[illegible] 제6권 제1호(99.1)

최단경로문제를 해결하는 효율적인 분산 알고리즘

박 정 호† · 박 윤 용†

요 약

최단경로를 포함한 어떤 문제를 해결하는데 필요한 정보가 네트워크상의 프로세서에 분산되어 있는 상황에서 그들 정보를 교환하면서, 그 문제를 해결하는 알고리즘을 분산알고리즘(Distributed Algorithm)이라고 한다. 본 논문에서는 비동기식 네트워크에서 최단경로문제를 해결하는 분산 알고리즘을 제안한다. 일반적으로 분산 알고리즘은 메시지 복잡도와 이상시간 복잡도로 평가되는데, 본 논문에서 제안하는 분산 알고리즘의 메시지 복잡도와 이상 시간 복잡도는 각각 $O(n^{5/3})$와 $O(n\sqrt{n})$이다. 여기서, n은 네트워크상에 있는 프로세서수를 나타낸다.

A Distributed Algorithm for Weighted Shortest Path Problem

Jung-Ho Park† · Yoon-Young Park†

ABSTRACT

Consider the situation that informations necessary to solve a certain problem are distributed among processors on a network. It is called a distributed algorithm that in this situation each processor exchanges the message with adjacent processors to solve the problem. This paper proposes a distributed algorithm to solve the problem that constructs the weighted shortest path tree in an asynchronous network system. In general, a distributed algorithm is estimated by the number of messages(message complexity) and the number of unit complexity(ideal time complexity). Message complexity and ideal-time complexity of the distributed algorithm proposed in this paper are $O(n^{5/3})$ and $O(n\sqrt{n})$ respectively, where n is the number of processors on the network.

1. 서 론

컴퓨터 네트워크상에서는 프로세서가 갑자기 정지하거나 정지상태에 있던 프로세서가 회복되는 경우가 있다. 또 프로세서간을 연결하는 링크도 사용을 못하게 되거나 두 개의 프로세서 사이에 새로 링크를 설치하는 경우가 있으므로 네트워크 형상은 일정하지 않고 동적으로 변화하는 것이라고 생각할 수 있다. 동적으로 네트워크 형상이 변화하는 네트워크 환경에 있어서는 그 토폴로지 정보를 하나의 프로세서가 통괄해서 관리하는 것보다 각 프로세서가 자기에 관한 토폴로지 정보만을 분담해서 관리하는 것이 바람직하다. 이와같이 어떤 문제를 해결하는데 필요한 정보가 네트워크상의 프로세서에 분산되어 있는 상황에서 그들 정보를 교환하면서 그 문제를 해결하는 알고리즘을 분산알고리즘(Distributed Algorithm)이라고 한다.

분산알고리즘이 실행되는 네트워크 환경에서는 메시지수가 코스트의 대부분을 차지하고, 또 메시지의 전송 시간이 분산알고리즘이 처리를 종료하는데 걸리는 시간의 대부분을 차지한다. 따라서, 분산알고리즘의 효율은 주로 메시지 복잡도(message complexity)와 이상시간복잡도(ideal tme cmplexity)로 평가된다. 메시지 복잡도란 분산알고리즘 실행중에 네트워크에서 교환되

※ 본 논문은 정보통신부에서 시행하는 대학기초 지원사업에서 일부를 지원받았음.
† 종신회원 : 선문대학교 정보과학부 교수
논문접수 : 1998년 8월 2일, 심사완료 : 1998년 10월 21일

는 메시지의 총수를 말한다. 대부분의 네트워크 모델에서는 메시지의 전송시간에 관해서 유한이라는 것이외에는 아무 것도 가정하지 않는다(비동기식 네트워크). 그러므로 일반적으로 순차알고리즘의 평가에 적용되고 있는 시간복잡도(time complexity)의 개념은 적용할 수 없다. 따라서 분산알고리즘의 실행시간을 평가하는 데는 이상시간복잡도라는 것을 채택하고 있는데, 이상시간복잡도란 프로세서 내에서의 처리시간을 무시하고, 메시지가 링크를 통과하는 전송시간을 1단위시간으로 했을 때의 분산알고리즘이 종료할 때까지의 단위시간수를 말한다.

네트워크 상에서 임의의 프로세서를 중심으로 하여 다른 모든 프로세서와의 최단경로(shortest path)를 구하는 문제는 효율적인 메시지 전송을 위해서 중요한 문제이다. 지금까지 생성트리구성문제(spanning tree constructing problem)라든가 리더선택문제(leader election problem) 등을 포함한 각종 문제를 해결하기 위한 많은 분산알고리즘이 제안되었으나[1,2,3,4,6,7,9], 링크에 무게(weight)가 할당되어 있는 유방향 그래프에 대한 최단경로문제(weighted shortest path problem)를 해결하는 분산알고리즘은 제안되지 않았다. 그러나, 최단경로문제는 Dijkstra의 순차알고리즘을 분산환경에 적용시키면 메시지 복잡도와 이상시간복잡도가 모두 $O(n^2)$로 해결할 수가 있다[5].

본 논문에서는 최단경로문제를 해결하는 효율적인 분산알고리즘 즉, 메시지 복잡도와 이상 시간 복잡도가 각각 $O(n^{5/3})$와 $O(n\sqrt{n})$인 알고리즘을 제안한다. 여기서, n은 네트워크상에 있는 프로세서수를 나타낸다. 따라서, 본 논문의 알고리즘은 메시지 복잡도와 이상시간복잡도 면에서 Dijkstra의 분산버전 알고리즘보다 효율적이다.

2. 정 의

본 논문에서는 무게가 있는 유방향연결 그래프(weighted connected directed graph)만을 취급하며, 본 논문에서 사용하는 유방향 연결 그래프에 관한 용어를 다음과 같이 정의한다. 단, 다중링크는 없는 것으로 한다.

[정의 1] 최단경로(shortest path) : 임의의 두 프로세서를 연결하는 경로(path)중에서 그 경로상에 있는 링크에 할당된 무게(weight)의 합이 가장 적은 경로를 말한다. 이후, 임의의 링크(u, v)에 할당된 무게를 $w(u, v)$로 나타내기로 한다.

[정의 2] 최단경로문제 : 하나의 지정된 프로세서 r과 r이외의 모든 프로세서에 대해 최단경로를 구하는 문제를 말한다. 이후, 지정된 프로세서 r을 루트(root)라고 한다.

네트워크와 분산 알고리즘에 관해 다음과 같은 가정을 둔다.

[가정 1] 네트워크상에 공유메모리는 없고, 프로세서간의 통신은 링크를 통한 메시지 교환만으로 행한다. 프로세서 u가 인접 프로세서 v에게 보낸 메시지는 보낸 순서대로 유한시간내에 도중에 없어지지 않고 v에게 반드시 보내진다.

[정의 3] 임의의 무게있는 유방향 그래프를 G=(V, E)라고 한다. G의 기초 그래프 N =(P, L)을 다음과 같이 정의한다.

$$P = V$$
$$L = \{(u, v) \mid (u, v)\in E \text{ 또는 } (v, u)\in E\}$$

즉, G의 유방향변을 무방향변으로 치환하고, 이로인해 다중변이 생길 경우는 그들을 하나의 무방향변으로 치환함으로써 얻어지는 네트워크가 G의 기초그래프이다. G의 각 $(u, v)\in E$에 대해 N의 프로세서 u, v가 각각 (u, v)에 할당된 무게(weight)를 알고 있을 때, N에 G의 무게가 주어져있다고 한다.

[정의 4] 최소경로구성문제(Weighted Spanning Path Problem : 이하 WSP문제라고 한다) :

WSP문제는 다음의 초기상태에서 시작하여 최종상태에 이르는 문제를 말한다.

(초기상태) N을 임의의 네트워크라고 하고, N을 기초네트워크로 하는 임의의 유방향그래프를 G라고 한다. G의 무게가 N에 주어져 있고, N의 하나의 프로세서(r이라고 한다)가 지정되어 있다.

(최종상태) G상에서 r과 다른 모든 노드사이의 최소경로가 구성되어 있다. 즉, N의 각 프로세서 v가 최소경로 r-v상에서 v의 아버지와 최소경로 r-v의 값 그리고 아들을 알고 있다.

[정의 5] 분산알고리즘 : 분산알고리즘이란 각 프로세서가 실행하는 프로그램으로 구성된다. 자발적으로 프로그램 실행을 개시하는 프로세서를 시작 프로세서

start processor)라고 한다. 시작 프로세서 이외의 프로세서는 인접 프로세서로부터 메시지를 받으면 프로토콜 실행을 개시하는데, 여기서 시작 프로세서는 하나라고 가정한다. 메시지 전송 지연의 차이, 프로세서 동작 속도의 차이 등에 의해 여러 가지 실행 과정이 발생할 수 있으나, 어느 실행 과정에 있어서도 유한개의 명령 실행후에 모든 프로세서가 프로그램 실행을 끝냈을 때 분산 알고리즘이 종료했다고 한다.

[가정 2] 각 프로세서가 실행하는 프로그램은 동일하다.

[가정 3] 분산알고리즘의 개시시에 각 프로세서는 인접링크에 할당된 무게(weight)를 알고 있다.

분산 알고리즘의 평가는 일반적으로 메시지량과 시간을 이용해서 행한다. 시간평가는 메시지의 전송지연시간과 각 프로세서의 동작 속도에 차이가 있기 때문에 용이하지 않아서 일반적으로 이상시간복잡도라는 개념을 이용한다.

[정의 6] 메시지 복잡도는 분산알고리즘 실행 도중에 네트워크의 모든 프로세서 사이에서 교환되는 메시지 수이다. 또, 이상시간복잡도는 프로세서내에서 처리시간을 무시하고, 메시지가 링크에 전달되는 전송시간을 1단위시간으로 했을 때 알고리즘이 종료할 때까지의 단위시간수이다.

3. 최단경로문제를 해결하는 분산알고리즘

본 논문에서는 무게가 있는 유방향 연결 그래프만을 취급한다. 이하에서는 G=(V, E)를 임의의 무게가 있는 유방향 연결 그래프라고 한다. 여기서는 G와 임의의 프로세서 r이 지정되었을 때, r을 루트로 하는 최단 경로 문제를 해결하는 분산 알고리즘을 제안한다. 이후, r과 다른 모든 프로세서간 u의 최단경로 r-u에 의해 구성되는 트리를 최단경로트리 WSPT T=(V, E') (E'⊆E)라고 한다.

3.1 단순한 알고리즘

루트프로세서 r은 각 프로세서의 경로를 구하도록 하기 위해 각 인접 프로세서 v에게 <PATH($w(r, v)$)>를 보낸다.

각 프로세서 u는 인접프로세서 v로부터 <PATH(val)>을 받았을 때 다음과 같이 한다. 이후, 각 프로세서 u에 대해 루트와의 거리를 기억한 변수를 $DIST_u$라고 한다.

(case 1) $val \geq DIST_v$일 때

프로세서 v에게 <NO>메시지를 보낸다.

(case 2) $val < DIST_v$일 때

$DIST_v$를 val로 하고, v를 u의 아버지로 한 후, 인접프로세서가 없으면 v에게 <YES>를 보낸다. 만일, 인접프로세서가 있으면 인접프로세서 z에게 <PATH($DIST_v+w(u, z)$)>를 보낸다.

인접프로세서에게 <PATH(val)>를 보낸 프로세서는 이들 프로세서로부터 <NO> 또는 <YES>를 회신으로 받았을 때 아버지에게 <YES>를 보낸다.

이 알고리즘을 평가해 보면, 각 프로세서는 최악의 경우 n-1번 거리를 갱신하고, 거리를 갱신할 때마다 인접프로세서에게 <PATH(val)>메시지를 보낸다. 따라서, 각 링크에는 최악의 경우 n-1개의 <PATH(val)> 메시지가 전송되므로, 전체적으로 O(ne)개의 메시지가 필요하다.

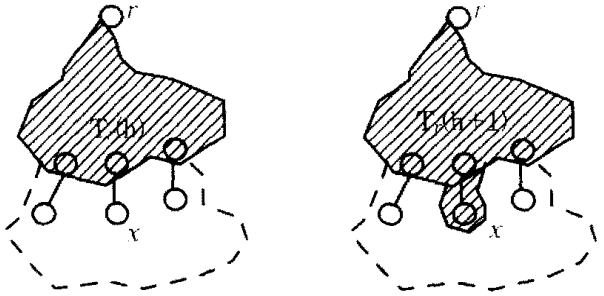
3.2 Dijkstra알고리즘의 분산화

3.1의 알고리즘의 경우, 각 프로세서가 거리를 최대 n-1번 갱신함으로 인해 메시지 효율이 나쁜데 비해, 여기서 설명할 알고리즘의 특징은 각 프로세서가 거리를 두 번 이상 갱신하지 않는다는 점이다.

이를 위해 Dijkstra알고리즘을 분산 네트워크에 적용한 경우의 기본 아이디어는 최단경로트리 WSPT T에 포함될 프로세서 중에서 최단거리가 작은 순으로 하나씩 최단 경로를 확정해 가는 것이다[5]. 즉, 처음에는 루트 프로세서 r만으로 구성된 서브트리 $T_r(0)=(V'', E'')$ ($V''=\{r\}$, $E''=\phi$)에서 시작하여 매번 하나의 프로세서를 서브트리에 추가해 감으로써, 이 알고리즘에서는 n번의 반복을 통해 최단경로트리 WSPT T를 완성하게 되는데, 각 $i(1\leq i\leq n)$단계를 시작하는 상황에서는 WSPT T의 서브트리인 $T_r(h)$에서 시작해서 하나의 프로세서를 추가하여 $T_r(h+1)$로 확장하게 된다. 즉, $i=h+1$이 성립한다. 이후, 프로세서 s를 루트로 하는 크기가 m인 서브트리를 $T_s(m)$로 나타내기로 한다. 단, 서브트리의 크기에 루트는 포함하지 않는 것으로 한다.

$T_r(h)$가 다음 (그림 1)의 (a)과 같이 구성되었다고 하고, 이때 최단경로 r-x가 가장 작다고 할 때, i번째

단계에서는 프로세서 x를 찾아서 $T_r(h)$에 추가하여 (b)와 같이 $T_r(h+1)$로 확장한다.



(a) 제 i번째 단계의 시작시 (b) 제 i번째 단계의 종료시

(그림 1) 분산 Dijkstra알고리즘의 개략

상기 프로시저를 좀더 자세히 설명하면 다음과 같다.

루트 r은 서브트리 $T_r(h)$를 확장하기 위해, 각 i번째 단계에서 서브트리 $T_r(h)$에 속하는 프로세서에게 <COMPUTE>라는 메시지를 보낸다. 이 메시지는 각 프로세서 $v(\in T_r(h))$로부터 확장해 갈 수 있는 인접 프로세서 $w(\notin T_r(h))$에 대한 거리를 계산하도록 하는 메시지이다. 아버지로부터 <COMPUTE>를 받은 프로세서 u는 다음과 같이 한다.

(case 1) $T_r(h)$상의 아들이 있을 때

(case 1.1) Tr(h)에 속하지 않는 프로세서가 있을 때 이들 프로세서 중에서 거리가 가장 가까운 프로세서(v라고 한다)에게 <PATH($DIST_u+w(u, v)$)>를 보낸다. 그리고, 아들에게는 <COMPUTE>를 보낸다.

(case 1.2) Tr(h)에 속하지 않는 프로세서가 없을 때 아들에게 <COMPUTE>를 보낸다.

(case 2) $T_r(h)$상의 아들이 없을 때

(case 2.1) $T_r(h)$에 속하지 않는 프로세서가 있을 때 이들 프로세서 중에서 거리가 가장 가까운 프로세서(v라고 한다)에게 <PATH($DIST_u+w(u, v)$)>를 보낸다.

(case 2.2) $T_r(h)$에 속하지 않는 프로세서가 없을 때 아버지에게 <NO>를 보낸다.

인접프로세서 w로부터 <PATH(val)>을 받은 프로세서 u는 $DIST_u \geq val$일 때 w에게 <NO>를 보내고, $DIST_u < val$일 때 w를 아버지로 하고, $DIST_u$를 val로 한 후, w에게 <CAN(val)>을 보낸다.

<COMPUTE>메시지나 <PATH(val)>메시지를 인접프로세서에게 보낸 프로세서는 이들 모든 인접프로세서로부터 <NO>나 <CAN(val)>을 받았을 때, 가상적은 값(min이라고 한다)을 대입한 <CAN(min)>메시지를 아버지에게 보낸다. 이렇게 함으로써 $T_r(h)$에 추가한 후보 프로세서에 대한 거리 정보는 서브트리 $T_r(h)$를 통해 루트로 올라오는데, 루트는 $T_r(h)$에 추가할 후보 중에서 최소값을 가진 프로세서를 서브트리 $T_r(h)$에 포함시킴으로써 $T_r(h+1)$로 확장하여 i번째 단계가 종료하게 된다.

이 알고리즘을 평가해 보면, 각 i번째 단계에서 루트가 보내는 <COMPUTE>메시지는 서브트리 $T_r(h)$를 통해 보내지고 <COMPUTE>메시지에 대한 회신인 <CAN(val)>메시지도 $T_r(h)$를 통해 루트 프로세서 r로 다시 보내지므로, 각 i번째 단계에서는 2n개의 메시지가 전송된다.이 알고리즘은 n번 반복되므로, <COMPUTE>메시지의 복잡도는 $O(n^2)$이 된다. 그리고 <PATH(val)>메시지는 각 링크에 한 번씩 보내지며, <NO>와 <CAN(val)>메시지가 회신이므로 2e개의 메시지가 전송된다. 따라서, 본 알고리즘의 메시지 복잡도는 $O(n^2)$이 된다. 마찬가지로, 이상 시간 복잡도가 $O(n^2)$임을 평가할 수 있다.

3.3 알고리즘 SPA의 개요

3.1의 알고리즘과 3.2의 알고리즘을 비교해 보면, 3.1의 알고리즘에서는 각 프로세서의 거리를 갱신하는 횟수가 많은데 비해, 3.2의 알고리즘은 각 프로세서가 두 번 이상 거리를 갱신할 수 없도록 하기 위해 <COMPUTE>라는 컨트롤용 메시지를 사용하고 있다(표 1 참조).

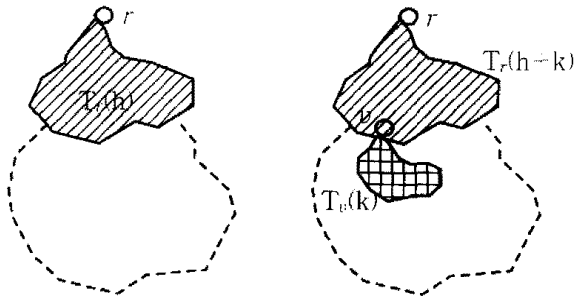
〈표 1〉 메시지 비교

	<PATH(val)>	<COMPUTE>
3.1의 알고리즘	O(ne)	없음
3.2의 알고리즘	O(e)	$O(n^2)$

상기 표로부터 3.1의 알고리즘은 최단거리를 구하기 위한 <PATH(val)>메시지가 많이 소요되는데 비해, 3.2의 알고리즘에서는 <PATH(val)>메시지는 줄어들었지만 컨트롤용 메시지인 <COMPUTE>를 많이 필요로 한다는 것을 알 수 있다.

본 논문에서 제안하는 알고리즘 SPA(Shortest Path Algorithm)는 이들 두 알고리즘의 절충안으로서, 각 프로세서가 거리를 갱신할 수 있는 횟수를 k개로 한다.

수. Dijkstra알고리즘의 경우 최단경로트리 WSPT T의 서브트리인 $T_r(h)$의 크기를 1씩 증가시켜 $T_r(h+1)$로 확장시키는데 비해, 알고리즘 SPA에서는 알고리즘의 효율화를 위하여, 각 i($1 \le i \le n/k$)번째 단계에서는 최단경로트리 WSPT T의 서브트리인 $T_r((i-1)\cdot k)$의 크기를 k개씩 증가시켜서 $T_r(i\cdot k)$로 확장해 간다(그림 2 참조). 이로인해, 알고리즘 SPA에서는 3.2의 알고리즘에서 e개를 필요로 하는 <PATH(val)>는 약간 늘어나는 반면, n^2개를 필요로 하는 <COMPUTE>메시지는 줄어들게 되어, 전체적으로는 3.2의 알고리즘보다 효율적인 알고리즘이 된다.



(a) 제 i번째 단계의 시작시 (b) 제 i번째 단계의 종료시

(그림 2) 알고리즘 SPA의 개략

그리고, Dijkstra알고리즘의 경우, 각 i($1 \le i \le n$)번째 단계에서 각 프로세서 $v(\in T_r(h))$는 인접 프로세서 $x(\notin T_r(h))$들에 대해서만 루트로부터의 거리 $DIST_v + w(v, x)$를 계산해서 최소값을 아버지에게 보낸다. 즉, 각 프로세서 v는 크기가 1인 서브트리 $T_v(1)$을 구성하는데 i+1단계에서는 $T_r(h)$에 속하는 모든 프로세서 z가 $T_z(1)$을 구성하는 것이 아니라, i번째 단계에서 프로세서 w가 서브트리 $T_r(h)$에 포함되었다고 할 때 다음 단계인 i+1번째 단계에서는 프로세서 w만이 $T_w(1)$을 구성한다.

알고리즘 SPA에서는 서브트리 $T_w(k)$를 구성해서 그들 서브트리 중에서 가장 적은 값을 가진 서브트리를 WSPT T에 추가하는 작업을 반복함으로써 WSPT T를 구성하게 되며, 각 i번째 단계에서 k개의 프로세서가 WSPT T의 서브트리에 추가되므로, n/k번 반복함으로써 WSPT T가 구성된다.

알고리즘 SPA의 각 i($1 \le i \le n/k$)번째 단계에서 프로세서의 동작은 다음과 같다.

[스텝 1] 루트는 i-1번째 단계에서 구성된 WSPT T의 서브트리인 $T_r((i-1)\cdot k)$에 속하는 각 프로세서 v에게 서브트리인 $T_r((i-1)\cdot k)$를 확장한다는 의미를 가진 메시지<COMPUTE>를 전송함으로써 i번째 단계를 시작하게 된다. 단, 스텝1은 첫 번째 단계(i=1)에서는 생략된다.

[스텝 2] <COMPUTE>메시지를 받은 프로세서 중에서 자신을 루트로 하는 서브트리를 보유하지 않은 각 프로세서 v는 $T_v(k)$를 구성한다. 즉, 전단계인 i-1번째 단계에서 새로 $T_r((i-1)\cdot k)$에 추가된 프로세서 또는 서브트리의 손상으로 인해 k개의 프로세서를 갖지 않은 프로세서 v즉 $|T_v(k)| < k$를 만족하는 프로세서 v만이 서브트리 $T_v(k)$를 새로 또는 재구성한다. 이때, 프로세서 v는 $T_v(k)$를 구성하기 위해 Dijkstra 알고리즘에서 사용된 <PATH(*val*)>과 <COMPUTE>메시지를 사용한다.

[스텝 3] 스텝2에서 구성된 각 서브트리의 루트인 각 프로세서 v는 서브트리 $T_v(k)$중에서 루트와의 거리가 가장 먼 값(*val*이라고 한다)을 가진 메시지<CAN(*val*)>을 아버지에게 전송한다. 메시지<CAN(*val*)>를 받은 각 프로세서는 모든 아들로부터 메시지<CAN(*val*)>을 받았을 때, 이들 *val*중에서 가장 적은 값을 계산해서 아버지에게 <CAN(*val*)>를 보낸다. 이와같이 메시지 <CAN(*val*)>을 통해 i번째 단계에서 WSPT T의 서브트리인 $T_r((i-1)\cdot k)$에 추가될 서브트리에 대한 데이터가 루트로 전송된다.

[스텝 4] 루트 r은 아들로부터 받은 <CAN(*val*)>중에서 가장 적은 *val*값을 가진 서브트리 $T_v(k)$을 WSPT T의 서브트리인 $T_r((i-1)\cdot k)$에 추가하여 $T_r(i\cdot k)$로 확장함으로써 i번째 반복을 종료한다. 그리고, 추가할 프로세서가 있으면 i+1번째 반복을 수행하고, 추가할 프로세서가 없으면 알고리즘을 종료한다.

4. 알고리즘 SPA의 평가

본 논문에서 제안하는 분산 알고리즘 SPA의 정당성을 증명하기로 한다.

[정리 1] 분산 알고리즘 SPA는 최단경로문제를 해결한다.
(약증) 알고리즘 SPA에서는 k개의 노드로 구성된 서브트리를 구성해서 그들 서브트리 중에서 가장 적은

값을 가진 서브트리를 WSPT T에 추가하는 작업을 반복함으로써 WSPT T를 구성한다. 즉, 각 i(1≤i≤n/k)번째 단계에서 k개의 프로세서를 WSPT T의 서브트리에 추가하여, n/k번 반복함으로써 WSPT T를 구성하게 된다.

알고리즘 SPA의 정당성은 i(1≤i≤n/k)에 관한 귀납법으로 증명하기로 한다.

(i=1일 때) 루트 프로세서 r이 k개의 프로세서로 서브트리 $T_r(k)$를 구성하는 단계로서, $T_r(k)$에 속하는 각 프로세서가 루트로부터의 최단거리 및 경로를 제대로 계산한다는 것은 Dijkstra알고리즘의 정당성으로부터 증명할 수 있다.

(i≤h일 때) i·k 개의 프로세서로 구성된 WSPT T의 서브트리 즉, $T_r(i\cdot k)$가 구성되어 있으며, $T_r(i\cdot k)$에 속하는 각 프로세서 w는 $T_w(k)$를 구성하고 있다. w를 루트로 하는 서브트리 $T_w(k)$상의 각 프로세서 z는 w로부터의 최단거리 및 경로를 알고 있으며, w는 $T_w(k)$ 중에서 가장 큰 값 $val(w)$즉, 루트로부터 거리가 가장 먼 프로세서의 (잠정적인) 거리를 알고 있다.

(i=h+1일 때) 루트 r은 $T_r(i\cdot k)$에 속한 각 프로세서 w가 가지고 있는 $val(w)$중에서 가장 적은 값을 가진 서브트리($T_p(k)$라고 한다)를 $T_r(i\cdot k)$에 추가하여 $T_r((i+1)\cdot k)$을 구성하는데, $T_p(k)$에 속하는 각 프로세서가 루트로부터의 최단거리 및 경로를 제대로 계산한다는 것도 Dijkstra알고리즘의 정당성으로부터 증명할 수 있다.

다음은 분산 알고리즘 SPA의 복잡도를 평가하기로 한다.

[정리 2] 분산 알고리즘 SPA의 메시지 복잡도는 $O(n^{3/3})$이다.

(증명) 본 알고리즘 SPA의 각 i번째 단계에서는 k개의 프로세서로 구성된 서브트리를 $T_r((i-1)\cdot k)$에 추가해서 $T_r(i\cdot k)$로 확장시켜가므로, (n/k)번을 반복함으로써 모든 프로세서를 포함하는 WSPT T를 구성할 수 있다.

다음은 각 i번째 단계에서 소요되는 메시지를 계산하기로 하자.

스텝 1에서 루트 r은 <COMPUTE>메시지를 전송함으로써 WSPT T의 서브트리인 $T_r((i-1)\cdot k)$를 이용하여 $T_r((i-1)\cdot k)$에 속하는 프로세서 v가 서브트리 $T_v(k)$를 구성하게 하므로 O(n)메시지가 필요로 한다. 스텝 2에서는 각 프로세서 v가 서브트리 $T_v(k)$를 구성하는데, 서브트리는 Dijkstra알고리즘을 이용하므로 하나의 서브트리를 구성하는데는 $O(k^2)$메시지가 필요하며, 서브트리 $T_v(k)$를 구성하는 프로세서는 전단계인 i-1번째 단계에서 WSPT T의 서브트리 $T_r((i-1)\cdot k)$에 새로 포함된 k개의 프로세서와, 서브트리 $T_v(k)$가 손상되어 k개의 프로세서를 갖지 못한 프로세서가 k개이하 이므로 모두 2k개의 프로세서가 다시 서브트리를 구성한다. 따라서, 스텝2에서 필요로 하는 메시지는 $O(k^3)$메시지이다. 스텝 3과 4에서는 <CAN(val)>메시지를 이용하여 WSPT T의 서브트리 $T_r((i-1)\cdot k)$에 추가할 서브트리를 결정해서 $T_r(i\cdot k)$에 추가하므로, 스텝 3과 4에서는 O(n)메시지가 소요된다.

따라서, 알고리즘 SPA의 메시지복잡도는 $(n+k^3)(n/k)$이 된다. 이때, 양쪽이 균형을 유지하도록 하기 위해 $k=(n^{1/3})$로 두면 메시지복잡도는 $O(n^{3/3})$이 된다.

[정리 3] 분산 알고리즘 SPA의 이상 시간 복잡도는 $O(n\sqrt{n})$이다.

(증명) 본 알고리즘 SPA의 각 i번째 단계에서는 k개의 프로세서로 구성된 서브트리를 $T_r((i-1)\cdot k)$에 추가해서 $T_r(i\cdot k)$로 확장시켜가므로, (n/k)번을 반복함으로써 모든 프로세서를 포함하는 WSPT T를 구성할 수 있다.

다음은 각 i번째 단계에서 소요되는 이상시간복잡도를 계산하기로 하자.

스텝 1에서 루트 r은 <COMPUTE>메시지를 전송함으로써 WSPT T의 서브트리인 $T_r((i-1)\cdot k)$를 이용하여 $T_r((i-1)\cdot k)$에 속하는 프로세서 v가 서브트리 $T_v(k)$를 구성하게 하므로 O(n)시간을 필요로 한다. 스텝 2에서는 각 프로세서 v가 서브트리 $T_v(k)$를 구성하는데, 서브트리는 Dijkstra알고리즘을 이용하므로 스텝2에서는 $O(k^2)$시간이 필요하다. 스텝 3과 4에서는 <CAN(val)>메시지를 이용하여 WSPT T의 서브트리 $T_r((i-1)\cdot k)$에 추가할 서브트리를 결정해서 $T_r((i-1)\cdot k)$에 추가하므로, 스텝 3과 4에서는 O(n)메시지가 소요된다.

따라서, 알고리즘 SPA의 이상시간복잡도는 $(n+k^2)(n/k)$이 된다. 이때, 양쪽이 균형을 유지하도록 하기 위해 $k=\sqrt{n}$으로 두면 이상시간복잡도는 $O(n\sqrt{n})$이 된다.

5. 결 론

본 논문에서는 비동기식 네트워크에서 최단경로문제를 해결하는 메시지 복잡도와 이상 시간 복잡도가 각각 $O(n^{3/2})$와 $O(n\sqrt{n})$인 분산 알고리즘을 제안하였다. 최단경로문제를 해결하는 기존 결과로는 Dijkstra의 순차알고리즘의 분산버전을 들 수 있는데, 이 경우 메시지 복잡도와 이상시간복잡도가 모두 $O(n^2)$이 되어, 본 논문의 알고리즘이 메시지복잡도와 이상시간복잡도 면에서 Dijkstra의 분산 환경알고리즘보다 효율적인 알고리즘이 된다.

참 고 문 헌

[1] B.Awerbuch : "Complexity of network synchronization," *Journal of ACM*, Vol.32, No.4, pp.804-823(Oct. 1985).

[2] B.Awerbuch and R.G.Gallager : "A new distributed algorithm to find breadth first search trees," *IEEE Trans. on Information Theory*, Vol.IT-33, No.3, pp.315-322(1987).

[3] B.Awerbuch : "Distributed shortest paths algorithm," Proc. of 21st *Symposium on Theory of Computing*, pp.490-500(1989).

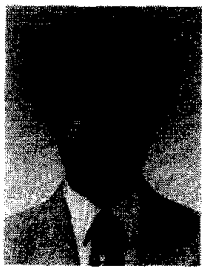
[4] M.Ahuja and Y.Zhu : "An efficient distributed algorithms for finding articulation points, bridges and biconnected components in asynchronous network," In Proc. 9th *Conference on Foundations of Software Technology and Theoretical Computer Science*(LNCS 405), pp.99-108, 1989.

[5] T.H.Cormen, C.E.Lerserson and R.L.Rivest : "Introduction to Algorithms," The MIT Press, 1990.

[6] J.Park, T.Masuzawa, K.Hagihara and N.Tokura : "An efficient distributed algorithm for breadth first spanning tree problem," *Journal of IEICE (D)*, Vol.J71-D, No.7, pp.1576-1188(1988).

[7] B.Swaminathan and K.J.Goldman : "An incremental distributed algorithm for computing biconnected components," I Proc. 8th *International Workshop on Distributed Algorithms*(LNCS 857), pp.238-252, 1994.

[8] T.Kameda and M.Yamashita, "Distributed Algorithms", Kindai-Kagaku-Sya, 1994.



박 정 호

e-mail : jhpark@omega.sunmon.ac.kr

1980년 성균관대학교 사범대학 졸업(문학사)

1980년~1982년 성균관대학교 경영대학원 정보처리학과(경영학석사)

1985년~1987년 日本 오사까대학교 대학원 정보공학전공(공학석사)

1987년~1990년 日本 오사까대학교 대학원 정보공학전공(공학박사)

1996년~현재 한국정보처리학회 총무이사

1991년~현재 선문대학교 정보과학부 부교수

관심분야 : 분산알고리즘, 원격교육, XML, 소프트웨어 공학

박 윤 용

e-mail : yypark@omega.sunmon.ac.kr

1983년 숭전대학교 계산통계학과 졸업(학사)

1985년 서울대학교 대학원 계산통계학과(이학 석사)

1994년 서울대학교 대학원 계산통계학과(이학 박사)

1985년~1992년 한국전자통신연구소 연구원

1992년~현재 선문대학교 전자계산학과 조교수

관심분야 : 분산처리운영체제, 멀티미디어시스템, 고장감래 시스템