OR 모델 기반의 분산 교착상태 발견 및 복구 기법
(A Distributed Deadlock Detection and Resolution Algorithm for the OR Model)

이 수 정 †
(Soojung Lee)

요 약 분산시스템에서는 어느 한 사이트가 전체 시스템의 상태 정보를 알 수 없기 때문에 분산 교착상태의 발견은 어려운 문제로 알려져 왔다. 본 논문은 분산 교착상태의 발견 및 해결을 위한 시간 효율적인 알고리즘을 제안한다. 알고리즘의 시작노드는 교착상태 발견을 위한 메시지를 전파하고 이에 대한 응답 정보로부터 wait-for graph를 구축한다. 제시한 알고리즘은 기존에 비해 교착상태를 발견하는데 걸리는 시간을 반으로 단축시키는 장점이 있다. 또한 대부분의 연구결과가 단지 알고리즘의 시작노드가 교착상태에 속하였는지의 여부를 판단할 수 있는 것과는 달리 본 알고리즘은 알고리즘을 수행하는 모든 노드들에 대해 그같은 결과를 말해준다. 이러한 특성으로 인해 교착상태는 보다 신속히 발견될 수 있다. 더우기 이제까지 교착상태의 해결에 대해 무관심하였거나 알고리즘의 시작노드를 중지시킴으로써 해결하였던 것에 반해 본 알고리즘은 효율적이고 상세한 교착상태 해결방법을 제시하였다.

키워드 : 분산시스템, 분산 알고리즘, 교착상태, 교착상태 발견, 교착상태 해결

Abstract Deadlock detection in distributed systems is considered difficult since no single site knows the exact information on the whole system state. This paper proposes a time-efficient algorithm for distributed deadlock detection and resolution. The initiator of the algorithm propagates a deadlock detection message and builds a reduced wait-for graph from the information carried by the replies. The time required for deadlock detection is reduced to half of that of the other algorithms. Moreover, any deadlock reachable from the initiator is detected whereas most previous algorithms only find out whether the initiator is involved in deadlock. This feature accelerates the detection of deadlock. Resolution of the detected deadlock is also simplified and precisely specified, while the current algorithms either present no resolution scheme or simply abort the initiator of the algorithm upon detecting deadlock.

Key words : distributed systems, distributed algorithms, deadlock detection, knot detection, deadlock resolution

1. 서 론

분산시스템에서 프로세스들이 서로간에 리소스에 대한 요청이 허락되길 무한정 기다리는 상태가 발생하면 이를 교착상태라고 일컫는다. 교착상태는 리소스 사용도를 저하시키고 프로세스의 진행을 방해하기 때문에 이의 신속한 발견 및 복구는 매우 중요하다[1]. 분산시스템에서 프로세스들간의 의존 관계는 wait-for graph(대기 그래프, WFG)라고 불리는 방향성 그래프(directed graph)로 모델화할 수 있다[2]. 이 그래프에서 각 노드는 프로세스를 의미하고 간선(edge)은 리소스를 대기 중인 프로세스로부터 그 리소스를 사용중인 프로세스로 이어진다.

분산시스템에서 수행 중인 프로세스들을 대상으로 여러 요청 모델들이 정의되었다[2]. AND 요청 모델에서는 임의의 프로세스가 요청한 모든 리소스들이 허락되어야만 그 프로세스는 계속 진행할 수 있다[3]. 한편, OR 요청 모델에서는 임의의 한 리소스에 대한 요청만 허락되면 프로세스는 수행을 지속한다. 보다 일반적인 (Generalized) 요청 모델에서는 대기중인 프로세스가 수

· 본 연구는 한국과학재단 목적기초연구(2000-130300-002-1)지원으로 수행되었음.
† 정 회 원 : 인천교육대학교 컴퓨터교육과 교수
sjlee@mail.inue.ac.kr
논문접수 : 2022년 5월 7일
심사완료 : 2002년 8월 9일

행을 재개하는 조건을 AND와 OR 연산자 및 요청한 리소스들을 포함하는 함수로 표현한다[4,5,6].

교착상태 발견 알고리즘들은 가정하는 요청 모델에 따라 분류될 수 있으며 교착상태는 전제하는 요청 모델에 따라 다르게 정의된다[2]. AND 요청 모델에서 프로세스는 요청한 모든 리소스가 허락되어야만 진행할 수 있기 때문에 이 모델에서 교착상태는 WFG 내 싸이클(cycle) 존재여부로 알 수 있다. OR 요청 모델에서는 WFG 내에 knot가 존재하면 교착상태를 의미한다. Knot는 WFG의 부분 그래프로서 이에 속한 임의의 노드는 같은 knot에 속한 다른 모든 노드들에게로 경로가 존재한다[7]. 그러므로 이 모델에서 싸이클은 교착상태의 필요조건이지만 충분조건은 아님을 알 수 있다.

본 논문에서 제안한 알고리즘은 OR 요청 모델을 가정한다. 이 모델은 여러 영역에서 그 실례를 찾아볼 수 있다. 복제 데이타베이스 시스템에서 데이타 항목의 읽기 요청은 그 항목의 여러 복사본들 중 하나를 획득하면 만족된다[8]. CSP 언어에서는 프로세스가 alternative 명령을 수행할 때 프로세스 집단으로부터 메시지를 기다린다[9]. OR 요청 모델을 통신네트웍 상에 적용하면 교착상태는 통신 교착상태로 불린다[10]. 이 상태에서는 프로세스가 다른 프로세스와 통신하려 하지만 메시지를 송수신하지 못하고 대기하게 된다.

OR 요청 모델에서 교착상태를 발견하기 위한 알고리즘들의 공통된 특징은 [11]에서 제시한 diffusing computation을 이용하는 것이다. 이 계산 방법에서는 알고리즘의 시작노드가 probe라는 특정 메시지를 전파하고 그에 대한 응답 메시지를 수신한다. 응답메시지에 실린 정보 또는 응답메시지 수를 토대로 하여 시작노드는 교착상태의 존재여부를 결정한다[10,12,13]. 이 방법은 2e 메시지와 2d 시간을 소요하게 되는데 이 때 e는 알고리즘 실행에 관계한 WFG의 간선수이고 d는 WFG의 폭을 나타낸다.

분산시스템에서 교착상태 문제를 다룬 대부분의 연구결과는 그러한 상태를 발견하는데에만 그치고 이의 복구 문제에 대해서는 언급하지 않았다[10,12,13]. 알고리즘의 시작노드가 교착상태를 선언하게 되면 이의 복구를 위해서 교착상태에 속한 노드들 중에서 희생자를 선택해야 한다. 이 때 희생의 댓가를 최소화하는 것이 바람직하다. 그러나 [14]에서는 시작노드를 희생자로 선정함으로써 교착상태를 해결하는 방안을 제시하였다.

교착상태의 신속한 해결은 시스템 성능을 위해 매우 중요하므로 본 연구에서는 기존의 알고리즘[10,12,13]에 비해 거의 두 배로 빨리 교착상태를 발견하는 알고리즘을 제시한다. 본 알고리즘에서는 probe가 전파되어 분산 탐색 트리를 만들고 응답메시지가 시작노드에게 곧바로 송신된다. 시작노드는 응답메시지에 실린 정보를 근거로 국부적인 WFG를 형성한다. 전달되는 정보량을 줄이기 위하여 시작노드로부터 각 프로세스에 이르기까지의 경로가 코드화되며 이로 인해 노드간의 모든 대기 관계를 응답메시지를 통해 전달할 필요가 없게 되며 일부 노드간 대기 관계는 시작노드에서 유추한다. 이는 [12,14]에서 모든 노드간 의존관계 정보를 전달한 것과 대조되는 방법이다. 또한 본 연구의 알고리즘은 [10,12,13,14]에서처럼 단지 시작노드가 교착상태에 속하였는지의 여부만을 판단하는 것이 아니라 시작노드로부터 도달가능한 모든 교착상태를 파악할 수 있다. 이는 보다 정확한 교착상태의 복구 절차를 가능하게 하며 또한 희생자의 수를 최소화하는데 도움을 준다. 이와 같은 장점에도 불구하고 소요되는 메시지나 시간은 타알고리즘에 비해 거의 반에 불과하다. 또한 기존의 알고리즘들이 교착상태의 복구방안을 제시하지 않거나[10,12,13] 단지 시작노드를 중지시킴으로써 해결한데 반하여[14] 본 알고리즘은 보다 정확하고 효율적인 해결책을 제안하였다.

논문의 나머지는 다음과 같이 구성되었다. 다음 절에서는 기존 알고리즘에 대한 구체적 분석 내용을 기술한다. 3절은 본 연구의 알고리즘을 소개하였고 4절에서 이의 정확성을 증명한다. 5절에서 성능 분석을 제시하였고 6절에서는 논문의 결론을 맺는다.

2. 기존 연구

Diffusing computation을 기반으로 한 [13]의 알고리즘에서는 대기중인 프로세스가 교착상태를 의심하게 될 때 알고리즘을 시작하여 probe를 발생시킨다. 프로세스는 자신이 요청한 리소스를 점유하고 있는 프로세스에게 probe를 전달한다. 프로세스가 probe를 받으면 자신이 진행상태일 때는 수신한 probe를 폐기하지만 대기상태일 때는 probe를 처음 수신하는 경우에만 이를 계속 전달한다. 프로세스가 처음 probe를 받게 되면 이 probe의 송신 프로세스를 자신의 연관자라고 부른다. 만약 대기상태의 프로세스가 이전에 이미 수신한 probe를 재수신하게 되면 probe의 송신자에게 곧바로 응답 메시지를 보낸다. 프로세스가 자신이 보낸 probe에 대한 응답 메시지를 모두 받으면 자신의 연관자에게 마침내 응답 메시지를 보내게 된다. 이와 같은 방식으로, 시작노드가 응답 메시지를 모두 받게 되면 교착상태를 선언한다. [10]에서도 이와 유사한 알고리즘을 사용하였는데 다만 각 프로세스를 위한 저장 공간을 축소시키는 대신

주기적으로 신호를 보냄으로써 교착상태를 발견하였다. 두 알고리즘을 수행한 결과로 시작노드는 교착상태에 속하는지의 여부를 알 수 있게 된다. 따라서 WFG의 모든 노드가 그와 같은 사실을 알려면 O(ne) 메시지가 소모된다(e는 알고리즘 실행에 관계한 WFG의 간선수이고 n는 WFG의 노드 수).

[12]에서는 [13]과 유사한 방안을 제시하였으나 보다 많은 정보를 응답메시지를 통해 전달한다. Probe에 대한 응답메시지는 각 프로세스의 상태 정보 뿐만 아니라 대기중의 프로세스가 요청한 리소스 정보를 전달한다. 이 때문에 고정 크기의 메시지를 사용한 기존의 연구결과[10,13]와는 달리 O(nlogn)에 달하는 가변크기의 메시지를 사용하게 된다. 알고리즘 종료시에 시작노드는 자신이 싸이클 또는 knot에 속하는지를 알게 된다. 이 알고리즘 또한 diffusing computation을 근거로 하기 때문에 2e 메시지와 2d 시간을 필요로 한다 (d는 WFG의 폭).

이와 같이 알고리즘을 한번 실행함으로써 시작노드의 상태만을 아는데 그치는 대신 Cidon은 알고리즘에 참여한 모든 노드가 knot에 속하였는지 여부를 알 수 있는 알고리즘을 개발하였다[15]. 이 알고리즘은 싸이클을 발견하고 무리짓게 하며 더 큰 무리에 합병시키는 기술을 사용한다. 이를 위해 사용하는 메시지 수는 13nlogn+3e를 초과하지 않는다. 이는 알고리즘의 복잡성과 속도는 차치하고라도 [10,13]와 비교했을 때 같은 결과를 위해 보다 적은 수의 메시지와 저장공간을 소모하는 것이다.

위에서 언급한 알고리즘들은 교착상태를 발견하기만 하고 이를 복구하는 것에 관심을 기울이지 않았다. 교착상태의 보편적인 복구방법은 적절한 프로세스를 선택하여 그가 점유한 모든 리소스들을 돌려 주고 작업을 재수행하게 하는 것이다. 대부분의 기존 연구가 복구를 위한 희생자 선택에 대해 언급하지 않은 것과는 달리 [14]는 [13]의 알고리즘에 교착상태 복구 방안을 추가하였다. [13]의 알고리즘에서와 같이 시작노드는 모든 응답메시지를 수신하면 교착상태에 속하였음을 알게 되며 이 때 스스로를 재시작함으로써 교착상태를 해결한다. 그러나 OR 요청모델에서 교착상태는 여러개의 싸이클로 구성될 수 있기 때문에 단지 시작노드만 희생함으로써 WFG 내의 모든 교착상태가 해결되리라는 보장이 없다. 따라서 남아 있는 교착상태의 해결을 위해서 또다른 알고리즘의 실행이 요구될 수도 있다. 그러므로 교착상태를 신속히 발견하는 문제 뿐만 아니라 발견된 교착상태를 완벽하고도 최소한의 비용으로 해결하는 것은 매우 중요하다. 시스템 성능 측면으로 볼 때 중지되는 프로세스수를 최소한으로 하는 것이 가장 바람직하나 이는 NP-complete 문제로 알려져 있다[7].

본 연구의 알고리즘에서는 구체적인 교착상태 복구 루틴을 제안하고 그에 따라 선정된 희생자에게 중지 명령을 내린다. 이는 희생자 수를 최적화하지는 않으나 불필요한 희생을 방지하는 알고리즘이다. 한편 [14]에서는 교착상태 복구를 위한 추가적인 메시지를 발생시키지 않는데 이는 교착상태를 발견한 시작노드가 스스로 재시작하기 때문이다.

분산적 알고리즘에서는 임의의 노드가 교착상태를 의심하면(즉, 일정한 시간(time-out) 동안 대기상태에 있게 되면) 알고리즘을 시작하게 된다. 따라서 시스템 내에 동시에 여러 알고리즘 작업이 진행될 수 있다. 시스템에 과부하를 줄이기 위해서 알고리즘 수행을 줄이는 것이 바람직하다. 이를 만족시키면서 한편으로는 교착상태를 신속히 발견 및 복구하는 것이 중요하다. [14]에 따르면 동시 진행되는 알고리즘 작업이 다수일 때 낮은 우선순위의 작업이 높은 우선순위의 작업으로 하여금 새롭게 또다시 알고리즘을 수행하도록 한다. 이 때문에 n개의 노드들로 구성된 교착상태를 발견하는데 $O(n^2)$의 알고리즘이 수행될 가능성이 있다. 반면에 본 연구의 알고리즘의 실행 횟수는 O(n)을 넘지 않는다.

본 연구의 알고리즘에서 사용하는 메시지 길이의 complexity는 5.1에서 증명되듯이$O(d\log_2 n)$이며 최적의 경우 $O(\log_2 n)$이다. 한편 [12,14]의 알고리즘에서는 메시지가 모든 노드 식별자를 운반할 가능성이 있으므로 메시지 길이는 $O(n\log_2 n)$이다. 대표적 알고리즘들의 성능 비교는 표 1에 요약하였다.

표 1 교착상태 발견을 위한 알고리즘의 성능 비교(e는 WFG의 간선수. d는 WFG의 폭. n은 WFG 노드 수)

	Chandy [13]	Boukerche [12]	Villadangos [14]	제안된 알고리즘
메시지 수	2e	2e	2e	<2e
소요시간	2d	2d	d	d
메시지 크기	고정	O(nlog2n)	O(nlog2n)	O(dlog2n)
교착상태 해결방안	제시 안함	제시 안함	시작노드가 희생자	불필요한 희생을 방지

3. 교착상태 발견 및 복구 알고리즘

3.1 시스템 모델

분산시스템은 컴퓨터 네트웍으로 연결된 사이트들로 이루어진다. 시스템은 공통 메모리를 갖지 않으며 임의의 두 사이트간에 메시지를 송수신함으로써 통신한다. 본 논문에서는 메시지 전송시간을 예측할 수 없는 비동

기 네트워을 가정한다. 그러나 메시지는 손실되지 않고 오류없이 목적지에 도달한다. 시스템 내의 메시지는 계산(computation) 메시지와 제어(control) 메시지로 구분한다. 계산 메시지는 프로세스의 수행에 필요한 REQUEST, REPLY 및 CANCEL 메시지이며 제어 메시지는 교착상태 발견을 위한 알고리즘이 발생시킨다.

프로세스의 상태는 대기(blocked) 또는 진행(active) 상태 중 하나이다. 프로세스가 리소스 요청을 위해 REQUEST 메시지를 발생시킬 때 대기상태로 돌입한다. REPLY 메시지는 요청이 허락되었음을 의미한다. 일단 프로세스가 대기상태가 되면 더 이상의 REQUEST 메시지를 보내지 못한다. 대기상태의 프로세스가 하나 이상의 REPLY를 수신하면 진행상태로 전환한다. 이 때 자신이 요청한 다른 리소스에 대해서 CANCEL 메시지를 보냄으로써 취소한다. Active 상태의 프로세스는 계산 메시지와 제어 메시지를 모두 보낼 수 있으나 대기상태의 프로세스는 제어 메시지만을 보낼 수 있다.

프로세스 p가 프로세스 q에게 REQUEST 메시지를 보내면 WFG 상에서 간선 (p,q)가 형성되며 q는 p의 계승자(successor)라고 부른다. 이 간선은 q가 p에게 REPLY 메시지를 보내거나 p가 q에게 CANCEL 메시지를 보내면 사라진다. 만약 WFG 상에서 p에서부터 q로 경로가 존재하면 q는 p로부터 도달가능하다라고 말하며 $(p,q)^* \in WFG$로 표기한다. 본 논문에서는 프로세스와 노드의 용어를 같은 의미로 사용하였다. 임의의 프로세스가 knot에 속하면 이는 교착상태에 돌입한 것이다. Knot란 노드들의 집합인데 이 집합에 속하는 임의의 노드에서 도달가능한 노드들이 바로 같은 집합을 이루는 특성을 가졌다. Knot의 정의는 아래와 같다.

[정의 1] Knot K는 WFG의 부분 그래프로써 다음과 같은 성질을 만족한다:

(1) K에 속한 임의의 두 노드 p와 q에 대해, $(p,q)^* \in$ K이다.

(2) K에 속한 임의의 노드 p에 대해, 만약 $(p,q)^* \in$ WFG라면 $q \in K$이다.

동적인 환경에서는 알고리즘이 수행되는 동안 WFG 구성이 변경될 수 있다. 따라서 알고리즘이 교착상태의 부재를 선언할 때 마침 시스템 내에 교착상태가 발생하였을 수 있다. 그러나 알고리즘의 분산적 성질에 의거하여 이 교착상태는 추후의 알고리즘 실행에 의해 발견된다. 그러므로 정확한 교착상태 발견 알고리즘은 다음의 두 성질만을 보장하게 된다; (i) 알고리즘 시작시에 교착상태가 존재하면 알고리즘은 이를 발견하며 (ii) 알고리즘이 교착상태를 선언하면 알고리즘 종료시에 실제로 교착상태가 존재한다.

3.2 교착상태 발견 알고리즘

알고리즘은 probe 전달과정과 응답 메시지 수신과정의 두 단계로 구성되는데 이 두 단계는 동시 진행될 수도 있다. 첫째 단계에서 알고리즘의 시작노드는 probe를 전파함으로써 distributed spanning tree(분산 신장 트리, DST)를 형성한다. 두번째 단계에서 probe의 응답메시지가 트리의 근노드(root)인 시작노드에게 직접 전달된다. 응답메시지에 실린 정보를 토대로 시작노드는 교착상태 여부를 결정하기 위하여 국부적으로 WFG를 구축한다. 본 알고리즘은 기존의 알고리즘들[12,14]과는 달리 WFG상의 모든 간선 정보를 보고하지 않는다. 대신 정보량을 축소시키기 위한 전략으로 path string이라는 개념을 개발하였다. 이제 알고리즘에 대한 구체적인 설명은 다음와 같다.

알고리즘의 시작노드는 자신의 모든 계승자들에게 probe를 송신한다. 노드가 만약 처음으로 probe를 받게 되면, DST 상에서 probe 송신자의 자노드(child)가 된다. 그런 다음 probe는 진행상태의 노드나 이미 트리의 구성원이 된 노드를 만날 때까지 계속 전달된다. 이 때 응답메시지가 시작노드에게 직접 전달된다. 따라서 응답메시지는 진행상태의 노드가 보내거나 non-tree 간선이 발견되었을 때 보내진다. 응답 메시지에 대해서는 본 절에서 후에 보다 자세히 기술하였다.

DST 상에서 임의의 두 노드간의 조상-자손 관계를 쉽게 인식하기 위하여 각 노드는 path string이라 부르는 유일한 이진 string을 부여받는다. 이는 근노드로부터 각 노드에 이르기까지의 경로를 암호화한 것으로서 DST상에서 분기(branch)간의 구별을 용이하게 한다. 노드의 path string은 그 노드의 부노드(parent)에 의하여 probe를 통하여 전달된다. Path string에 관한 연산 및 정의는 다음과 같다.

[정의 2] 1. 임의의 두 path string α와 β의 concatenation은 $\alpha \| \beta$로서 표기하며 이는 α에 β를 덧붙여 새로운 path string을 만드는 연산이다.

2. λ는 그 길이, 즉 $|\lambda|$가 0인 path string이며 임의의 path string α에 대해 $\alpha \| \lambda = \lambda \| \alpha = \alpha$이다.

3. α, β, Υ를 임의의 path string이라 하자. 만약 $\beta = \alpha \| \Upsilon$이고 $\Upsilon \neq \lambda$이라면 α는 β의 prefix라고 하며 $\alpha <_p \beta$로 표기한다.

DST 상에서 조상이나 자손 관계가 아닌 노드들을 제노드(*sibling*)라고 하자. *Path string*을 부여하는 데 있어서 목적은 노드의 path string이 그 노드 자손들의 path string의 prefix가 되게 하고 제노드들의 path

string 간에는 prefix 관계가 없도록 하는 것이다. 그렇다면 임의의 두 노드의 path string들로부터 그 노드들이 조상-자손 관계에 있는지를 쉽게 알 수 있다. 이 목적을 달성하기 위하여 시작노드의 path string은 λ로 정하고 각 계승자 i를 위하여 이진 string α_i를 각기 다르게 형성한 후 λ||α_i를 probe를 통하여 i에게 전달한다. Probe를 처음 받으면, 노드는 전달받은 path string을 자신의 것으로 저장하고 계승자들을 위하여 시작노드에서와 마찬가지 방식으로 path string을 형성한 후 전달한다. 노드의 계승자 수가 m(>1)이라면 각기 다른 path string을 부여하기 위하여 노드는 $\lceil \log_2 m \rceil$ 비트를 사용해야 함을 쉽게 알 수 있다.

알고리즘이 사용하는 응답 메시지는 ACTIVE와 REPORT의 두 종류가 있다. REPORT는 non-tree 간선 정보를 시작노드에게 전달한다. 이 정보는 각 non-tree 간선의 양 노드 식별자와 path string을 포함한다. Active 노드는 자신의 식별자와 path string을 ACTIVE 메시지를 통하여 시작노드에게 전달한다. 이러한 정보를 토대로 시작노드는 교착상태 존재 여부를 결정하기 위하여 WFG를 구축한다. WFG의 모든 노드들이 시작노드에게 응답 메시지를 보내지는 않으므로 시작노드의 WFG를 축소된 WFG(RWFG)라고 부르기로 한다.

RWFG는 다음과 같이 형성된다. 응답메시지가 전달한 모든 노드 및 간선 정보는 그대로 RWFG에 포함된다. 시작노드가 응답 메시지를 모두 받으면, RWFG 내의 임의의 두 노드간에 조상-자손 관계를 추론하여 간선을 그에 따라 삽입한다. 이를 위하여 Knuth-Morris-Pratt의 문자열 조회 알고리즘을 사용할 수 있다 [7]. 임의의 노드 w에 대하여 가장 젊은 부노드 v를 찾아 RWFG 내에 (v,w)를 포함시킨다. Path string을 지정하는 방법에 의하면 노드 w의 가장 젊은 부노드는 RWFG 내 노드들 중에서 w의 path string의 가장 큰 prefix를 갖게 된다. 이에 대한 구체적 정의는 아래 제시하였다.

[정의 3] 노드 i의 path string을 $pstr_i$라고 표기하자. RWFG의 임의의 두 path string $pstr_v$과 $pstr_w$에 대해 다음의 조건을 모두 만족하면 $pstr_v$을 $pstr_w$의 largest prefix라고 하며 $pstr_v <_{lp} pstr_w$로 표기한다. 또한 노드 v를 노드 w의 closest ancestor라고 부른다.

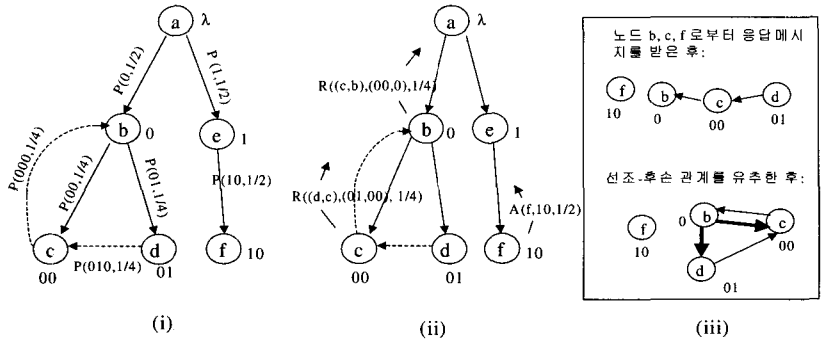
(1) $pstr_v <_p pstr_w$

(2) RWFG 내 임의의 path string Υ에 대해 $\Upsilon <_p pstr_w$라면 $\Upsilon <_p pstr_v$이다.

시작노드에서 가장 큰 prefix를 찾는 방법 중 하나는 RWFG 내의 path string을 노드로 하는 이진 트리를 형성하는 것이다. 예를 들어, λ를 근노드로 하고 path string "0"와 "1"을 각각 좌측 자노드와 우측 자노드로 형성한다. 마찬가지로 "00", "01"을 "0"의 좌측과 우측 자노드로 한다. 그러면 이와 같이 형성된 이진 트리에서 임의의 노드의 부노드가 그 노드의 closest ancestor임을 쉽게 알 수 있다. 이진 트리의 구성에 소요되는 시간은 O(*n*log*n*)이고 각 노드의 *closest ancestor*를 찾는 데는 O(n)의 시간이 소요된다.

시작노드에서 응답메시지의 수신완료에 대한 인식은 중량(weight)를 분산하는 기술을 이용한다[10,16]; 시작노드는 노드들에게 1의 중량을 분산시키고 응답메시지를 통하여 중량을 수집하며 수집된 중량의 합이 1이 되면 모든 응답메시지를 수신한 것으로 판단한다. 보다 상세히 설명하면, 만약 시작노드가 n개의 계승자를 가졌으면 각 계승자에게 1/n의 중량을 probe를 통하여 송신한다. 이와 마찬가지로 임의의 노드가 probe를 통하여 w의 중량을 받으면 자신의 계승자들에게 probe를 통하여 각기 w/n의 중량을 나눠준다(n개의 계승자를 가정). Weight가 w인 probe를 받으면 진행상태의 노드는 같은 중량을 ACTIVE 메시지를 통해 시작노드에게 전달한다. 만약 그 probe가 non-tree 간선을 통하여 전달되었다면 중량 w를 REPORT 메시지를 통해 시작노드에게 전달한다.

제안한 알고리즘의 수행과정은 그림 1에 예시되었다. 간편성을 위하여 메시지 이름은 그 첫글자로 생략되었다. Probe를 의미하는 P 메시지는 계승자를 위한 path string과 중량만을 전달하는 것으로 나타냈다. 각 노드의 path string은 원으로 둘러싸인 노드 식별자 옆에 표시되었다. 그림 (i)에서 path string λ를 가진 노드 a가 알고리즘을 시작한다. 노드 a는 두 계승자들을 가졌기 때문에 path string "0"와 "1"을 각 계승자에게 보낸다. 노드 b가 probe를 받으면 노드 a의 자노드가 되고 전달받은 "0"를 자신의 path string으로 기록한다. 또한 자신의 계승자들을 위해 $pstr_b$||"0"와 $pstr_b$||"1", 즉 "00"와 "01"을 생성하여 probe를 통해 각 계승자에게 보낸다. 한편 노드 f가 probe를 받으면 "10"을 자신의 path string으로 저장하며 진행 상태이기 때문에 더이상의 probe를 송신하지 않는다. 노드 c와 d는 하나의 계승자만을 가졌기 때문에 자신의 path string에 "0" 를 첨가해 계승자에게 보낸다. 이는 path string의 유일성을 보존하기 위함이다. 이 예제에서 보는 바와 같이 임의의 노드의 path string은 후손의 path string의 prefix가 됨을 알 수 있다. 그림 (ii)는 응답 메시지의 전달 과정을 예시한다. Tree 간선은 실선으로, non-tree 간선은



(i) (ii) (iii)

그림 1 (i) a가 시작노드인 알고리즘에서 probe를 송신하는 과정 (ii) 응답메시지의 전달 과정 (iii) 시작노드에서 구축하는 RWFG

점선으로 표시하였다. 응답 메시지 ACTIVE의 첫째 인자는 노드 식별자, 둘째 인자는 그 노드의 path string 이며 마지막 인자는 중량이다. 노드 f는 노드 e로부터 probe를 받으면 ACTIVE 메시지를 시작노드에게 보냄으로써 자신이 진행상태임을 알린다. 노드 b가 노드 c로부터 probe를 받으면, b는 과거에 이미 probe를 받아서 알고리즘의 수행에 참여하고 있기 때문에 probe가 전달된 간선 (c,b)를 non-tree로 인식하고 REPORT 메시지를 시작노드에게 보낸다. 이 때 간선 (c,b) 및 노드 c와 b의 path string, 중량 등을 전달한다. 이와 마찬가지로 노드 c 또한 d로부터 probe를 받으면 간선 (d,c)에 대한 정보를 REPORT를 통해 시작노드에게 보낸다. 조상-자손 관계는 시작노드가 path string으로부터 유추할 수 있지만 non-tree 간선은 알 수 없기 때문에 그림에서와 같이 WFG 내의 모든 non-tree 간선은 송신된다. 시작노드가 전달받은 중량의 합은 1이 됨을 알 수 있다. 그림 (iii)에서는 시작노드가 구축하는 RWFG 형태를 보여준다. 그림 (iii)의 하단 그래프에서 노드 b가 노드 c의 closest ancestor이기 때문에 (b,c)가 삽입되었다. 마찬가지로 간선 (b,d)가 포함되었다. 그림에서 조상-자손 관계로 추론된 간선은 굵게 표시되었다. 구체적인 알고리즘은 다음과 같다.

노드 i를 위한 자료 구조 (초기값은 괄호 안에 있음)

- out_i: 노드 i의 계승자들의 집합
- $pstr_i$: 노드 i의 path string. (λ)
- $parent_i$: 노드 i에게 probe를 처음 전달한 노드. (0)
- $weight_i$: 노드 i의 중량 값. (0)

메시지 형식 (w 인자는 중량 값을 의미)

- PROBE(init, succ_pstr, $pstr_j$, j, w): path string이 $pstr_j$인 노드 j가 보내는 probe 메시지. succ_pstr은 이 메시지의 수신자를 위해 생성된 path string이며 init은 알고리즘의 시작노드를 나타낸다.
- ACTIVE(i, $pstr_i$, w): path string이 $pstr_i$인 노드 i가 진행상태임을 알리는 probe에 대한 응답 메시지.
- REPORT((j,i), ($pstr_j$,$pstr_i$), w): non-tree 간선 (j,i)의 발생을 알리는 응답 메시지.

(I) 노드 i가 알고리즘을 시작할 때:
$pstr_i$:= λ;
proc_propagate(i, 1);

(II) 노드 i가 PROBE(init, succ_pstr, $pstr_j$, j, w)를 수신할 때:

(II.1) **if** $parent_i$=0 **and** i≠init **then begin**
/* tree 간선 */
$parent_i$:= j;
$pstr_i$:= $pstr_j$ || succ_pstr;
$weight_i$:= w;
if out_i =∅ **or** i has sent a REPLY message to j **then**
send ACTIVE(i, $pstr_i$, w) to init;
else if i is blocked **then** proc_propagate (init, w);
end if

(II.2) **else** send REPORT((j,i), ($pstr_j$,$pstr_i$), w) to init; /* non-tree 간선 */

(III) 시작노드 init이 ACTIVE(j, $pstr_j$, w)를 수신할 때:
RWFG := RWFG ∪{j};
proc_collect_replies(init, w);

(IV) 시작노드 init이 REPORT((j,i), ($pstr_j$,$pstr_i$), w)를 수신할 때:
RWFG := RWFG ∪ {i, j};

```
    RWFG := RWFG ∪ {(j,i)};
    proc_collect_replies(init, w);
(V) procedure proc_collect_replies(init, w)
    weight_init := weight_init + w;
    if weight_init = 1 then begin  /* 모든 응답
                메시지 수신함. */
        for each x∈RWFG  begin
            find the closest ancestor y∈RWFG
            of x;
            if (y,x)*∉RWFG then  RWFG :=
              RWFG∪{(y,x)};
        end for
        resolution_rtn();    /* 2.3절에서 제시 */
    end if
end procedure
(VI) procedure proc_propagate(init, w)  /* 노드
    i에서 수행 */
    if |out_i| > 1  then begin
        succ_pstr := ⌈ log2|out_i|⌉  number of 0's;
        for each  successor k∈out_i  begin
                send PROBE(init, succ_pstr,
                pstr_i, i, w/|out_i|) to k;
                succ_pstr:=succ_pstr[+]1;1)
        end for
    end if
    else if |out_i|=1 then  send PROBE(init, ''0'',
    pstr_i, i,w) to the successor;
end procedure
```

3.3 교착상태의 해결

RWFG 구축이 완료되면 시작노드는 RWFG 상에서 교착상태 발견을 위한 절차를 시작한다. 따라서 보다 간단하고 효율적으로 해결하기 위하여 다음과 같은 방식을 제안한다. 우선 RWFG에서 계승자를 가진 노드는 unfree 상태로, 그렇지 않은 노드는 free 상태로 초기화한다. 그런 다음 unfree 상태의 노드가 free인 노드에게로 경로가 존재하는지를 점검한다. 만약 그렇다면 그 노드의 상태를 free로 변경한다. 이 과정을 노드의 상태 변화가 더이상 일어나지 않을 때까지 반복적으로 수행한다. 결국에는 knot에 속하거나 모든 계승자가 knot으로 귀착되는 노드는 unfree 상태로 남게 된다. 모든 노드의 상태가 이와 같이 결정된 후 unfree 상태의 노드 중 희생자가 결정된다. 그런 다음 희생자가 점유한 모든 리소스들을 해지하는 것을 시뮬레이션한다; 즉 희생자의 상태는 free로 변경하고 희생자에게 경로가 있는 모든 노드들의 상태도 마찬가지로 free로 변경한다. 이 루틴은 더이상 unfree 상태의 노드가 RWFG 내에 존재하지 않을 때까지 반복된다. 이 루틴의 종료시에 시작노드는 선택된 희생자(들)에게 ABORT 메시지를 송신한다. 다음은 이상과 같은 교착상태 해결을 위한 알고리즘이다.

RWFG의 노드 i를 위한 자료 구조

- $succ_i$: 노드 i의 RWFG 내 계승자들의 집합
- $state_i$: 노드 i가 교착상태인지를 나타내는 변수.

```
procedure resolution_rtn()
    for each i∈RWFG
        if succ_i=∅  then  state_i:=free;
        else  state_i:=unfree;
L1: if ∃i∈RWFG with state_i=unfree  then
finished := false;
    else  finished := true;
    while not  finished  begin
        finished := true;
        for each  i∈RWFG with state_i=unfree
            if  ∃j∈succ_i such that state_j=free
            then begin
                state_i := free;
                finished := false;
            end if
    end while
    if  ∃i∈RWFG with state_i=unfree  then begin
        select a victim v according to the
        predefined policy;
        state_v := free;
        go to L1;
    end if
end procedure
```

4. 알고리즘의 정확성

알고리즘의 정확성은 우선 시작노드에서 구축되는 RWFG에 의해 좌우된다. 그러므로 WFG 내에 교착상태의 존재는 곧 RWFG 내에서도 존재하고 그 반대의 경우도 성립함을 증명하고자 한다. 증명을 위해 필요한 용어의 정의를 아래에 기술하였다.

SC_i: DST 내 노드 i의 자노드들의 집합

SST_i: DST 내 노드 i를 근노드로 하는 부분 트리

$PROBE_{i\to j}$: 노드 i가 노드 j에게 보낸 PROBE

1) [+] 연산은 스트링을 이진수로 취급하여 더한다. 예를 들어, succ_pstr="00"이라면 연산 결과 새로운 succ_pstr 값은 "00"[+]"01", 즉, "01"이 된다.

PROBE.w: PROBE가 전달하는 중량 값

노드 i의 중량은 두 개의 항목의 합으로 이루어지는데 첫째 항목은 노드 i의 자노드들의 중량 값이고 둘째 항목은 i를 시작점으로 하는 non-tree 간선을 통해 전달되는 PROBE에 실린 중량 값이다. 다음 lemma는 이를 증명하였다.

Lemma 1. $weight_i = \sum_{k \in SC_i} weight_k + \sum_{PROBE_{i \to j} \ \& \ j \notin SC_i} PROBE.w$.

증명. 알고리즘의 단계 (VI)에 의하면, 노드 i는 그의 각 계승자에게 $1/|out_i|$의 중량을 PROBE를 통해 보낸다. 집합 out_i는 DST 내 노드 i의 자노드들의 집합과 non-tree 간선 (i,j)의 노드 j(들)을 포함한다. 단계 (II.1)에서 SC_i의 각 노드 k는 $1/|out_i|$를 자신의 중량 값으로 저장한다. 따라서 lemma 는 성립한다. ■

다음 lemma는 노드 i의 중량이 두 항목의 합과 같음을 말해 준다. 첫째 항목은 SST_i 내 진행상태 노드들의 중량 합이며 둘째 항목은 SST_i 내 노드들을 시작점으로 하는 non-tree 간선을 통해 전달되는 PROBE.w 합이다.

Lemma 2. $N=\{n \mid n \in SST_i \ \& \ n$은 진행상태$\}$, $M=\{PROBE_{p \to q} \mid p \in SST_i \ \& \ q \notin SC_p\}$이라 하면

$weight_i = \sum_{n \in N} weight_n + \sum_{PROBE \in M} PROBE.w$.

증명. $A_i = \sum_{k \in SC_i} weight_k$이고 $B_i = \sum_{PROBE_{i \to j} \ \& \ j \notin SC_i} PROBE.w$라 하자. Lemma 1에 의해 $weight_i = A_i + B_i$이다. Lemma 1을 i의 각 자노드 k에 적용하면 $weight_i = \sum_{k \in SC_i}(A_k + B_k) + B_i$이 된다. 또한 Lemma 1을 k의 각 자노드에 적용할 수 있으며 귀납적으로 Lemma 1은 더이상 자노드가 없을 때까지 i의 모든 후손에 적용된다. 즉, leaf(단노드)를 만날 때까지 계속 적용한다. 그러므로

$$weight_i = \sum_{n \in N} weight_n + B_i + \sum_{k1 \in SC_i}(B_{k1} + \sum_{k2 \in SC_{k1}}(B_{k2} + \sum_{k3 \in SC_{k2}}(B_{k3} + \ldots = \sum_{n \in N} weight_n + \sum_{PROBE \in M} PROBE.w. \quad \blacksquare$$

Lemma 3. 만약 p가 DST 내의 단노드라면 $p \in RWFG$이다.

증명. 노드 p는 알고리즘 수행에 참여할 시에 진행상태이거나 대기상태이다. 만약 p가 진행상태라면 단계 (II.1)에 의하여 시작노드에게 ACTIVE 메시지를 보낸다. 따라서 단계 (III)에 의하여 $p \in RWFG$이다. 이와 반대로 p가 대기상태라면, p는 단노드이기 때문에 그로부터 시작되는 모든 간선은 non-tree 타입이다. (p,q)를 이 non-tree 간선들 중 하나라고 하자. (p,q)를 통해 PROBE를 수신할 때 노드 q는 단계 (II.2)에 의하여 (p,q) 정보를 REPORT 메시지를 통해 시작노드에게 전달한다. 이 REPORT를 받으면 시작노드는 단계 (IV)에 의해 RWFG 내에 p와 q를 삽입시킨다. 그러므로 $p \in RWFG$이다. ■

Lemma 4. p, $q \in DST$이고 또한 p, $q \in RWFG$라고 가정하자. 그러면 $(p, q)^* \in DST$는 $(p, q)^* \in RWFG$의 필요충분조건이다.

증명. 우선 $(p, q)^* \in DST$를 가정하자. 또한 p에서 q로의 경로가 tree 간선으로만 구성되었다고 가정하자. 그러면 $pstr_p <_p pstr_q$이다. 만약 $pstr_p <_{lp} pstr_q$라면 단계 (V)에 의해 $(p, q) \in RWFG$이다. 그렇지 않다면 $pstr_r <_{lp} pstr_q$를 만족하는 RWFG 내 노드 r이 존재한다. 그러므로 $(r, q) \in RWFG$이다. 위와 같은 논리는 p와 r에 회귀적으로 적용된다. 따라서 $(p, q)^* \in RWFG$이다. 이제 p에서 q로의 경로 내에 non-tree 간선인 (v,w)가 존재함을 가정하자. 알고리즘의 단계 (II.2)와 (IV)에 따르면 $(v,w) \in RWFG$이다. $pstr_p <_p pstr_v$이고 p, $v \in RWFG$이기 때문에 위 tree 간선 경우에 의하여 $(p, v)^*$는 RWFG에 존재한다. 이와 마찬가지로 $(w, q)^* \in RWFG$이다. 그러므로 $(p, q)^* \in RWFG$이다. p에서 q로의 경로 내에 non-tree 간선 수에 대하여 귀납법을 적용하면 $(p, q)^* \in RWFG$이다. 이제 Lemma의 증명을 위해 $(p, q)^* \in RWFG$를 가정한 경우는 위와 유사하게 증명되며 따라서 생략한다. ■

Theorem 1. 알고리즘은 한정된 시간 내에 종료한다.

증명. 알고리즘의 단계 (V)에 따르면 시작노드는 응답메시지에 실린 중량의 합이 1이 되면 RWFG 상에서 교착상태를 찾기 시작한다. Lemma 2로부터 시작노드의 중량은 DST의 진행상태 노드들의 중량 합과 non-tree 간선을 통하여 전달된 PROBE가 운반한 중량 합을 더한 것임을 알 수 있다. 진행상태 노드들의 중량은 단계 (II.1)에 의해서 ACTIVE 메시지를 통하여 전달된다. 한편 non-tree 간선을 통하여 전달된 중량은 단계 (II.2)에 의하면 REPORT 메시지를 통하여 시작노드에게 전달된다. 이 메시지들은 네트워 가정에 따라 중도 손실 없이 목적지까지 전달된다. 그러므로 Lemma 2에 의하여 메시지를 통하여 시작노드에게 전달된 중량 합은 1과 같다. 이제 교착상태 해결 방식을 살펴 보면, unfree 상태의 노드들만이 희생자 후보가 된다. 일단 노드가 희생자로 선정되면 그 노드의 상태는 free로 변경된다. 그러므로 선택되는 희생자의 수는 유한하다. ■

Theorem 2. 임의의 RWFG 내 노드 p에 대하여 만약 p가 RWFG 내의 knot K_R에 속하면 p는 또한 DST 내에서 knot에 속한다.

증명. K_R'을 집합 $\{p \in DST \mid p \in K_R$ in the RWFG$\}$과 그에 연관된 간선들이라고 정의하자. 또한 K_R''을

$K_R' \cup \{v \in DST \mid v \notin RWFG, \exists p, q \in K_R$ such that $pstr_p <_p pstr_v <_p pstr_q\}$와 그와 연관된 간선들이라고 하자. 정리를 증명하기 위하여 K_R''이 DST 내에서 knot임을 보여야 한다. 그 반대를 가정하자. 즉, K_R''에 속한 임의의 노드 p에 대해

(i) $(p,q) \in DST$와 $q \notin K_R''$을 만족하는 q가 DST에 존재하거나

(ii) $(p,q)^* \notin K_R''$를 만족하는 K_R'' 내의 q가 존재한다.

먼저 (i)의 경우를 생각하자. (p,q)가 non-tree 간선라고 가정하자. 그러면 단계 (II.2)에 의하여 $(p,q) \in RWFG$이다. $p \in K_R''$이고 $p \in RWFG$이므로 $p \in K_R'$이다. 따라서 K_R'의 정의로부터 $p \in K_R$이다. 또한, $q \notin K_R''$이기 때문에 K_R''의 정의에 의해 $q \notin K_R$이다. 이는 K_R이 knot이라는 사실에 위배된다. 이제 (p,q)가 DST 내의 tree 간선라고 가정하자. 증명은 세가지 경우로 나뉘어진다: (a) q가 단노드인 경우 (b) q가 non-tree 간선의 끝단 노드인 경우 (c) 앞의 두 경우에 해당되지 않는 경우. 우선 (a)의 경우를 고려하자. $p \in K_R'$이라고 가정하자. q는 단노드기 때문에 Lemma 3에 의해 $q \in RWFG$이다. 또한 $p \in K_R'$이기 때문에 K_R'의 정의에 의해서 $p \in K_R$이다. 더우기 p, $q \in RWFG$이고 $(p,q) \in DST$이므로 Lemma 4에 의해 $(p,q)^* \in RWFG$이다. 그러나 K_R''의 정의에 의해 $q \notin K_R''$이므로 $q \notin K_R$이다. 이는 K_R이 knot이라는 가정에 위배된다. 그 이유는 $p \in K_R$, $q \notin K_R$이고 $(p, q)^* \in RWFG$이기 때문이다. 이제 $p \in K_R'' - K_R'$를 가정하자. 즉, $p \notin K_R$이다. K_R'' 정의에 의하여 $pstr_r <_p pstr_p$를 만족하는 노드 r이 K_R'에 존재한다. 즉, $r \in K_R$이다. (p,q)는 tree 간선이기 때문에 단계 (V)에 의해서 $pstr_r <_p pstr_p <_p pstr_q$가 성립한다. 그러므로, $(r, q)^* \in RWFG$이다. 그러나 $q \notin K_R''$이기 때문에 $q \notin K_R$이다. 이는 K_R이 knot이라는 사실에 위배된다. 이제 (b)의 경우를 가정하자. 노드 q는 non-tree 간선과 연결되어 있으므로 단계 (II.2)에 의해서 $q \in RWFG$이다. (a) 경우의 증명과 마찬가지로 $p \in K_R'$이거나 $p \in K_R'' - K_R'$이다. 그러면 K_R 내 노드 p 혹은 r이 RWFG 상에서 q에게로 경로가 있다. 그러나 $q \notin K_R''$이므로 $q \notin K_R$이며 이는 K_R이 knot임에 위배된다. 이제 (c)의 경우를 가정하자. DST에서 q의 후손인 노드 k가 존재해서 k는 단노드거나 non-tree 간선과 연결되어 있다. 따라서 (c) 경우의 증명은 (a)와 (b) 경우의 증명과 마찬가지이므로 생략한다. 이제 증명의 첫부분에서 언급한 (ii)의 경우에 대해 생각하자. 우선 p, $q \in K_R'$라고 가정하자. Lemma 4와 (ii) 경우에 설정한 가정에 의해서 $(p, q)^* \notin RWFG$이다. 그러나 이는 p, $q \in K_R$이고 K_R이 knot이므로 사실과 다르다. 이제 $p \in K_R'' - K_R'$이거나 $q \in K_R'' - K_R'$임을 가정하자. 만약 $p \in K_R'' - K_R'$라면 K_R''의 정의에 의해서 $pstr_p <_p pstr_r$를 만족하는 노드 r이 K_R'에 존재한다. $q \in K_R'' - K_R'$이라면 $pstr_s <_p pstr_q$를 만족하는 노드 s가 K_R'에 존재한다. 그러므로 $(p,r)^* \in DST$ 또는 $(s,q)^* \in DST$이다. $(r,s)^* \in K_R'$에 대한 증명은 p, $q \in K_R'$ 경우와 유사하다(p를 r로, q를 s로 대치). 그러므로 $(p,q)^* \in K_R''$이다. ■

Theorem 3. DST 내에 knot K가 존재하면 RWFG에 knot K_R이 존재하며 이 때 K_R의 노드 집합은 $\{p \in RWFG \mid p \in K\}$ 이다.

증명. K는 적어도 한 싸이클을 포함하므로 적어도 한 non-tree 간선 (v,w)가 존재한다. 단계 (II.2)에 의해 v, $w \in RWFG$임을 알 수 있다. v, $w \in K$이므로 v, $w \in K_R$이다. 그러므로 $K_R \neq \emptyset$. 정리를 증명하기 위해 그 반대의 경우를 가정하자. 즉, K_R 내 노드 p가 존재해서

(i) $\exists q \notin K_R$ where $(p, q) \in RWFG$를 만족하거나

(ii) $\exists q \in K_R$ where $(p, q)^* \notin K_R$를 만족한다.

우선 (i)의 경우를 생각하자. p, $q \in RWFG$이므로 p, $q \in DST$이다. Lemma 4에 의하여 $(p, q) \in RWFG$이기 때문에 $(p,q)^* \in DST$이다. 게다가 $p \in K_R$이므로 $p \in K$이다. 이러한 사실들과 K가 knot이라는 가정에 의해 $q \in K$임이 성립한다. 또한 $q \in K$이고 $q \in RWFG$이므로 $q \in K_R$이다. 이는 $q \notin K_R$이라는 (i)의 가정에 위배된다. 이제 (ii)의 경우를 고려하자. 가정과 Lemma 4에 의해 $(p,q)^* \notin WFG$임이 성립한다. 그러나 p, $q \in K_R$이므로 p, $q \in K$이다. 이는 K가 knot이라는 사실에 위배된다. ■

Corollary 1. RWFG 내에 knot이 존재함은 DST에 knot이 존재하는 것의 필요충분조건이다.

증명. 정리 2와 3에 의해 성립된다. ■

이제까지 정적인 WFG를 가정하여 RWFG 구축의 정확성을 증명하였다. 그러나 WFG는 역동적으로 변화할 수 있으며 이를 고려하여 증명이 이루어져야 한다. 다음 정리는 이 점을 고려하였다.

Theorem 4. DST 내에 knot K가 존재하는 것은 알고리즘 시작시간에 WFG 내에 K가 존재함의 필요충분조건이다.

증명. 우선 WFG 내에 K가 존재함을 가정하자. Knot은 변화하지 않는 성질을 지녔다. 즉, 그에 포함된 노드나 간선이 자발적으로 사라지지 않는다. 그러므로 단계 (II.1)에 의해 PROBE 메시지는 K의 모든 간선들을 통해 전달된다. 이는 $K \subset DST$임을 의미한다. 이제 DST 내에 K가 존재함을 가정하고 정리의 증명을 위해 결론의 반대를 가정하자. 즉, $K \not\subset WFG$. e=(p,q)를 K에서 최

초로 사라지는 간선이라고 하자. K에서 간선이 사라지는 경우를 살펴보면 첫째, q가 p에게 REPLY를 보낼 때와 둘째는 p가 CANCEL 메시지를 q에게 보낼 때이다. 우선 첫째 경우를 고려하자. q가 p에게 REPLY를 보내는 시각을 t1이라 하자. 또한 p가 q에게 PROBE를 보내는 시각을 t2라고 하자. q가 p로부터 PROBE를 처음 받을 때 q∈K이기 때문에 q의 계승자 집합은 공집합이 아니며 단계 (II.1)에 의해서 q는 p에게 REPLY를 보낸 적도 없다. 그러므로 단계 (II.1)에 의해서 q는 그의 계승자들에게 시각 t3(>t2)에 PROBE를 보낸다. q가 p에게 t1 시각에 REPLY를 보낸다는 가정 때문에 q는 t1 시각 이전에 진행상태임이 성립한다. q가 진행상태이려면 OR 요청 모델이므로 적어도 하나의 계승자로부터 REPLY를 수신해야 한다. 그 시각을 t4라고 하자. 그러면 t3 < t4이다. 만약 (p,q)가 non-tree 간선이라면, q는 p가 아닌 다른 노드로부터 PROBE를 수신하여 그의 계승자들에게 t4 시각 전에 PROBE를 전달하였다. 따라서 귀납적으로, K∈DST이므로 단계 (II.1)에 의해서 PROBE는 K의 모든 노드들에게 전달된다. 또한 t4 < t1가 성립한다. 이는 K로부터 최초로 사라진 간선이 e라는 사실에 위배된다. 만약 (p,q)가 p에서 q로 전달된 CANCEL 메시지로 인해 사라졌다면 CANCEL이 전달되기 전 임의의 노드 r이 p에게 (p,r) 간선의 역방향으로 REPLY를 보냈을 것이다. 그렇다면 (p,q)에 적용된 위의 논리는 유사하게 (p,r)에 적용될 수 있다. 그러므로 K에서 사라지는 간선은 없으며 따라서 K∈WFG가 성립한다. ■

Corollary 2. 알고리즘 시작시간에 WFG가 교착상태를 포함하면 알고리즘은 이를 해결하며 그 반대의 경우도 성립한다.

증명. Theorem 4에 의하면 WFG가 knot을 포함할 때만 DST는 knot을 포함한다. Corollary 1에 의하여 DST가 knot을 포함할 때만 RWFG는 knot을 포함한다. 그러므로 알고리즘은 knot을 발견할 것이고 교착상태 해결 방법에 의해 이는 해결될 것이다. ■

5. 성능 비교 분석

5.1 Complexity

본 절에서 우선 알고리즘 실행 작업이 하나일 경우의 complexity(복잡도)를 고려한다. 임의의 두 노드간에 메시지 송신은 한 단위시간 내에 이루어진다고 가정하자. 알고리즘 수행에 참여한 WFG 간선 수는 e로 표기하고 노드 수는 n이라고 표기하자. 또한 d는 DST의 폭을 의미한다. 제안한 알고리즘은 각 간선당 하나의 PROBE를 전송한다. 응답 메시지는 진행상태 노드 또는 non-tree 간선 당 하나씩 시작노드에게 직접 전달된다. 진행상태 노드 수는 DST 내의 단노드 수보다 적다는 것을 감안하면 알고리즘이 발생시키는 전체 메시지 수는 $e+(e-n+1)+l$을 넘지 않는다 (l은 단노드 수). 시작노드에서 도달가능한 모든 교착상태를 발견하는데 걸리는 시간은 d+1이다. 이는 기존의 알고리즘[10,12,13]이 단지 시작노드가 교착상태에 속하였는지의 여부만을 판단하는데 2e 메시지와 2d 시간을 소요하는 것과 비교된다. 메시지 길이는 알고리즘에서 사용하는 path string 길이에 따라 결정된다.

Lemma 5. ancestor_i는 i의 DST 내 조상들의 집합이고 out_j는 j의 계승자 집합으로 표기하자. 그러면 $|pstr_i| = \sum_{j \in ancestor_i} f(|out_j|)$이다. 이 때 m>1이면 f(m)=$\log_2 m \rceil$ 이며 m=1일 때 f(m)=1로 정의한다.

증명. 노드 i가 DST의 근노드라면 $|pstr_i|$=0이고 이는 단계 (I)에서 근노드에 부여되는 λ의 길이와 같다. 노드 i가 근노드의 자노드라고 하자. 알고리즘의 proc_propagate에 따르면 근노드는 |succ_pstr|가 $\lceil \log_2|out_{root}| \rceil$ 인 PROBE를 송신하거나 또는 $|out_{root}|$=1일 경우에는 |succ_pstr|가 1인 PROBE를 송신한다. 따라서 단계 (II.1)에 의하여 노드 i는 자신의 path string을 $pstr_{root}$||succ_pstr 로 저장한다. 노드 i가 근노드의 자노드의 자노드일 경우 위와 유사하게 수행한다. 즉, 자신의 path string 길이를 $|pstr_j|$+|succ_pstr|로 하는데 이 때 j는 parent 이며 |succ_pstr|= $\lceil \log_2|out_j| \rceil$ 이거나 $|out_j|$=1인 경우 |succ_pstr| 는 1과 같다. 그러므로 노드 i의 path string 길이는 lemma에서 정의한 함수 f를 사용하여 f($|out_{root}|$)+ ⊦f($|out_j|$)임이 성립한다. 귀납적으로 lemma는 근노드의 후손에 모두 적용된다. ■

알고리즘의 성능을 비교하는데 단순히 한번의 실행을 가정하는 것은 부정확하다. 교착상태는 알고리즘이 추가적으로 시작되는 것을 방지하기 위해서라도 신속히 발견되고 해결되어야 한다. 이런 관점에서 특정 알고리즘이 교착상태가 발생한 후 이를 얼마나 신속히 발견하는지 평가하는 것은 매우 가치있는 일이다. 이는 다음 절에서 다룬다.

5.2 교착상태 발견 시간

제안한 알고리즘은 한번의 수행으로 시작노드에서 도달가능한 모든 교착상태를 발견하고 해결하는 특성을 가졌다. 이러한 성질이 없더라도 알고리즘이 분산적 성격을 지녔다면 교착상태에 속한 노드가 알고리즘을 시작하면 결국 임의의 교착상태는 발견될 것이다. 한번의 수행으로 모든 도달가능한 교착상태를 해결하는 것의

효율성을 알아보기 위하여 [12,13]의 알고리즘과 간단한 분석을 통하여 비교하고자 한다. [12,13]의 알고리즘은 시작노드가 교착상태에 속하였는지의 여부만을 판단한다.

대기 상태의 노드는 매 time-out(T_O) 시간마다 알고리즘을 시작한다고 가정하자. 같은 효과를 위하여 본 연구의 알고리즘에서는 임의의 노드가 시작노드에 상관없이 마지막으로 알고리즘 실행에 참여한 시간으로부터 T_O이 지나면 알고리즘을 시작하도록 한다. 이는 제안한 알고리즘에서는 노드가 알고리즘을 실행하면 그 노드로부터 도달가능한 모든 교착상태가 해결되기 때문이다. 그러나 [12,13]의 알고리즘에 의하면 시작노드는 자신이 교착상태에 속하였는지의 여부만을 알 수 있다. 그러므로 대기상태 노드는 매 T_O 시간마다 알고리즘을 시작해야 한다.

성능 측정 지수로는 알고리즘 시작 회수(N)와 교착상태 지속시간(D)을 사용한다. 우선 임의의 대기상태 노드인 v가 시작하는 알고리즘 횟수를 고려하자. 노드 v가 T_b 시간 동안 대기상태에 있다고 가정하자. [12,13]에서 각 노드는 독립적으로 알고리즘을 시작할 수 있으므로 노드 v는 대기상태에 있는 동안 매 T_O 시간마다 알고리즘을 시작한다. 그러므로 $N=\lfloor T_b/T_O \rfloor$이다. 본 연구의 알고리즘에서 N을 측정하기 위하여, 타노드가 시작하는 알고리즘을 v가 수행하는 시간 간격의 순열을 T_n, n=1,2,...로 표기하자. T_n 은 λ를 매개변수로 하는 지수(exponential) 분포함수에 의거하여 독립적이고 동일하게 분포되었다고 가정하자. 제안된 알고리즘에 의하면 노드 v는 T_n(n=1,2,...)이 T_O를 초과할 때만 알고리즘을 시작한다. 그러므로 노드 v가 알고리즘을 시작할 확률 p는 $P(T_n > T_O) = e^{-\lambda T_O}$ 이다. 따라서 노드 v가 T_b 시간동안 알고리즘을 i 번 시작할 확률은 ${}_kC_i p^i (1-p)^{k-i}$ 이며 여기서 $k=\lfloor T_b/T_O \rfloor$이다. 그러므로 노드 v의 평균 $N=\sum_{i=0}^{k} {}_kC_i p^i (1-p)^{k-i} i$ 이다. 예를 들어, 만약 T_b =500, T_O=100, 그리고 λ=0.01일 때 [12,13]에 따르면 $N=\lfloor 500/100 \rfloor$이고 본 알고리즘에 의하면 N≅1.8이다. 이러한 성능 차는 하나의 노드만을 고려한 것이므로 모든 노드들을 대상으로 할 때는 보다 현격한 차이가 드러날 것이다.

이제 교착상태 지속시간(D)을 측정하자. 시작노드에 관계없이 T_d를 알고리즘 시작 후에 교착상태를 발견하는데 걸리는 시간이라고 표기하자. 또한 교착상태에 속하는 모든 노드들은 거의 같은 시간인 t_c에 교착상태로 돌입했다고 가정하자. [12,13]에서 교착상태에 속한 노드는 t_c+T_O 시각에 알고리즘을 시작한다. 그러므로 교착상태를 발견하는 데 걸리는 시간은 T_O+T_d 이다. 그러나 본 연구의 알고리즘에서는 교착상태의 외부에서 이의 해결을 기다리고 있는 노드가 $t_{init}(t_c < t_{init} \le t_c+T_O)$ 시각에 알고리즘을 시작한다면 교착상태는 $t_{init}-t_c+T_d$ 시간만에 발견될 수 있다. 그와 같은 노드가 없을 경우에는 [12,13]의 경우와 마찬가지로 교착상태에 속한 노드가 알고리즘을 시작하면 교착상태 지속 시간은 T_O+T_d 가 된다. T를 $t_{init}-t_c$ 를 의미하는 확률변수라고 하고 이는 Υ를 매개변수로 하는 지수분포를 이룬다고 가정하자. 따라서 $D=\int_0^{T_o} td(P(T\le t)) + \int_{T_o}^{\infty} T_O d(P(T\le t)) + T_d$이며 이 때 $P(T\le t)=1-e^{-\Upsilon t}$ 이다. 만약 T_O=100, T_d=5, 그리고 Υ=0.01이라면 [12,13]에 의하면 D=105이며 본 연구의 알고리즘에 따르면 D≅68이다. 이 분석에서 간편성을 위하여 T_d는 시작노드에 상관없이 동일하다고 가정하였다. 그러나 시작노드가 교착상태의 외부에 있다면 그의 메시지가 교착상태의 노드까지 도달하는데 걸리는 시간도 정확성을 위해서 고려할 필요가 있다. 또한 위에서 사용한 가정들에 대한 실험적이고 분석적인 보다 철저한 점검이 필요하다.

6. 결 론

본 연구에서는 분산시스템에서 OR 요청 모델의 가정 아래 교착상태를 발견 및 해결하는 알고리즘을 제안하였다. 기존의 diffusing computation을 사용하는 알고리즘과는 달리 본 알고리즘은 응답 메시지 전송을 위한 시간을 감소시켰다. Probe의 전달과정으로 분산 탐색 트리가 형성되며 시작노드는 응답 메시지에 실린 정보를 근거로 교착상태를 발견하기 위해 축소된 WFG를 구축한다. 시작노드에게 송신되는 정보량을 줄이기 위해 노드간의 조상-자손 관계는 전달되지 않고 시작노드에서 추론되는 방안을 개발하였다. 제안한 알고리즘은 기존에 비해 여러 장점을 제공한다: 첫째, 알고리즘 수행 시간이 거의 반으로 축소되었다. 둘째, 시작노드는 기존처럼 단지 교착상태에 속하였는지만을 판단하는 것이 아니라 도달가능한 모든 교착상태의 존재를 발견한다. 셋째, 시작노드는 노드간 의존관계를 알 수 있기 때문에 교착상태의 해결문제가 용이하다. 기존 알고리즘은 해결방안을 제시하지 않거나 시작노드를 중지시킴으로써 해결하였다[10,12,13,14]. 제시한 해결방안은 RWFG의 노드들 중에서 희생자를 선정하였기 때문에, 즉, 교착상태의 해결을 위해 WFG의 모든 노드들을 고려대상으로 하지 않았기에, 모든 프로세스들이 동일한 우선순위로 취급되는 시스템에 적합할 것이다.

교착상태 지속시간과 알고리즘의 시작횟수 측면에서 기존의 알고리즘과 성능 비교분석을 제시하였다. 제안한 알고리즘에 따르면 교착상태 지속시간이 크게 단축되는 것으로 드러났으며 이는 알고리즘의 시작횟수도 현저히 줄이는 결과를 가져온다. 보다 상세한 실험적 또는 이론적 분석은 향후의 과제로 남아있다.

본 알고리즘은 시작노드가 교착상태 발견에 필요한 정보를 수집하는 것으로 미루어 보아 중앙집중식이라고 간주될 수도 있다. 그러나, 전형적인 중앙집중식 알고리즘에서와 같이 중앙 컨트롤러가 지정되어 있지 않고 또한 교착상태를 의심하는 노드는 언제든지 독립적으로 알고리즘을 시작할 수 있고 전체 시스템 상태를 유지관리하는 노드가 정해져 있지 않으므로 분산적 성격의 알고리즘이라 할 수 있다.

참 고 문 헌

[1] R. C. Holt, *"Some deadlock properties of computer systems," ACM Computing Surveys*, Vol. 4, pp. 179-196, 1972

[2] M. Singhal, *"Deadlock detection in distributed systems," IEEE Computer*, Vol. 22, pp. 37-48, 1989

[3] S. Lee and J. L. Kim, *"Performance Analysis of Distributed Deadlock Detection Algorithms," IEEE Transactions on Knowledge and Data Engineering*, Vol. 13, No. 4, pp.623-636, 2001.

[4] S. Lee, *"Fast detection and resolution of generalized distributed deadlocks," 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pp. 429-436, 2002.

[5] A. D. Kshemkalyani and M. Singhal, *"Distributed detection of generalized deadlocks," 17th Int'l Conf. Distributed Computing Systems*, pp. 553-560, 1997

[6] A. D. Kshemkalyani and M. Singhal, *"A one-phase algorithm to detect distributed deadlocks in replicated databases," IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 6, pp. 880-895, 1999

[7] A. V. Aho and J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974

[8] P. A. Bernstein and V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987

[9] G. Bracha and S. Toueg, *"A distributed algorithm for generalized deadlock detection," Distributed Computing*, Vol. 2, pp. 127-138, 1987

[10] N. Natarajan, *"A distributed scheme for detecting communication deadlocks," IEEE Transactions on Software Engineering*, Vol. 12, pp. 531-537, 1986

[11] E. W. Dijkstra and C. S. Scholten, *"Termination detection for diffusing computations," Information Processing Letters*, Vol. 11, No. 1, pp. 1-4, 1980

[12] A. Boukerche and C. Tropper, *"A distributed graph algorithm for the detection of local cycles and knots," IEEE Trans. Parallel and Distributed Systems*, Vol. 9, No. 8, pp. 748-757, 1998

[13] K. M. Chandy and J. Misra and L. M. Haas, *"Distributed deadlock detection," ACM Transactions on Computer Systems*, Vol. 1, pp. 144-156, 1983

[14] J. Villadangos and F. Farina and J. R. Mendivil, *"A safe distributed deadlock resolution algorithm for the OR request model, The 6th Euromicro Workshop on Parallel and Distributed Processing*, pp. 150-156, 1998

[15] I. Cidon, *"An efficient distributed knot detection algorithm," IEEE Transactions on Software Engineering*, Vol. 15, pp. 644-649, 1989

[16] J. Wang and S. Huang and N. Chen, *"A distributed algorithm for detecting generalized deadlocks,"* Tech. Rep., Dept. of Computer Science, National Tsing-Hua Univ., 1990

이 수 정

1985년 이화여자대학교 수학교육과 졸업. 1985년 ~ 1987년 삼성생명 정보시스템실 근무. 1988년 ~ 1994년 미국 Texas A&M 대학교 컴퓨터과학과 졸업(M.S, Ph.D). 1994년 ~ 1998년 삼성전자 통신개발연구소 근무. 1998년 ~ 현재 인천교육대학교 컴퓨터교육과 조교수. 관심분야는 분산시스템, 교착상태 알고리즘, 라우팅 알고리즘 등