이동 에이전트를 갖는 네트워크 환경에서 단절점을 찾기 위한 방법

김영균° 김경하 김영학 오길호
금오공과대학교 컴퓨터공학과
{ygkim, khkim, yhkim, gilho}@cespc1.kumoh.ac.kr

A Method for Finding Articulation Points on a Mobile Agent Environment

Young-Gyun Kim° Kyoung-Ha Kim Young-Hak Kim Gil-Ho Oh
Dept. of Computer Engineering, Kumoh National University of Technology

요 약

분산 네트워크 환경에서 단절점의 존재는 신뢰성을 떨어뜨리는 주요한 요인이며, 통신 트래픽의 부하가 집중될 수 있는 지점이다. 본 논문에서는 분산 이동 컴퓨팅 환경을 갖는 네트워크상에서 분산 깊이 우선 탐색 방법을 이용하여 단절점을 찾는 효율적인 방법을 제안 한다. 시작 노드는 단절점을 발견하기 위한 이동 에이전트를 인접한 노드에 보내고, 파견된 이동 에이전트는 네트워크상의 각 노드들을 차례로 방문하여 깊이 우선 번호를 기본으로 하여 단절점을 찾게 된다.

1. 서론

분산 환경에서 단절점(articulation point)의 존재는 신뢰성을 떨어뜨리는 주요한 요인이며, 통신 트래픽이 집중 될 수 있는 지점이다. 신뢰성 높은 분산 환경이 요구되는 경우, 이러한 단절점을 탐색하여, 적절한 조치를 취하는 것이 바람직하다.

깊이 우선 탐색은 분산 컴퓨팅 네트워크 상에서 폭넓게 사용 되고 있다[1]. 예로서, 이러한 탐색 방법은 어떤 사이트의 실패 또는 네트워크의 절단(disconnect)을 확인하기 위해 사용될 수 있는 이중 결합 요소(biconnected components)를 찾거나 또는 유향 네트워크의 데드락(deadlocks)을 발생할 수 있는 유향 사이클(directed cycles)을 탐지하기 위한 강결합 요소(strongly connected components)를 찾기 위해 사용 되고 있다[2,3].

본 논문에서는 [1]에서 제안한 분산 DFS 알고리즘을 변형하여 이동 에이전트 환경에서 네트워크상에 존재하는 단절점을 찾기 위한 방법을 제안한다. 분산 환경에서 이동 에이전트는 전자상거래, 네트워크 관리, 병렬 계산 등의 다양한 응용 분야에 사용되고 있다. [1]에서는 동적 백트래킹을 사용하여 RETURN 메시지의 수를 줄이고 있으나, 제안한 방법에서는 동적 백트래킹을 사용하지 않는다.

분산 깊이 우선 탐색 알고리즘에 관한 많은 선행 연구들이 이루어져 왔다[1,3,4,5,6]. 선행 연구들은 주로 분산 프로세스(process) 환경에서 깊이 우선 탐색을 연구 하였다. 따라서, 네트워크 상에서 존재하는 탐색 노드들에 해당 프로세스를 수작업으로 수행시키야 하는 문제점이 있다. 본 논문에서는 이러한 문제점을 보완하기 위해 이동 에이전트 환경에서 깊이 우선 탐색 방법으로 각 노드들을 방문하여 단절점을 찾는다.

대표적인 이동 에이전트 시스템으로서 IBM의 Aglet [8]이 있다. IBM의 Aglet은 Java로 구현되어 이질적인 환경에서 수행이 가능하며, 쉽게 기능을 확장할 수 있는 이동 에이전트 시스템이다. 본 논문에서는 단절점을 찾기 위한 이동 에이전트 시스템으로 IBM의 Aglet을 사용 한다.

2. 단절점을 찾기 위한 순차 알고리즘

그래프 G의 정점들 중에서 그 정점을 그 정점에 부속한 모든 간선들과 같이 삭제하면 최소한 두 개의 연결 요소를 갖는 그래프 G' 를 생성하는 정점 v를 단절점이라 한다[7]. 주어진 그래프 G의 단절점은 다음과 같이 구한다. 그림 1과 같이 연결그래프가 있다고 하자. 원

내부에 있는 번호는 노드를 식별하기 위한 번호 이다. 그림 1의 연결 그래프에서의 단절점은 노드번호 1,3,5,7이다. 이 노드들 중에서 실패(failure)가 발생한 경우 네트워크는 적어도 2개 이상의 서로 다른 네트워크로 분할 된다. 그림 1은 깊이 우선 탐색을 노드 3을 루트 노드로 했을 때 각 노드를 방문한 것이다. 노드 옆에 명시된 번호는 노드를 방문한 순서 번호, 즉 dfn번호 이다. dfn번호와 식1과 같이 정의된 low값을 이용해서 단절점을 구한다.

low(u) = min{ dfn(u), min{ low(w) | w는 u의 자식 노드 }, min{ dfn(v) | (u,v)는 백 간선 } } (식1)

정점 u가 2개 이상의 자식을 갖는 신장 트리의 루트이거나, 루트가 아니면서 low(w) ≥ dfn(u) 를 만족하는 자식 w를 갖게 되면 단절점이 된다[7].

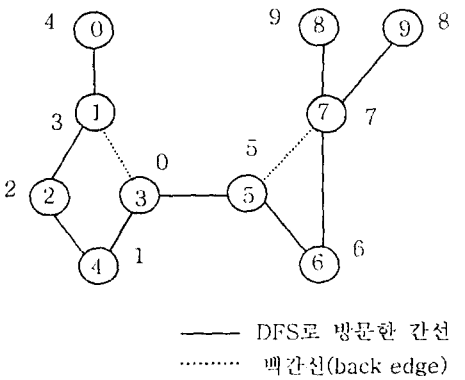

그림 1. 연결 그래프의 깊이 우선 탐색.

3. 이동 에이전트 컴퓨팅 방법

3.1 가정 및 제안한 방식의 개요

본 논문에서 제안하고 있는 알고리즘의 수행 모델은 무향의 통신 그래프 G=(V,E)로 상호 집속된 네트워크를 구성하였다. V는 통신 네트워크를 구성하는 노드(node)들의 집합이고, E는 노드들을 연결하기 위한 양방향 통신 링크들의 집합이다. 각 노드는 Aglet 이동 에이전트를 수행하기 위한 자바 가상 환경(java virtual environment)이 설치되어 있고, Aglet 이동 에이전트 시스템을 수행하고 있다고 가정한다. Aglet은 에이전트의 복사본을 만드는 기능(clone), 주어진 목적지 노드로 에이전트를 전송하는 기능(dispatch), 목적지에 도착 했을 때 자동으로 수행되는 기능(onArrival), 자신의 에이전트를 문맥으로부터 제거하는 기능(dispose) 등의 기본적인 기능을 갖추고 있다.

탐색을 시작하는 초기에, 루트 노드는 인접한 이웃 노드들에게 본 논문에서 제안한 분산 깊이 우선 탐색을 이용한 단절점 탐색 알고리즘1을 수행하기 위한 이동 에이전트를 파견 한다. 이동 에이전트는 하위 노드로 디스패치된 이후에, 즉시 깊이 우선 탐색 방문 규칙에 따라 디스패치된 노드의 이웃 노드들에게 FORWARD 메시지를 포함한 사본의 이동 에이전트를 생성하여, 디스패치 한다.

이동 에이전트를 하위의 노드로 파견한 노드는 인접한 이웃 노드의 에이전트로부터 RETURN 메시지가 수신되기를 기다린다. RETURN 메시지를 수신한 노드는 자신의 이웃 노드 중에서 아직 이동 에이전트를 파견하지 않은(방문하지 않은) 노드가 존재하는 경우, 이동 에이전트를 파견 한다. 그렇지 않은 경우, RETURN 메시지를 수신한 노드는 현재 노드의 단절점 여부를 판단하기 위한 충분한 정보를 RETURN 메시지로부터 추출하게 된다. 추출한 정보를 근거로 현재 노드가 단절점인지를 판단하고, 결과를 포함한 새로운 RETURN 메시지를 구성하고, 상위의 노드로 새롭게 구성한 RETURN 메시지를 백트래킹 한다.

최종적으로 마지막 RETURN 메시지를 수신한 루트 노드가 방문한 전체 노드로 구성된 네트워크의 단절점 집합을 출력한다. 식1에서 방문한 각 노드의 dfn값은 3.3절에서 설명하는 그림 2의 FORWARD 메시지로부터 구할 수 있고, low값은 그림 3의 RETURN 메시지로부터 구할 수 있다.

3.2 이동 에이전트를 이용한 DFS 알고리즘

알고리즘 1을 수행하는 이동 에이전트가 네트워크 상의 노드를 차례로 방문 한다. 먼저 노드에 도착한 에이전트는 onArrival 이벤트가 발생되어 도착하는 즉시 에이전트 자신에게 지역 FORWARD 메시지를 전송 한다. 현재 노드가 루트 노드인 경우, 알고리즘 1을 수행하기 전에 { 방문한 노드 } = ϕ로 초기화 된다.
메시지 타입은 FORWARD와 RETURN 타입으로 구분 되며, 각 노드는 이웃노드들에 대한 지역 정보를 갖고 있다.

[알고리즘 1: Finding_AP]

```
begin /* 노드 도착시(FORWARD) 및 리턴 메시지 수신시에 수행됨.*/
  if onArrival then  /* 노드에 도착시 수행 됨. */
     자신의 노드에 있는 에이전트에게 FORWARD 메시지 전송.
  endif
  if 메시지 타입 = FORWARD then
       현재 노드의 이웃 노드들의 집합을 구성.
       { 방문한 노드 } := { 방문한 노드 } ∪ { 현재 노드 }
  endif
  { 방문하지 않은 노드 } := { 이웃 노드 }
                - ( { 방문한 노드 } ∩ { 이웃 노드 } )
  if  { 방문하지 않은 노드 } ≠ 0 then
    begin
       다음에 방문할 노드 := Next( { 방문하지 않은 노드 } )
       { 방문하지 않은 노드 } := { 방문하지 않은 노드 }
                              - { 다음에 방문할 노드 }
       FORWARD 메시지를 참조하여 현재 노드의 dfn번호를 구함.
       현재 노드의 dfn을 추가한 새로운 FORWARD 메시지를 구성.
       FORWARD 메시지를 포함한 Finding_AP 이동 에이전트를
                                  다음에 방문할 노드로 디스패치.
    end
  else  /* RETURN 메시지 수신시에 수행 됨. */
```

```
if 현재 노드 = 루트 노드 then
  begin
    RETURN 메시지로부터 방문한 노드들의 dfn과 low값을 구함.
    for ( 모든 방문한 노드 u에 대해 )
      begin
        for ( u의 모든 자식 노드 w에 대해 )
          if low(w) ≥ dfn(u) then
              단절점의 집합 = 단절점의 집합 ∪ { u }
      endfor
    단절점의 집합을 출력.
    Stop.
  end
else  /* 루트 노드가 아닌 경우 수행 됨. */
  begin
    RETURN 메시지의 dfn과 low값 테이블로부터
          현재 노드의 low값을 구한 후 RETURN 메시지에 추가.
    RETURN 메시지를 부모노드로 전송.
    에이전트 자신을 현재 노드의 이동 에이전트 환경으로부터
                                        제거(dispose).
  end
 endif
endif
end.
```

3.3 메시지의 구조 및 분석

네트워크상의 각 노드에 파견된 이동 에이전트는 그림 2와 같은 구조의 FORWARD 메시지와 함께 자신의 사본 에이전트를 아직 방문하지 않은 이웃 노드로 디스패치 할 수 있고, 그림 3과 같은 구조의 RETURN 메시지를 수신할 수 있다. FORWARD 메시지와 함께 사본의 에이전트는 방문한 현재 노드에서 수행을 시작 한다. 방문한 현재 노드의 id를 추가하고, 방문한 노드의 dfn을 구한 후 이를 수신한 메시지의 뒷부분에 추가한 FORWARD 메시지를 새롭게 구성 한다. 새롭게 구성된 FORWARD 메시지를 자신의 사본 에이전트와 함께 방문하지 않은 인접한 노드로 전송한다.

메시지 타입	방문한 노드의 id	방문한 노드의 dfn
FORWARD		

그림 2. FORWARD 메시지의 구조

메시지 타입	방문한 노드의 id	방문한 노드의 dfn
RETURN		

방문한 노드의 low값	

그림 3. RETURN 메시지의 구조

네트워크의 마지막 노드는 그림3과 동일한 구조의 RETURN 메시지를 부모 노드로 전송 한다. 이미 수신한 FORWARD 메시지로부터 방문한 노드들의 dfn을 추출하고, 식1에 따라 현재 노드의 low값을 구한 후, 이를 추가한 RETURN 메시지를 부모 노드로 전송 한다.

최종적으로 RETURN 메시지를 수신한 루트 노드는 수신한 RETURN 메시지로부터 방문한 노드들의 dfn과 low값을 참조하여 단절점을 판단한 후 출력을 내게 된다.

제안한 방식은 분산 깊이 우선 탐색의 방문 순서로 단절점을 발견하기 위한 이동 에이전트를 디스패치 한다. 제안한 방식의 메시지 복잡도(message complexity)는 방문한 노드들의 크기 |V|에 비례 한다. 메시지 복잡도는 이동 에이전트를 디스패치할 때 |V|-1번의 이동 에이전트를 전송하고, 루트 노드를 제외한 노드들은 최소한 1번의 RETURN 메시지를 전송하므로, RETURN 메시지를 |V|-1번 만큼 전송 한다. 따라서, 메시지 복잡도는 2|V|-2 이다. 각 메시지를 저장하기 위해 요구되는 비트 수는 $O(|V|\log_2|V|)$ 크기가 요구 된다. 본 논문에서는 순차적인 알고리즘을 기본 알고리즘으로 사용하였고, 단절점을 찾기 위한 이동 에이전트를 이웃한 노드들에 동시에 파견하여, 단절점 탐색 시간을 줄일 수 있도록 확장할 수 있다.

4. 결론

본 논문에서 제안한 방법은 차후 단절점 뿐만 아니라, 네트워크 상의 다양한 정보를 탐색하기 위해서 새로운 코드를 갖는 이동 에이전트를 다시 파견하도록 함으로 쉽게 확장이 가능하다. 차후 과제로 병렬적으로 탐색하여 탐색 시간을 줄일 수 있는 방법과 통신 복잡도를 낮출 수 있는 방법에 대하여 연구할 예정이다.

참고 문헌

[1] S.A.M Makki, George Havas, Distributed algorithms for depth-first search, *Information Processing Letters* 60(1996) 7-12.

[2] E.J.H. Chang, Echo algorithm: Depth parallel operations on general graphs, *IEEE Trans. Software Engineering* 8 (1982) 391-401.

[3] B. Awerbuch, A new distributed depth-frist search algorithm, *Information Processing Letters* 20 (1985) 147-150.

[4] T. Cheung, Graph traversal techniques and the maximum flow problem in distributed computation, *IEEE Trans, Software Engineering* 9 (1983) 504-512.

[5] D. Kumar, S.S. Iyengar and M.B. Sharma, Corrections to a distributed depth-first search algorithm, *Inform. Process. Lett.* 35 (1990) 55-56.

[6] I. Cidon, Yet another distributed depth-first-search algorithm, *Inform. Process. Lett.* 26 (1987/88) 301-305.

[7] Horowitz, Sahni, Anderson-Freed, Fundamentals of Data Structures in C (1993), W.H. Freeman and company

[8] Danny B. Lange / MITSURU OSHIMA, Programming And Deploying Java Mobile Agents With Aglets, Addison Wesley, 1998.