노드의 고장이나 회복은 그 노드이 모든 이웃 링크의 고장이나 회복으로 가정하였다. 그러나, 만약 노드가 올바로 기능을 하지 않으면(malfunction), 이는 Byzantine General Problem[20]과 같은 문제를 야기할 것이다 이 문제는 아직 해결되지 않은 문제로써, 향후 연구되어야 할 중요한 과제이다.

참 고 문 헌

[1] Y. Afek, B. Awerbuch, and E. Gafni, "Applying Static Network Protocols to Dynamic Networks", IEEE, 1987, pp358-370

[2] B. Awerbuch and S. Even, "Reliable Broadcast Protocols in Unreliable Networks", Networks, Vol. 16, 1986, pp381-396

[3] B. Awerbuch, "A New Distributed Depth-First-Search Algorithm", Information Processing Letter, April, 1985, pp147-150

[4] C. Cheng, "Graph Traversal Techniques and The Maximum Flow Problem in Distributed Computation", IEEE Trans. Sofware Eng., 1983, pp504-512

[5] J.W. Choe and S. Kumar, "A More Efficient Distributed Depth First Search Algorithm in Fixed Topology", Northcon/91, Oct. 1991, pp333-338

[6] J.W. Choe and S. Kumar, "A Resilient Depth First Search Algorithm in Changing Topology", Proceedings of SIMM Inter. Conf., Dec. 1991, pp56-59

[7] I. Cidon, "Yet Another Distributed Depth-First-Search Algorithm", Inform. Process. Letters, 26, Jan, 1988, pp301-305

[8] I.A. Cimet and S.P.R. Kumar, "A Resilient Distributed Algorithm for Minimum Weight Spanning Trees", Int'l Conf. on Parallel Proc., St. Charles, Il, Aug. 1987, pp1076-1084

[9] S. Even, 'Graph Algorithms', Computer Science Press, 1979.

[10] K.B. Lakshmanan, N. Mecnakshi and K. Thulasiraman, "A Time Optimal Message-Efficient Distributed Algorithm for Depth-First-Search", Inform. Process. Lett. 25, 1987, pp103-109

[11] P. Merlin and A. Segall, "A Failsafe Distributed Routing Protocol", IEEE Trans. on Comm., Vol. Com-27, no. 9, Sept., 1979, pp1280-1287

[12] J.H. Reif, "Depth-First-Search Is Inherently Sequential", Inform. Process. Lett. 20, 1985, pp229-234

[13] A. Segall, "Distributed Network Protocols", IEEE Transactions on Information Theory, Vol. IT-29, No. 1, Jan. 1983, pp23-35

[14] M. B. Sharma and S.S. Iyengar, "An Efficient Distributed Depth-First-Search Algorithm", Inform. Processing. Lett. 32, Sept., 1989, pp183-186

[15] 조이남, 박경호, "네트워크상의 강 결합 요소를 구하는 최적의 분산 알고리즘", 정보과학회논문지, Vol. 19, No. 4, July 1992, pp409-419

[16] 박영태, 구용원, "분산 시스템을 위한 교착 상태 탐지 알고리즘", 정보과학회논문지, Vol. 19, No. 6, Nov. 1992 pp649-660

[17] E. H. Chang, "Echo Algorithms: Depth Parallel Operations on General Graphs", IEEE Trans. on Soft. Eng, Vol. SE-8, No. 4, Jul, 1982.

[18] B. Awerbuch, "Complexity of Network Synch-

ronization", J. ACM, Vol. 32, No. 4, Oct. 1985, pp804-823

[19] C. Cheng, I. A. Cimet and S. Kumars, "A Protocol to Maintain a Minimum Spanning Tree in a Dynamic Topology", ACM SIGCOM Symp. Comm. Arch. and Protocols, Aug., 1988, pp330-338

[20] L. Lamport, R. Shostak, and H. Pease, "The Byzantine Generals Problem", ACM Trans. Program. Lang. Syst. 4(3), Jul. 1982, pp382-401

최 종 원

1984년 서울대학교 전자계산기 공학과 졸업(학사)
1986년 서울대학교 전자계산기 공학과 졸업(공학석사)
1992년 Northwestern University 졸업(공학박사)
1993년~현재 숙명여자대학교 전산학과 조교수
관심분야:분산 라우팅 알고리즘, Multimedia in Education, 디지탈 도서관 등