분산 망에서 자원발견을 위한 결정 알고리즘
(A Deterministic Resource Discovery Algorithm in Distributed Networks)

박 혜 경 † 유 관 우 ††
(Hae Kyeong Park) (Kwan Woo Ryu)

요 약 본 논문에서는 네트웍으로 연결된 일련의 장치들이 서로를 발견하는 문제인 자원 발견(Resource Discovery)문제를 해결하는 알고리즘을 제안한다. 최근 Harchol등은, 장치의 수를 n이라 할 때, $O(n\log^2 n)$ 연결 통신복잡도와 $O(n^2\log^2 n)$ 포인터 통신복잡도를 가지고 $O(\log^2 n)$ 시간복잡도에 이 문제를 해결하는 알고리즘을 제안하였는데, 이는 임의(randomized) 알고리즘이며 종료시점(convergence)을 식별할 방법이 없다는 단점을 가진다. 본 논문에서 우리는 이러한 단점을 없앤 더욱 효율적인 결정(deterministic) 알고리즘을 제안한다. 제안 알고리즘은, 총 링크 수를 m이라 할 때, $O(m\log n)$ 연결 통신복잡도와 $O(n^2\log n)$ 포인터 통신복잡도를 가지고 $O(\log n)$ 시간복잡도에 자원발견 문제를 해결한다.

Abstract In this paper, we propose a deterministic algorithm to solve the resource discovery problem, that is, some subset of machines to learn the existence of each other in a large distributed network. Harchol et al. proposed a randomized algorithm solving this problem within $O(\log^2 n)$ rounds with high probability, which requires $O(n\log^2 n)$ connection communication complexity and $O(n^2\log^2 n)$ pointer communication complexity, where n is the number of machines in the network. His solution is based on randomization method and it is difficult to determine convergence time. We propose an efficient algorithm which improve performance and the non-deterministic characteristics. Our algorithm requires $O(\log n)$ rounds which shows $O(m\log n)$ connection communication complexity and $O(n^2\log n)$ pointer communication complexity, where m is the number of links in the network.

1. 서 론

큰 규모의 분산 망에서는 여러 개의 컴퓨터 혹은 장치들이 하나의 작업을 위해 협력해야 하는 경우가 종종 있다[1,2,3,4,5,6,7,8]. 예를 들어, 분산 웹 캐싱 프로토콜[9], 분산 파일 시스템[10], 도메인 네임 서버[11], 혹은 분산 계산[12] 등을 구현하는 경우이다. 이러한 응용들을 수행하기에 앞서, 각 장치들은 같이 작업할 다른 장치들을 인식할 수 있어야 하는데, 각각의 장치가 공동작업할 다른 모든 장치들을 발견하는 문제를 자원 발견(Resource Discovery)문제라고 정의한다[13].

이 문제를 해결하기 위하여 Harchol 등은 주어진 분산 망을 약하게 연결된 방향성 그래프(weekly connected directed graph)라 가정하고 이를 해결하는 임의 알고리즘(randomized algorithm)을 개발하였다[13]. 이 알고리즘은 큰 규모의 분산 캐쉬(large-scale distributed cache)를 개발하는 프로젝트의 일부분으로서 MIT Computer Science 연구실에서 구현되었는데, 현재 웹 캐쉬 회사인 Akamai Technologies가 그 특허권을 가지고 있다. 이 프로젝트를 시작한 동기는 인터넷 컨텐츠 분산 시스템에서 접근 빈도가 높은 컨텐츠 공급자(major content supplier) 웹 사이트의 웹 페이지에 대한 사용자들의 접근 속도를 높이자는 것이었다. 이때, 각 장치들이 공동작업을 하기 위해서는 서로의 위치를 알아내는 작업 즉 자원 발견이 선행되어야 했다.

자원발견 문제를 해결하는 알고리즘에서 가장 중요하게 요구되는 조건은 시간과 통신면에서의 효율성이다. 다시 말해 각 장치들이 얼마나 빨리 그리고 적은 통신 비용으로 서로를 발견할 수 있는가 하는 것이다. 특히 망의 상태가 자주 변경되어 반복적으로 이 알고리즘을

† 비 회 원 : 한국전자통신연구원 MPLS S/W팀 연구원
phk@etri.re.kr
†† 종신회원 : 경북대학교 컴퓨터공학과 교수
kwryu@bh.knu.ac.kr
논문접수 : 2000년 12월 28일
심사완료 : 2001년 8월 2일

수행하게 되는 경우, 시간과 통신의 효율성은 더욱 중요한 의미를 가지게 된다.

먼저 시간의 척도로 병렬 라운드를 정의하면, 자원발견 알고리즘이 각 컴퓨터에서 동기화 된 병렬 라운드에 맞춰 작업을 한다고 가정할 때, 하나의 병렬 라운드는 각 장치가 간단한 계산을 하고 자신의 이웃들에게 임의 크기의 메시지를 보낼 수 있는 시간이다. 그러면 자원발견 알고리즘의 수행 시간은 각 장치가 다른 모든 장치들을 발견할 때까지 걸리는 병렬 라운드의 수가 된다. 실제로 각 라운드의 수행시간은 모두 일정하지 않은데, 이는 이웃노드와의 거리가 다양하고, 각 장치 즉 라우터나 컴퓨터의 처리속도 또한 다르기 때문이다. 그러나 시간 복잡도를 간단하게 표현하기 위해 병렬 라운드를 사용하고 있으며, 실제 시스템에서는 처리시간이 가장 긴 병렬 라운드를 기준으로 수행하도록 구현하면 된다.

그리고 또 다른 성능 척도인 통신량은, 알고리즘이 수행되는 동안의 통신한 메시지 수를 나타내는 연결 통신 복잡도와 통신할 때 전달되는 데이타 양을 나타내는 포인터 통신 복잡도로 측정이 된다.

전체 노드 수가 n일 때, Harchol등이 제안한 Name-dropper알고리즘은 자원발견 문제를 $O(\log^2 n)$ 병렬 라운드에 해결할 확률이 높은 임의 알고리즘으로, $O(n\log^2 n)$의 연결 통신 복잡도와 $O(n^2\log^2 n)$의 포인터 통신 복잡도를 가진다[13]. 그러나 임의 알고리즘의 특성상, 특정한 수행시간 내에 반드시 해결된다는 것을 보장하기가 힘들기 때문에 실시간 응용에 사용하기 어려운 단점을 가지며, 또한 이 알고리즘은 어느 시점에 수행을 끝내야 하는지를 알 수 없다는 단점을 가진다.

본 논문에서는 이러한 단점들을 해결하면서 더 빠른 시간과 더 적은 통신량으로 자원발견 문제를 해결하는 결정 알고리즘을 제안한다. 제안 알고리즘은 $O(m\log n)$의 연결 통신 복잡도와 $O(n^2\log n)$의 포인터 통신 복잡도를 가지고 $O(\log n)$ 병렬 라운드 내에 자원발견 문제를 해결한다.

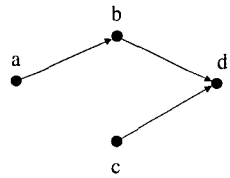
2. 모 델

먼저 네트웍 상의 각 장치는 유일한 식별자를 가진다고 가정하자. 실제로 각 장치의 식별자를 IP주소로 생각할 수 있는데, 각 장치는 다른 장치와 협력하기 위해 그 장치의 IP주소 즉 식별자를 알아야 한다.

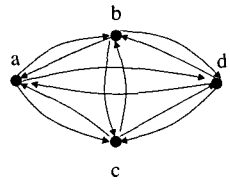
이러한 분산 시스템은 방향성 그래프 G=(V,E)로 쉽게 표현 될 수 있다. 노드들의 집합 V는 n개의 장치들의 집합을 나타내며, 각 노드 v는 각기 다른 식별자 ID(v)를 가지고 있다. 그리고 방향성 에지 $\langle u,v\rangle \in E \in (u, v \in V)$는 노드 u가 노드 v의 ID를 알고 있고 따라서 v에게 메시지를 보낼 수 있다는 것을 나타낸다.

초기 방향성 그래프가 주어진 상태에서, 자원발견 알고리즘이 수행되는 동안 각 노드들은 다른 노드를 발견하여 에지를 추가하게 되고, 최종적으로 완전 그래프(complete graph)를 형성하게 되면 각 노드가 다른 모든 노드를 발견할 수 있으므로 수행을 멈추게 된다.

예를 들어 그림 1(a)과 같은 연결된 방향성 그래프 G=(V,E), V={a,b,c,d} E={<a,b>, <b,d>, <c,d>}가 입력으로 주어질 때, 이는 장치 a는 장치 b를, b는 d를, c는 d를 알고 있고 메시지를 보낼 수 있음을 나타낸다. 그래프 G를 입력으로 하여 각 장치들이 자원발견 알고리즘을 수행하면, 그림 1(b)와 같은 완전 그래프 $G_{complete}=(V, E_{complete})$, $E_{complete}=\{\langle a,b\rangle, \langle a,c\rangle, \langle a,d\rangle, \langle b,a\rangle, \langle b,c\rangle, \langle b,d\rangle, \langle c,a\rangle, \langle c,b\rangle, \langle c,d\rangle, \langle d,a\rangle, \langle d,b\rangle, \langle d,c\rangle\}$가 형성되고, 각 노드는 자신에서 다른 모든 노드로 향하는 에지를 가지게 되어, 다른 모든 노드를 발견하게 된다.



(a) 입력 방향성 그래프 G=(V,E)



(b) 출력 완전 그래프 $G_{complete}=(V,E_{complete})$

그림 1 자원발견 문제의 입력 그래프와 결과 그래프

3. 기존 자원발견 알고리즘들

자원 발견 문제를 해결하기 위한 기존 알고리즘들은 다음과 같다. 먼저 현재 인터넷 라우터에 사용되는 Flooding 알고리즘을 이 문제를 해결하기 위해 사용할 수 있다[14]. 초기에 각 장치들은 이웃 장치들에 대한 정보를 가지며, 수행되는 동안 초기의 이웃 장치 들과만 통신하도록 제한하는 방법이다. 각 라운드마다 각 장

치는 이전 라운드까지 발견한 모든 장치들에 대한 정보를 이웃 장치에게 전달한다. 따라서 $d_{initial}$이 초기 그래프의 지름이라고 할 때, 수행을 마칠 때까지 $\Theta(d_{initial})$ 라운드가 걸리며, 노드의 수를 n이라 할 때, 그래프의 지름은 최악의 경우 $\Theta(n)$이 되므로 수행시간은 $\Theta(n)$이 된다. 그리고 $m_{initial}$을 초기 그래프에서 에지의 수라 할 때, 연결 통신 복잡도는 $\Theta(d_{initial} m_{initial})$이 된다. 따라서 Flooding 알고리즘은 시간과 통신량에 있어서 효율적이지 못하다.

다음, Swamping 알고리즘은 각 라운드마다 자신이 발견한 모든 이웃 장치와 통신을 한다는 점만 제외하고 Flooding 알고리즘과 같다. 이 알고리즘은 log n 라운드 이상 걸리지 않는 매우 빠른 알고리즘이다. 그러나 너무 과도한 통신량을 가진다는 단점이 있다. 최악의 경우 연결 통신 복잡도는 $\Omega(n^2)$이며, 포인터 통신 복잡도는 $\Omega(n^3)$이 된다.

마지막으로, Name-dropper알고리즘은 자원발견 문제를 해결하기 위해 Harchol 등에 의해 개발되었다[13]. 이 알고리즘의 기본 접근방법은 각 라운드 동안 각 노드는 하나의 이웃노드를 임의로 선택하고, 선택된 노드에게 자신의 이웃 리스트들을 전달하는 것이다. 예를 들어 그림 2와 같이 A노드는 이웃노드 B,C,D,E를 알고 있고, B노드는 이웃노드 F와 G를 알고있다고 할 때, A노드는 자신의 이웃노드 중 임의의 노드 B를 선택하여 B에게 A의 이웃 리스트 {B,C,D,E}를 전달한다. 그러면 B는 F,G노드와 A,B,C,D,E노드를 모두 알게 된다. A노드와 마찬가지로 다른 모든 노드에서도 같은 병렬 라운드에 이와 같은 과정을 수행한다. 이 알고리즘은 $O(\log^2 n)$라운드에 자원발견 문제를 해결할 확률이 높으며, 각 장치는 각 라운드 동안 하나의 연결을 맺기 때문에 연결 통신 복잡도는 $O(n\log^2 n)$이며, 포인터 통신 복잡도는 $O(n^2\log^2 n)$이다. 이 알고리즘은 수행되는 연산이 간단하다는 장점을 가지며, 하나의 노드에 동시에 많은 연결이 요구되는 경우, 실제로 이를 모두 수용하지 못하고 이중 일부에 대해서만 연결을 허용할 때에도, 같은 병렬 라운드 내에 자원발견 문제를 해결할 수 있을 확률이 높다는 장점을 가진다. 그러나 임의 알고리즘의 특성상, 특정한 수행시간 내에 반드시 해결된다는 것은 보장하기 힘들기 때문에 실시간 응용에 사용하기 어려운 단점을 가지며, 수행이 완료된 시점을 인식하는 방법이 없다는 단점을 가진다. 그리고 네트웍이 정적이라는 것을 가정하고 있으므로, 알고리즘이 수행되는 동안 장치가 추가 혹은 삭제 되는 경우에 대해서는 고려하지 못하고 있다.

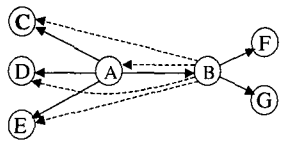


그림 2 Name dropper 알고리즘 방식

4. 제안 알고리즘

본 절에서는 자원발견 문제 해결을 위한 결정 알고리즘을 제안한다. 제안 알고리즘은 트리 합병(tree grafting) 기술과 포인터 점핑(pointer jumping) 기술을 사용하여, 이 문제를 O(log n) 병렬라운드와 O(mlog n) 연결 통신 복잡도와 $O(n^2\log n)$ 포인터 통신 복잡도에 해결하는데, 이러한 기술들은 [15,16]에서 PRAM(Parallel Random Access Machine) 모델 상에서 무방향 그래프(undirected graph)의 연결요소(connected components)들을 구하기 위해 사용된 적이 있는 기술들이다.

제안 알고리즘은, |V|=n 이고 |E|=m 인 주어진 방향성 그래프 G=(V,E)로부터, 각 방향성 에지가 역 에지를 가지는 대칭 그래프(symmetric directed graph) G′=(V,E′)을 구성한 다음, 구성된 그래프의 에지 정보를 이용하여 포리스트(forest)를 구성해 가면서, 트리의 높이를 줄이는 포인터 점핑과 트리의 개수를 줄이는 트리들의 합병을 반복하여, 높이가 1인 하나의 트리 즉 하나의 스타(star) 그래프가 생길 때까지 계속 수행한다.

다음은 본 논문에서 제안하는 자원발견 알고리즘이다. 입력으로 약하게 연결된 방향성 그래프 G=(V,E)가 주어질 때, 출력으로 완전 그래프를 생성한다. 즉 초기에 각 노드 v의 이웃 노드 들의 집합 Adj(v)는 $Adj(v) = \{w_i \mid \langle v, w_i\rangle \in E\}$이며, 수행을 마친 후에 각 노드 v의 Adj(v)는 v외의 다른 모든 노드들의 집합이 된다. 각 노드는 자신의 *ID*를 가지는데, v의 *ID*인 *ID*(v)는 v라 하자. 먼저 제안 알고리즘을 간단히 소개한 다음, 각 단계를 자세히 설명한다

알고리즘 자원발견

입력 : 입력 그래프 G=(V,E)는 연결된 방향성 그래프이다. 이때 |V|=n 이고 |E|=m 이다.
각 노드 v는 이웃노드의 집합 $Adj(v)=\{w_0, w_1, w_2, ..., w_{d-1}\}$를 가진다. 이때 d는 v의 차수(degree)를 나타낸다.

출력 : G의 완전 그래프, 즉 Adj(v)=V-{v}인 그래프.

[단계1] /* 대칭 그래프 G′=(V,E′)를 생성하고, 각 노

드의 parent 값을 초기화 한다. */
For each node v,
{Send ID(v) to each node of Adj(v)={w_0, w_1, w_2, ..., w_{d-1}};
When node v receives ID(u) from node u, update Adj(v)=Adj(v)∪{u};
parent(v)=v;}
/* 단계2에서 단계4까지 반복 수행한다. */
[단계2] /* 단계1에서 생성된 G'의 정보를 기반으로 트리와 트리를 합병한다. */
For each node v,
If parent(v)==parent(parent(v)) && (parent(w_i)<parent(v))
parent(parent(v))=parent(w_i);
[단계3] /* G'의 정보를 기반으로 스타를 트리에 합병한다. */
For each node v,
If **In_rooted_star(v)**&&(parent(w_i)≠parent(v))
parent(parent(v)) = parent(w_i);
[단계4] /* 포인터 점핑에 의해 트리의 높이를 줄인다. */
For each node v,
If **In_rooted_star(v)** then exit to Step 5;
else {parent(v)=parent(parent(v));
goto Step 2;}
[단계5] /* 구해진 한 개의 스타 그래프로부터 완전 그래프를 구한다. */
For each node v,
If v≠parent(v)
then send ID(v) to parent(v);
else {receive all the ID's of its children;
update Adj(v)=V-{v};
send Adj(v) to all its children;}
If v≠parent(v)
then {receive Adj(parent(v)) from its parent;
update Adj(v)=V-{v}; }

제안한 자원발견 알고리즘의 각 단계를 자세히 살펴보면, 먼저 단계1에서는 입력으로 주어진 방향성 그래프 G의 대칭 그래프 G'=(V,E')를 생성한다. 이를 위해 각 노드 v는 자신의 모든 이웃 노드 w_i에게 ID(v)를 보낸 다음, 또한 다른 노드들로부터 이러한 정보를 받아서, 이를 Adj(v)에 추가한다. 그리고 노드 v의 부모 parent(v)를 자기 자신인 v로 초기화한다.

그런 다음 단계2에서 단계4까지를 반복해서 수행하게 되는데, 단계2에서는 단계1에서 생성된 대칭 그래프 G'=(V,E')를 기반으로 트리들을 합병한다. 먼저 노드 v 또는 노드 parent(v)가 루트노드인지 검사하기 위해, 즉 parent(v)==parent(parent(v))인지 검사하기 위해, 노드 v는 노드 parent(v)에 parent(parent(v))값을 요구하고, 노드 parent(v)는 parent(parent(v))값을 v에게 보낸다. 이때 parent(v)==parent(parent(v))이면, parent(w_i)< parent(v)인 w_i를 찾기 위해, v는 모든 w_i에게 parent(w_i) 값을 요구하고, w_i는 parent(w_i)를 v에게 보낸다. v는 이들 중에서 parent(w_i) < parent(v)를 만족하는 w_i가 하나 이상 존재할 경우 임의로 하나를 선택하여 parent(v)에게 parent(w_i) 값을 보낸다. 이때, parent(v)가 여러 노드로부터 parent(w_i) 값을 받은 경우(그림 3 참조), 그 중 임의로 하나를 선택하여 parent(parent(v)) = parent(w_i)를 수행한다.

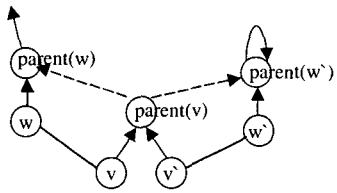


그림 3 하나 이상의 v가 parent(parent(v)) 값을 바꾸려고 하는 경우

단계3은 높이가 1인 트리인 스타를 다른 트리에 합병하는 과정이다. 먼저 노드 v가 스타에 속하면, parent(w_i)≠parent(v) 인 노드 w_i를 찾기 위해 모든 w_i에게 parent(w_i) 값을 요구하고, w_i는 parent(w_i) 값을 v에게 보낸다. 그러면 v는 이들 중에서 parent(w_i)≠parent(v)를 만족하는 w_i를 임의로 하나를 선택해서 parent(v)에게 parent(w_i) 값을 보낸다. 이때 parent(v) 역시 이러한 v 중 임의로 하나를 선택해서(그림 3참조) parent(v) 값을 parent(w_i) 로 해준다.

단계4는 포인터 점핑으로 트리들의 높이를 줄이는 과정이다. 포인터 점핑을 하기 전에 먼저 모든 노드들이 하나의 스타에 속해있는지 검사하여, 그런 경우에는 단계5로 넘어간다. 그리고 만약 노드 v가 스타 그래프에 속하지 않는 경우는 포인터 점핑에 의해 트리들의 높이를 줄인 다음 단계2로 간다. 포인터 점핑을 위해, 각 노드 v는 parent(v)에게 parent(parent(v)) 값을 요구하고 parent(v)는 이 값을 v에게 전송한다. v는 노드 parent(v)로부터 받은 이 값을 parent(v) 영역에 저장한다.

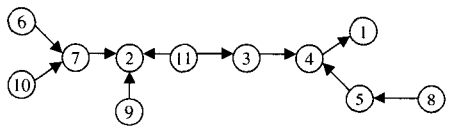
단계5는 지금까지의 과정으로부터 구해진 한 개의 스

타로부터 제안 알고리즘의 출력인 완전 그래프를 구하는 과정이다. 이때 루트가 아닌 모든 노드 v는 자신의 정보를 parent(v) 즉 루트에게 전송하고, 루트는 모든 자식 노드들로 받은 이 정보를 Adj(root)에 저장한 다음, 이 정보를 모든 자식 노드에게도 전송한다. 루트가 아닌 모든 노드 v도 이 정보를 Adj(v)에 저장하면 전체 그래프는 완전 그래프가 되고 수행을 종료된다. 즉, 모든 노드들이 다른 모든 노드들을 발견하게 된 것이다.

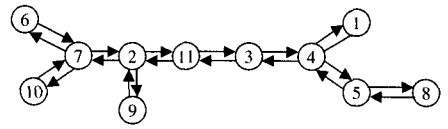
다음의 그림 4는 제안 알고리즘에 의해 자원 발견 문제를 해결하는 과정을 예를 들어 보여주고 있다. 입력으로 그림 4(a)와 같은 방향성 그래프가 주어질 때, 단계1에서는 4(b)와 같은 대칭 그래프를 생성하는데, 그 결과 각 노드의 이웃 노드들은 Adj(1)={4}, Adj(2)={7,9,11}, Adj(3)={4,11}, Adj(4)={1,3,5}, Adj(5)={4,8}, Adj(6)={7}, Adj(7)={2,6,10}, Adj(8)={5}, Adj(9)={2}, Ajd(10)={7}, Adj(11)={2,3}이 된다. 그리고 그림 4(c)처럼 모든 노드들이 자신을 parent로 가리키도록 한다.

다음, 단계2의 트리 합병과정을 보여주는 그림 4(d)에서는, 노드 4는 1과 3중에서 1을 parent로 하고, 노드 5는 4를, 노드 7은 2와 6중에서 6을, 노드 8은 5를, 노드 9는 2를, 노드 10은 7을, 노드 11은 2와 3중에서 2를 parent로 한, 트리들을 보여준다. 여기서 노드 2,9,11과 노드 3이 스타에 속함을 유의하라.

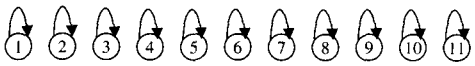
그리고 단계3에서는 이러한 스타들이 트리에 합병되는데, 노드 3은 4와 11중에서 4를 w_i로 선택하여 parent(parent(3))=parent(4) 즉 parent(3)=1로 하고, 노드 2는 7을 w_i로 선택하여 parent(parent(2))=parent(7) 즉 parent(2)=6로 하여, 그림 4(e)와 같이 두 개의 트리가 구성된다. 다음 단계4에서는 포인터 점핑을 수행하여 그림 4(f)와 같이 트리의 높이가 줄어들게 된다.



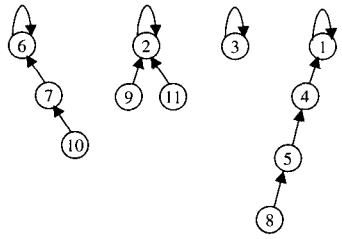
(a) 입력 그래프 G=(V,E), V={1,2,3,4,5,6,7,8,9,10,11}, E={<6,7>,<10,7>,<7,2>,<9,2>,<11,2>,<11,3>,<3,4>,<4,1>,<5,4>,<6,5>}



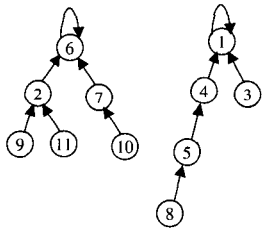
(b) 단계1 수행 후 구성된 대칭적 방향성 그래프 G′=(V,E′)



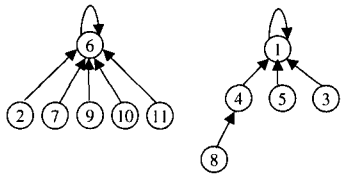
(c) 초기 트리들의 구조



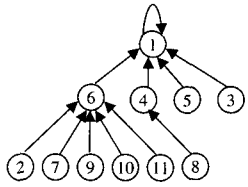
(d) 첫번째 반복의 단계2 수행 후



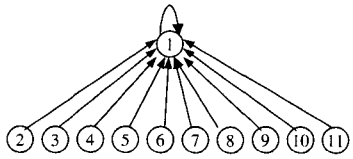
(e) 첫번째 반복의 단계3 수행 후



(f) 첫번째 반복의 단계4 수행 후



(g) 두번째 반복의 단계2 수행 후



(h) 두번째 반복의 단계4 수행 후

그림 4 자원발견 알고리즘 수행 예

그리고, 두 번째 반복의 단계2에서는 노드 11이 3을 w_i로 선택하여 parent(parent(11))=parent(3) 즉 parent(6)=1이 되어, 그림 4(g)와 같이 높이가 2인 하나의 트리가 생성된다. 다음, 단계3에서는 스타가 존재하지 않으므로 아무 동작도 하지 않게 되고, 단계4에서는 포인터 점핑을 수행하여 그림 4(h)와 같이 하나의 스타를 형성하게 된다.

그리고, 세 번째 반복의 단계2와 3에서는 아무 동작도 하지 않으며, 단계4의 조건문에서 모든 노드들이 스타에 속하므로 단계5로 가서 각 노드들은 루트 노드인 1번 노드에게 자신의 정보를 보내고, 루트노드는 이 정보들을 다시 모든 자식 노드들에게 보냄으로 알고리즘을 종료하게 된다.

자원발견 알고리즘의 단계3과 4에서 v가 스타 그래프에 속하는지 검사하기 위한 루틴은 다음과 같다.

In_rooted_star(v)

1. star(v) = true;
2. if parent(v)≠parent(parent(v))
   star(v) = false;
   star(parent(v)) = false;
   star(parent(parent(v))) = false;
3. star(v) = star(parent(v));
4. return star(v);

다음은 이 루틴이 O(n) 연결 통신복잡도와 포인터 통신복잡도, 그리고 O(1) 병렬 라운드에 수행됨을 보여준다.

[소정리 1] **In_rooted_star(v)**는 6n 연결 통신복잡도와 4n 포인터 통신복잡도, 그리고 4 병렬 라운드에 수행된다.

<증명> 단계2에서 조건을 검사할 때, parent(parent(v))를 알기 위해 parent(v)와 통신 해야 한다. 이때 메시지를 주고 받게 되므로 2개의 메시지가 전달되며, 데이타는 parent(parent(v)) 하나가 전달된다. 그리고 조건이 만족될 경우, 노드 v는 parent(v)에게 false를 전달하고, 다음 병렬 라운드에 또 한번 star(v)가 false인 노드 v가 parent(v)에게 false정보를 전달하면 된다. 단계3에서는 parent의 star값을 요구하고 받는 메시지가 필요하고 star(parent(v))데이타가 전달된다. 따라서 각 노드에서 주고 받는 메시지 수는 단계2의 조건문에서 2개, parent(v)에게 false를 전달하기 위해 1개, 또 한번 parent(v)에게 false를 전달하기 위해 1개, 그리고 단계3에서 2개, 모두 6개이며, 또한 통신되는 데이타의 수는 단계2에서 3개, 단계3에서 1개로 모두 4개가 이다.

그리고 단계1과 단계2는 조건문은 모두 1 병렬 라운드에 수행될 수 있고, 단계2의 나머지는 2 병렬라운드에 수행될 수 있으며, 단계3이 1 병렬 라운드에 수행될 수 있으므로, 모두 4 병렬 라운드에 수행될 수 있다.

따라서 **In_rooted_star(v)**는 6n 연결 통신복잡도와 4n 포인터 통신복잡도, 4 병렬 라운드에 수행될 수 있다. ■

다음의 소정리는 제안한 자원발견 알고리즘에서 단계3을 수행한 후에 스타가 존재한다면 모든 노드들이 하나의 스타에 속해 있다는 것을 보여준다.

[소정리 2] 자원발견 알고리즘의 단계3을 수행한 후, 스타가 존재한다면 모든 노드들이 하나의 스타에 속해 있다.

<증명> 임의의 반복에서, 단계3을 수행한 후 하나의 스타가 존재하고, 또한 이 스타와는 다른 하나 이상의 트리가 존재한다고 가정하자. 그렇다면, 단계1에서 만들어진 그래프가 연결된 대칭 그래프이기 때문에, 이 스타는 단계3에서 다른 트리로 합병되었어야 하므로 단계3을 수행한 후에는 스타로 남아 있을 수 가 없다. 그러므로, 단계3을 수행한 후 스타가 존재한다면, 이 스타 외에 다른 트리는 존재할 수가 없다. ■

아래의 소정리 3과 정리 1은 제안한 자원 발견 알고리즘의 단계2, 3, 4를 O(log n)번 수행하면 하나의 스타가 형성됨 보여준다.

[소정리 3] 입력 그래프 G에서, 단계 2, 3, 4를 k번 반복 수행한 후에 생성된 트리들의 높이의 총합을 $h_k(G)$라 하자. 이때 하나의 스타가 형성된 경우가 아니라면, $h_k(G) \leq (2/3)^k n$ 이다.

<증명> [15,16] 참조. ■

[정리 1] 자원 발견 알고리즘에서 단계2에서 단계4까지를 O(log n)번만 수행하면 하나의 스타를 형성된다.

<증명> 소정리 2에 의해 자원 발견 알고리즘의 수행이 끝이 났을 때에는 단 한 개의 스타만 존재함을 알 수 있고, 또한 소정리 3에 의해 트리들의 높이의 총합이 한번 반복할 때 마다 적어도 2/3 비율로 감소함을 알 수 있다. 그러므로 단계2에서 4까지를 O(log n)번만 반복하면 높이의 총합이 1인 한 개의 스타가 생성된다. ■

아래 정리 2는 제안한 자원 발견 알고리즘이 O(m log n) 연결 통신복잡도와 O(n^2log n)포인터 통신복잡도, 그리고 O(log n) 병렬 라운드에 자원 발견 문제를 해결함

수 있음을 보여주고 있다.

[정리 2] 제안한 자원 발견 알고리즘은 O(mlog n) 연결 통신복잡도와 O(n^2log n)포인터 통신복잡도, 그리고 O(log n) 병렬 라운드에 자원 발견 문제를 해결할 수 있다.

<증명> 단계1은 m개의 메시지와 m개의 데이타 그리고 1 병렬라운드에 수행되고, 단계2, 3, 4는 모든 노드들이 자신의 부모노드와 연결을 가지고, 자신과 에지가 존재하는 노드와 연결을 가지므로 O(n+m) 즉 O(m)의 연결 통신 복잡도를 가지며, O(n^2)의 포인터 통신 복잡도를 가진다는 것을 알 수 있다. 그리고 각 단계는 O(1)병렬 라운드에 수행됨을 알 수 있다. 따라서, 위의 정리 1에 의해 단계2에서 4까지가 최대 O(log n)번 반복되므로, 제안 알고리즘은 O(mlog n) 연결 통신복잡도와 O(n^2log n)포인터 통신복잡도와 O(log n) 병렬 라운드 내에 하나의 스타를 형성하게 된다. 그리고 하나의 스타가 형성된 다음, 단계5에서는 모든 노드들이 루트로 자신의 정보를 보내고, 루트는 이 정보들을 모아서 다시 모든 자식 노드에게 전달하므로, n-1개의 자식 노드와 루트 사이에 n-1개의 연결이 설정되고 루트에서 자식 노드로 최대 n개의 데이타가 전달되므로 모두 n^2개의 데이타들이 전달된다. 따라서 단계5는 O(n) 연결 통신복잡도와 O(n^2)포인터 통신복잡도와 2 병렬 라운드에 모든 노드들이 다른 모든 노드들을 알게 된다. 따라서 제안 알고리즘은 O(mlog n)연결 통신복잡도와 O(n^2log n)포인터 통신복잡도 그리고 O(log n) 병렬 라운드 내에 자원 발견 문제를 해결할 수 있다. ■

5. 결론 및 제언

본 논문에서는 분산 망에서 자원발견 문제를 해결하는 결정 알고리즘을 제안하였다. 제안 알고리즘은 트리 합병 기술과 포인터 점핑 기술을 사용하여 O(mlog n) 연결 통신 복잡도와 O(n^2log n) 포인터 통신 복잡도와 O(log n) 병렬라운드에 자원발견 문제를 해결하였다. 이 자원발견 알고리즘은 결정 알고리즘이며, 모든 노드들이 끝나는 시점을 스스로 판단할 수 있다는 장점을 가진다. 그리고 입력 그래프는 약하게 연결된 그래프이므로 m은 O(n)이 되며, 연결 통신 복잡도 O(mlog n)은 O(nlog n)과 같아진다.

또한, 이 알고리즘을 수행하는 중에 새로운 노드가 추가 되더라도 전체적인 수행에 영향을 주지 않고 쉽게 추가될 수 있다는 장점을 가지는데, 이는 새로운 노드가 하나 추가되면 추가된 노드에서 단계 1을 수행하여 초기화한 다음, 단계 2부터 다른 노드들과 동기를 맞춰 수행하면 되기 때문이다. 이때, 다른 노드들은 추가된 노드에 의해 영향을 받지 않고 작업을 계속할 수 있으며, 시간 복잡도에도 변화가 없게 된다.

그리고 알고리즘 수행 중 노드가 삭제 될 경우, 알고리즘을 일부 수정 함으로 수행될 수 있다. 예를 들어 2,3,4의 트리 구성과정에서 부모노드가 삭제 되어 응답이 없을 경우, 자신을 루트로 만들어 주는 과정이 추가 되어야 하고, 단계 5에서 루트가 아닌 노드가 삭제 될 경우는 문제가 되지 않으며, 루트가 삭제 된 경우에는 처음부터 다시 수행해야 한다. 이와 함께 단계 5의 경우 계산 및 통신이 루트 노드에서 집중적으로 일어나게 된다.

또한 제안 알고리즘은 이론적으로는 최적의 시간에 자원 발견 문제를 해결하고 있으나, 실제로 제안 알고리즘을 구현하기위해서는 병렬라운드가 실제로 얼마의 시간을 요구하게 될 것인가와 네트웍에서 동기화를 하기 위한 지연 시간 등의 문제를 고려하여야 한다.

앞으로 위에서 언급한 사항들을 고려하여 제안 알고리즘을 기반으로 실제 환경에서 적용 가능한 효율적인 알고리즘을 제안하고, 제안 알고리즘이 어느 정도의 성능을 나타내는지를 알아보기 위하여 모의 실험 등을 통해 비교 분석하고자 한다.

참 고 문 헌

[1] D. Agrawal, A. E. Abbadi, and R. Steinke, Epidemic algorithms in replicated database, In Proceedings of Sixth ACM SIAACT-SIGMOD-DIGART Symposium of Principles of Database Systems, pp.161-172, Tucson, Arizona, May 1997.

[2] S. Assmann and D. Kleitman, The number of rounds needed to exchange information within graph, SIAM Discrete Applied Maths, 6, pp.117-125, 1993.

[3] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, and J. Larson, Epidemic algorithms for replicated database maintenance, In Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, pp.1-12, Canada, August 1987.

[4] S. Even and B. Monien, On the number of rounds necessary to disseminate information, In Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, pp.261-266,1999.

[5] S. Hedetniemi, S. Hedetniemi, and A. Liestman, A survey of gossiping and broadcasting in communication networks, Networks, Vol. 18, pp.319-349, 1988.

[6] D. C. Oppen and Y. K. Dalal, The clearinghouse: A decentralized agent for locating named objects in distributed environment, ACM Transactions on Office Information Systems, 1(3), pp.230-253, July 1983.
[7] A. Pelc, Fault-tolerant broadcasting and gossiping in communication, Networks, 28(3): pp.143-156, October 1996.
[8] R. Renesse, Y. Minsky and M. Hayden, A gossip-style failure detector, 1998.
[9] Akamai Inc. www.akamai.com
[10] C. A. Thekkath, T. Mann, and E. K. Lee, Frangipani: A scalable distributed file system, In ACM Symposium on Operating System Principles, October 1997
[11] S. Keshav, An engineering approach to computer networking, Addison-Wesley, 1997.
[12] F. T. Leighton, Introduction to parallel algorithms and architectures: arrays, trees, and hypercubes, Morgan Kaufmann Publishers, 1992.
[13] M. Harchol-Halter, T. Leighton, and D. Lewin, Resource discovery in distributed networks, In Proceeding of 19th ACM Symposium on Principles of Distributed Computing, May 1999.
[14] J. Moy, OSPF version 2, rfc1583, 1994.
[15] J. JaJa, An introduction to parallel algorithms, Addison Wesley, 1992.
[16] Y. Shiloach, and U. Vishkin, An O(log n) parallel connectivity algorithm, Journal of Algorithms, 3(1), pp.57-67, 1982.

박 혜 경
1990년 창원대학교 전자계산학과 학사. 1992년 경북대학교 컴퓨터공학과 석사. 1992년 ~ 1993년 한국원자력연구소 근무. 1997년 경북대학교 컴퓨터공학과 공학박사. 1997년 ~ 현재 한국전자통신연구원 선임연구원.

유 관 우
1980년 경북대학교 전자공학과 졸업(공학사). 1982년 한국과학기술원 전산학과 졸업(이학석사). 1990년 미국 메릴랜드대학(University of Maryland) 졸업. 1982년 ~ 현재 경북대학교 컴퓨터공학과 부교수.