모바일 컴퓨팅 환경에서의 토큰기반 상호배제 알고리즘

(A Token-based Mutual Exclusion Algorithm in Mobile Computing Environments)

양 승 일 † 이 태 규 † 박 성 훈 ††
(Seung-Il Yang) (Tae-Gyu Lee) (Sung-Hoon Park)

요 약 기존의 시스템에 적용되었던 상호배제 문제는 정적인 분산 컴퓨팅 환경에 적합하도록 설계되어 있다. 하지만 현재는 모바일 컴퓨팅환경이 진행되고 있으므로 정적 분산 환경에서의 상호배제 문제가 새로운 컴퓨팅 환경에 적용할 수 있도록 설계되어야 한다. 이를 위하여 본 연구에서는 모바일 컴퓨팅환경에 맞는 알고리즘을 연구하였다. 모바일 컴퓨팅환경이라는 새로운 환경에 알맞은 상호배제문제는 기존의 정적인 분산컴퓨팅환경의 상호배제보다 단말 이동성 및 자원 취약성 때문에 더 복잡한 시스템 구성을 보인다. 본 논문은 정적 분산 환경에서의 상호배제를 모바일 컴퓨팅 환경으로 확장 할 수 있는 새로운 상호배제 알고리즘을 제안한다. 모바일 분산시스템 노드들의 상호관계를 트리 구조로 나타내고 이동 호스트들 사이의 토큰 전달을 통해서 Deadlock과 Starvation으로부터 자유로운 상호배제를 지원하는 모바일 상호배제 알고리즘을 제안한다.

키워드 : 상호배제, 모바일 컴퓨팅, 알고리즘, 토큰

Abstract Mutual exclusion that applied on existing systems was designed for static distributed systems. but now computing environments are going to mobile computing environments. Therefore a mutual exclusion algorithm in static distributed environments should be designed for new computing environments. So this paper proposes a mobile mutual exclusion algorithm to support the mutual exclusion of shared resources in mobile computer environments. Mobile computing resources as wireless hosts cause new issues because of their mobility and weakness and made mutual exclusion problem more complex than stationary distributed environments. So we proposed a new mobile token mutual exclusion algorithm with deadlock-free and starvation-free in mobile computing environments based on spanning tree topology and extend for mobile computing environments. The proposed algorithm minimizes message complexity in case of free hopping in cellular networks.

Key words : Mutual Exclusion, Mobile Computing, Algorithm, Token

· 본 연구는 교육과학기술부와 한국산업기술재단의 지역혁신인력양성사업으로 수행된 연구결과임

† 정 회 원 : 충북대학교 컴퓨터공학과
ysipaul@yahoo.co.kr
tigerlee88@empal.com
(Corresponding author임)
†† 정 회 원 : 충북대학교 전기전자컴퓨터공학부 교수
spark@cbnu.ac.kr

논문접수 : 2009년 10월 29일
심사완료 : 2009년 11월 26일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제3호(2010.3)

1. 서 론

현재 컴퓨팅 환경은 정적인 분산 환경에서 유비쿼터스 환경으로 빠르게 변화하고 있다. 지하철이나 공항, 그리고 호텔, 세미나장, 대학, 병원 등 많은 공공장소에서 무선인터넷을 자유롭게 사용할 수 있는 환경이 되었다. 앞으로도 가정과 사무실 등 어디서나 무선인터넷을 자유롭게 사용하게 될 것이다. 발전을 거듭해가는 모바일 컴퓨팅은 결국 유비쿼터스 사용자를 위한 다양한 응용 서비스를 제공할 것이다. 이러한 컴퓨팅 환경의 변화에 따라 분산 공유자원에 대한 상호배제 문제도 지속적인 연구가 수행되어 왔다. 하지만 아직도 대부분의 상호배제 문제에 대한 연구는 정적인 경우로 국한 되어있다.

모바일 환경에서의 상호배제 문제는 좀 더 복잡한 문제들에 직면하게 된다. 모바일 환경에서 자원의 무결성을 보장하기 위해서 상호 배제가 필수적으로 요구된다. 자원의 분산, 광역정보의 부족, 그리고 전송의 지연은 모바일 분산 상호배제를 어렵게 만들었다. 고정 호스트들에 기초한 분산 상호배제는 많은 연구가 진행 되어 왔다[1-3,9-12]. 이러한 기존 연구를 모바일 환경에 적용할 경우 알고리즘의 안전성 및 성능에 저하를 가져올 수 있다.

기존에 제시된 분산 알고리즘에 대한 분류는 크게 토큰기반 알고리즘(token-based algorithm)과 비 토큰기반 알고리즘(none-token-based algorithm) 그리고 하이브리드 알고리즘(hybrid algorithm)등 세 가지로 분류를 나눌 수 있다[5-7]. 첫째, 토큰기반 알고리즘은 브로드캐스트 기반(broadcast-based) 알고리즘과 논리적 구조 기반(logical-structure-based) 알고리즘으로 구분되는데 브로드캐스트 기반 알고리즘은 정적(static)알고리즘과 동적(dynamic)알고리즘으로 다시 분류 된다. 정적알고리즘은 메모리가 없는 알고리즘이고 동적 알고리즘은 토큰 위치의 히스토리(history)를 기억하는 알고리즘 특성을 갖는다. 또한 논리적 구조 기반 알고리즘도 정적(static)알고리즘과 동적(dynamic)알고리즘으로 분류되고 정적 알고리즘은 링(ring), 트리(tree), 그래픽(graphic)으로 분류되는데 논리적 구조기반의 동적알고리즘의 대표적인 예는 Singhal 알고리즘이다[13]. 둘째, 비 토큰 기반 알고리즘은 퍼미션 기반(permission)알고리즘이라고도 하는데 일반적인(generalized) 형태와 Ricart-Agrawala 형태 그리고 Maekawa 형태로 분류된다. Ricart-Agrawala 형태는 정적인 것과 동적인 것으로 분류된다. 또한 Maekawa 형태는 Deadlock-경향(prone)과 Deadlock-자유(free) 알고리즘으로 분류 된다. 셋째, Hybrid 형태의 알고리즘은 분산 상호배제 알고리즘의 속도와 메시지 복잡도 사이에 존재하는 trade-off관계를 두 가지 모두 동시에 최소화하고자 제안된 알고리즘이다[14]. 기존연구에서 낮은 평균 메시지 복잡도를 나타내는 알고리즘은 논리적 구조기반의 알고리즘으로 이에 해당하는 Raymond 알고리즘(RM : Raymond Algorithm)은 논리적 구조가 변하지 않고 토큰의 방향을 가리키는 edge들의 방향만 변하는 알고리즘의 특징을 가지고 있다.

본 논문은 기존의 다양한 알고리즘 중에서 트리 기반 RM 알고리즘을 mobile에 적용할 수 있는 방법을 제시한다. RM 알고리즘은 평균적으로 $O(\log N)$의 메시지 복잡도를 나타내고[5] 지역(local) FIFO 큐와 토큰을 가리키는 변수를 중심으로 운행되는 알고리즘이다. 지금까지 제안된 많은 알고리즘은 정적 분산 환경에 적합하도록 설계되어 있어 모바일 환경과 유비쿼터스 환경에 적용하는 것이 어려웠다. 그러므로 본 논문은 기지국 기반 모바일 환경인 two-tier형태의 구조를 이용하여 이에 맞도록 RM 알고리즘을 수정 적용한다[8].

본 논문의 구성은 다음과 같다. 2장에서는 분산 상호배제의 관련 연구를 살펴보고, 3장에서는 모바일 환경에서의 상호배제 알고리즘 제시하며, 4장에서는 제안 알고리즘을 증명하고, 5장에서는 성능분석을 수행하고, 6장에서는 결론 및 향후 연구과제에 대하여 기술한다.

2. 관련 연구

분산 상호배제 알고리즘 중 정적인 환경에서 제안된 Ricart-Agrawala 알고리즘을 이동환경으로 확대 적용한 알고리즘[4]과 본 논문에서 수정 적용할 토큰기반 알고리즘인 RM 알고리즘[2]은 다음과 같다.

2.1 모바일 기반 Ricart-Agrawala 알고리즘

이 알고리즘은 Look-ahead technique개념을 통하여 모바일 환경에서도 정적 분산 컴퓨팅에서 제안되었던 Ricart-Agrawala 알고리즘을 효율적으로 사용가능하도록 제안하였다. 이 연구는 불필요한 트래픽을 제거하여 이동 컴퓨팅 환경에서 발생할 수 있는 메시지 트래픽을 효율적으로 처리하도록 제안하였다[4].

Look-ahead technique원리는 다음과 같다. 모든 노드들의 집합 S는 임의의 노드 S_i의 아이디로 구성된다. 노드들의 전체 집합 $S=\{S_1, S_2, S_3, ..., S_{n-1}, S_n\}$일 때 S를 S_i가 요청을 보낼 노드들의 집합 $info_set_i$와 S_i에게 요청을 보낼 노드들의 집합 $status_set_i$로 나눈다. 최초에 어떤 노드도 request를 요청하지 않았으면 모든 노드는 $status_set_i$의 원소가 된다. 임의의 노드 S_i가 S_j부터 요청메시지를 받으면 S_i는 S_j를 $info_set_i$에 넣는다. 만일 S_i가 CS에 들어갈 때 $info_set_i$의 S_j와 $info\_set_i$ 내의 모든 노드에게 요청메시지를 보낸다. S_i는 이런 방법으로 먼저 Look-ahead technique을 이용하여 $info_set_i$를 파악하고 요청을 보낼 때 $info_set_i$에게만 메시지를 보내면 불필요한 트래픽을 많이 줄일 수 있다. Look-ahead 상호배제 알고리즘은 임계구역에 대한 요청들이 지역화(localized)된다면 임계구역에 들어가기 원하는 메시지 트래픽을 감소시킬 수 있지만 모든 사이트들이 임계구역 사용을 요청한다면 Ricart-Agrawala알고리즘과 동일하다[3].

2.2 토큰 기반 Raymond 알고리즘

이 알고리즘은 트리구조를 기반으로 하는 토큰기반 상호배제 알고리즘이다[2]. 이는 트리의 acyclic 특성을 이용하기 때문에 "circular wait"의 잠재성을 구조적으로 제거하므로 교착상태로 부터 자유로운 알고리즘이다. 토큰기반 Raymond 알고리즘은 각 노드가 가지고 있는 토

큰 요구 메시지 큐인 REQUEST_Q의 순서가 선입선출(FIFO)을 기반으로 하고 있기 때문에 기근회피를 보장한다. 공정성(fairness) 측면에서는 정확한 공정성이 지켜지지 않지만 이것을 어느 정도 희생하더라도 메시지 복잡도와 성능측면에서 장점을 가진 알고리즘이다. 다음 그림 1은 Raymond 알고리즘 흐름을 나타내고 있다.

초기상태의 스패닝트리는 근(root) 노드가 없는 경우 그림 1에서 방향성을 갖지 않은 트리형태가 되지만 노드 E가 토큰을 가지고 있으면 그림 1과 같이 모든 노드들이 노드 E를 향하여 방향성을 갖게 된다. 이때 각 노드는 홀더(HOLDER)라는 변수를 가지고 있는데 이 변수는 토큰 노드를 가리키고 있는 변수이다. 이 HOLDER 값은 다음과 같이 변한다.

$HOLDER_A$=D, $HOLDER_B$=A $HOLDER_C$=A,
$HOLDER_D$=E, $HOLDER_E$=SELF, $HOLDER_F$=D

여기서 $HOLDER_X$=Y의 의미는 X에서 Y로의 방향을 나타내고 동작 방법은 노드 A가 임계구역에 들어가기 원할 때 요청 메시지를 $HOLDER_A$의 값 D에게 요청 메시지를 보낸다. 토큰을 가지고 있지 않은 노드D는 REQUEST 메시지를 $HOLDER_D$의 값 E에게 보낸다. 연속적인 REQUEST 메시지가 요청노드 A에서 토큰 노드 E 사이의 path를 따라 전달된다. E가 임계구역에서 나올 때 토큰 메시지를 D에게 보내고 $HOLDER_E$=D로 바꿔 준다. D도 A에게 토큰 메시지를 보내고 $HOLDER_D$=A로 바꾼다. 노드A가 토큰을 받으면 그림 1의 노드A처럼 HOLDER의 값이 변하여 토큰노드에 대한 방향성이 바뀐다. Raymond 알고리즘은 이웃노드들의 요청을 대신 전달하는 기법을 사용한 것으로 노드는 자신의 자식노드들에 대한 대표성을 가지고 행동한다.

이 알고리즘은 이동(mobile) 환경을 고려하지 않았기 때문에 이동 호스트가 이동하면서 발생할 수 있는 다음과 같은 문제점 들을 일으킬 수 있다. 첫째, 이동호스트 MH가 토큰을 요청하고 이동하는 경우 기근현상(starvation)이 발생한다. 둘째, 이동호스트 MH가 토큰을 소유하고 여러 셀을 이동하는 경우 데드락(Deadlock)이 발생할 수 있다. 이러한 문제가 발생할 수 있다는 것을 그림 2, 그림 3, 그림 4를 통하여 보여주고 있다. 그림 2는 모바일 호스트가 토큰을 요청하고 여러 셀을 이동하는 경우 토큰요청메시지를 계속 보낼 경우 이동하는 노드의 요청사항은 항상 큐의 마지막 요소(tail 또는 rear)에 항상 위치하게 되므로 최악의 경우 결국 토큰을 받지 못할 경우가 발생할 수 있다. 또한 토큰이 이동호스트가 이동하기 전의 고정호스트에 도달하였지만 이미 다른 셀로 이동한 경우 심각한 오류가 발생할 수 있음을 보여주고 있다. 그림 3과 그림 4는 이동호스트가 토큰을 소유하고 여러 셀을 이동하는 경우에 발생할 수 있는 문제점을 보여주고 있다. 그림 3은 이동호스트가 토큰을 소유하고 다른 셀로 이동을 했을 때 토큰을 사용을 마치고 현재 고정호스트에게 토큰을 전달해도 그 고정호스트가 어느 셀에 토큰을 전달해야할지를 모르는 경우가 발생하는 경우를 보여준다. 왜냐하면 최초에 토큰을 받은 셀에서 다음 토큰이 전달될 정보를 가지고 있는데, 현재의 셀에서 요청사항이 없는 경우 REQUEST_Q는 비어 있고 어느 노드에게 보내야할 지 모르는 상황이 발생할 수 있다. 또한 그림 4는 이동호스트

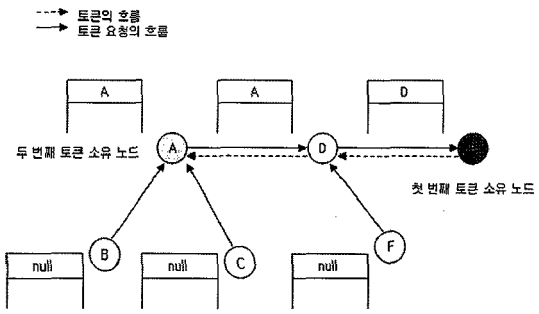

그림 1 Raymond 알고리즘 흐름도

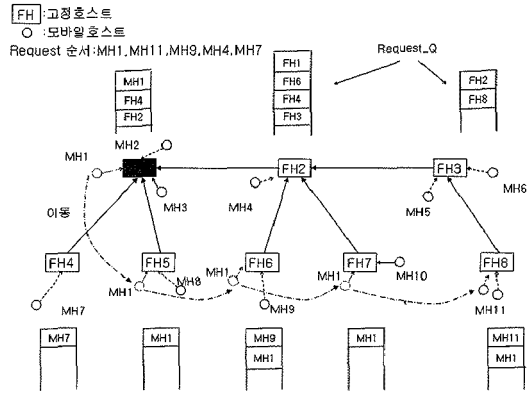

그림 2 MH 토큰을 요청하고 이동하는 경우

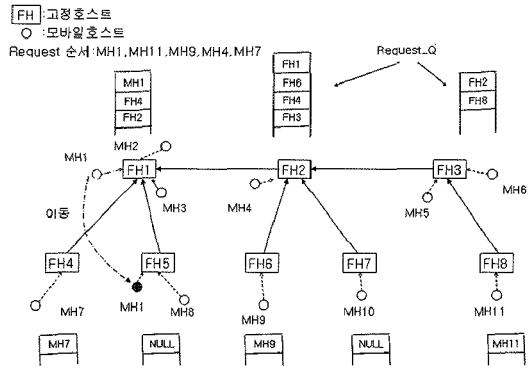

그림 3 MH가 토큰을 가지고 한 셀을 이동하는 경우

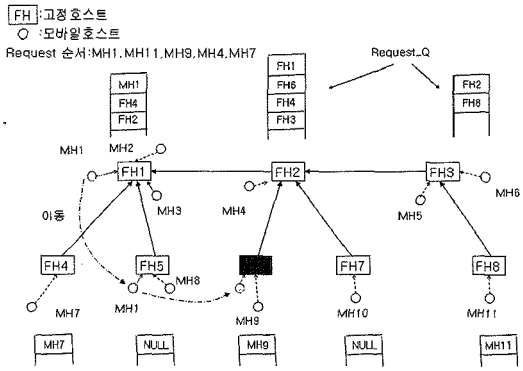

그림 4 토큰MH가 두 셀 이상 이동 후 토큰 반납하는 경우

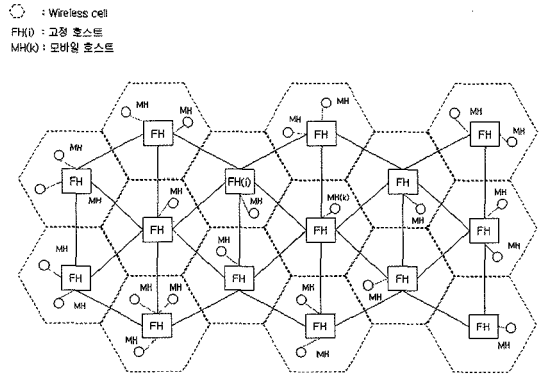

그림 5 시스템 모델

MH7이 토큰을 요청했지만 토큰을 가진 노드에게 요청 사항이 전달되지 않는 것을 나타낸다. 토큰을 가진 노드가 계속 다른 셀로 이동하다가 토큰을 FH6에게 전달하면 FH6은 토큰을 MH9에게 전달한다. MH7의 요청사항이 결국 토큰을 가진 노드에게 전달하지 못하는 경우가 발생할 수 있다. 그러면 MH7은 기근현상이 발생한다. 또한 요청사항이 없는 NULL상태의 고정호스트에 토큰을 전달하면 다른 요청사항이 전달되지 않아 데드락 상태에 진입한다.

이와 같이 Raymond 알고리즘은 이동환경에 적합하게 설계되지 않았으므로 이동환경에 적용하기에는 어렵다. 본 논문은 이러한 이동성 문제들을 고려하여 설계한 상호배제 알고리즘을 제안한다.

3. 모바일 환경에서의 상호배제 알고리즘

3.1 System모델 및 환경

모바일 환경에서는 기지국인 고정 호스트(FH : Fixed host)들과 이동호스트(MH : Mobile host)들로 구성되어 있기 때문에 시스템 모델은 그림 5와 같다.

이동(move)은 network 연결을 유지하면서 호스트가 이동할 수 있음을 의미하고, 네트워크 연결을 유지하면서 움직일 수 있는 단말을 이동호스트(MH)라 한다. 이동호스트들과 통신할 수 있는 기초 시스템을 고정호스트(FH)라 한다. 셀(cell)이란 물리적 또는 지리적으로 고정호스트인 기지국에 무선접속 영역(Coverage area)을 의미한다. 특정 셀 내에 있는 이동호스트 MH는 셀의 고정호스트 FH에 대해 로컬(local)이라 한다. 셀과 셀 사이네트워크는 물리적으로 완전히 연결된 상태(fully connected)이다.

이동호스트는 고정호스트와 직접 통신하고 모든 이동호스트와 고정호스트간의 통신, 고정호스트와 고정호스트간의 통신은 신뢰할 수 있다고 가정한다. 임의의 이동호스트는 항상 논리적으로 단지 하나의 셀에만 포함되어야 한다. 모든 고정호스트들 사이의 통신은 유선 네트워크로 구성되고, 고정호스트와 이동호스트사이는 무선 네트워크로 구성된다. 네트워크 구조는 고정 네트워크 사이는 논리적인 비용 트리(spanning tree) 구조이고 각 셀은 FH와 로컬 MH들 사이에 무선링크 1:N(FH:MH) 관계의 중앙 집중형 네트워크 구조를 갖는다. 고정호스트는 자신의 셀 내의 모든 이동호스트와 통신하지 않고 request 메시지에 대해 요청 수신(receive request), 요청 전달(transfer request), 토큰 전달(transfer token) 등을 책임진다. 무선 네트워크는 CDMA/GSM과 같은 중첩 셀(Overlapped cell)구조를 가지며 소프트 핸드오프(soft-handoff)에 기초한다.

본 시스템의 토큰은 항상 하나만 존재하고 이동호스트의 이동 상태는 이동하지 않은 상태(not moved), 이동한 상태(moved)가 있으며 이동 중(moving)의 상태는 이동하지 않은 상태나 이동한 상태중 하나로 결정된다. 고정호스트는 항상 이동호스트의 메시지를 전달하고 큐(queue)를 이용하여 토큰(token)을 관리한다. 그리고 고정호스트 자신이 임계구역에 들어가기 위한 자기요청(self request)은 없다. 이동호스트는 항상 현재의 고정호스트의 ID를 인식하고 있다. 이동호스트 MH는 계속 이동하다가 언젠가는 특정 임의의 최소의 단위시간(tu : time unit)동안 한 셀에서 반드시 멈춘다고 가정한다. 다음 그림과 같이 이 알고리즘은 토큰의 요구, 생성 및 토큰 전이(transfer)상황이 트리구조 및 토큰 메시지 큐의 자료 구조에 반영 되고 공유자원(임계구역의)의 상호배제를 지원한다.

그림 6은 그림 5의 모델을 논리적 트리형태로 구성한 것이다. 본 알고리즘은 그림 6과 같은 트리형태로 논리적인 구성을 기반으로 제안하였다.

고정호스트에서 관리하는 REQUEST_Q의 구조는 그림 7과 같다. req_id는 요청을 보낸 호스트의 id를 의미하고 origin_id는 MH가 최초에 CS를 요청한 셀id이다.

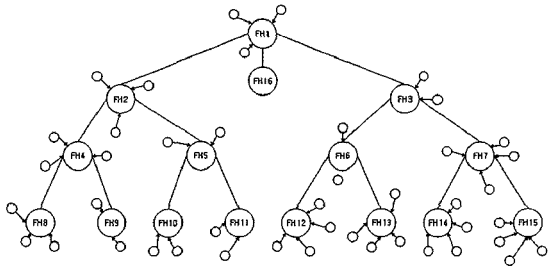

그림 6 논리적인 시스템 모델

그림 7 REQUEST_Q의 구조

또한 MH가 새로운 셀로 이동한 경우 new_id에 새로운 셀의 id를 입력한다. 만약 고정호스트가 MH를 대신하여 요청메시지를 보내는 경우 origin_id는 자신의 id가 되고 new_id는 null값 처리 된다. 또한 MH가 최초에 요청메시지를 보내는 경우 origin_id값과 new_id값은 동일한 값을 입력한다.

3.2 시스템 상태 전이

본 절은 고정호스트와 이동호스트의 접속상태 및 이동현황에 따른 상태 전이 등을 정의한다. 다음은 MH의 상태 전이를 나타내는 Local state와 이동호스트(MH)의 요청을 처리하고, 토큰을 관리하는 고정호스트(FH)의 상태전이를 나타내는 Global state이다. 여기서 최초의 셀을 로컬(local), 다른 셀을 원격(remote)라고 정의하였고, 이동호스트가 영향을 줄 수 있는 이동에 대하여 이동하지 않은 행위(static), 이동하여 셀을 나간 행위(move.out), 이동하여 셀을 들어온 행위(move.in)라 정의하였다. 또한 토큰을 요청하는 행위를 request, 토큰을 받는 것을 receive, 토큰을 사용 후 반환하는 것을 release라고 정의하였다.

첫째, 이동호스트의 상태전이를 정의 한다. 그림 8은 이동호스트의 상태가 전이되는 과정을 그림으로 나타내고 있다. 이동호스트의 상태를 토큰을 요청하지도 않고 토큰을 받지도 않은 상태(Mn), 토큰을 요청하고 기다리는 상태(Mw), 토큰을 가지고 있는 상태(Mh)의 3가지 상태로 나눌 때 이동호스트의 상태 집합(Sm)은 Sm = {Mn, Mw, Mh}이고, 그림 8과 같이 이동호스트의 상태가 전이된다.

둘째, 고정호스트의 상태전이를 정의 한다. 그림 9는 고정호스트의 상태 전이 과정을 보여준다. 고정호스트의 상태를 아무요청이 없는 상태(Fn), 토큰을 요청하고 기다리는 상태(Fw), 토큰을 소유한 상태(Fh)의 세 가지 상태로 나누고 고정호스트의 상태 집합(Sf)을 정의하면

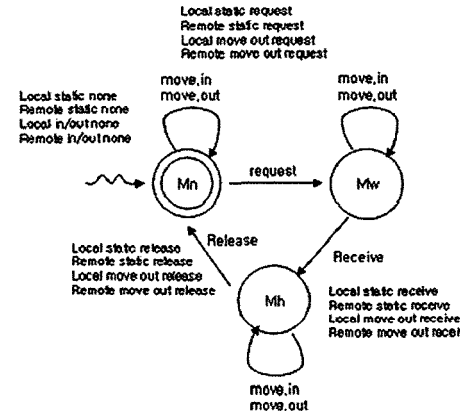

그림 8 이동호스트의 상태 전이도

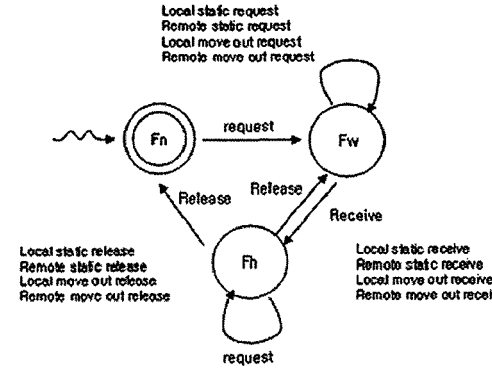

그림 9 고정호스트의 상태 전이도

Sf = {Fn, Fw, Fh}이라 할 수 있다. 이때 고정호스트의 상태는 이동호스트의 상태와 이동(move)에 영향을 받는다. 그림 9는 이동호스트의 이동에 따른 고정호스트의 상태를 나타낸다.

이동호스트의 상태와 고정호스트의 상태가 서로 결합하면 다음과 같이 집합으로 표시된다.

$$Sm \times Sf = \{(Mn, Fn), (Mn, Fw), (Mn, Fh), (Mw, Fn), (Mw, Fw), (Mw, Fh), (Mh, Fn), (Mh, Fw), (Mh, Fh)\}$$

여기서, 토큰이 동시에 2개 이상의 호스트에 존재할 수 없다는 가정에 의하여 (Mh, Fh)는 제거 가능하다.

본 논문은 그림 8 이동호스트의 상태 전이도와 그림 9 고정호스트의 상태 전이도 에서 보여주는 것처럼 입력되는 이벤트에 따라 상태가 바뀌는 것을 알고리즘에 반영하였다. 이동호스트의 상태는 알고리즘에서 이동호스트의 상태 변화를 나타내는 Boolean 변수인 asked, using 값을 통하여 이동호스트의 상태를 표시하였다. 두 변수 값이 모두 false인 경우는 아무 요청도 없는 none 상태를 의미한다. 고정호스트의 상태는 알고리즘에서

Boolean변수인 asked와 wait를 통하여 나타내었다. 변수 asked의 값은 CS를 요청하면 true가 된다. 고정호스트는 직접 CS를 사용하지 않으므로 using이라는 변수는 사용하지 않고 wait 변수를 이용하여 이동호스트에게 전달해 주기 전에 잠시 토큰을 소유하는 것을 표시해 준다. 고정호스트의 두 변수가 모두 false인 경우는 CS요청도 하지 않고 토큰을 소유하지도 않은 경우를 의미한다.

본 논문의 알고리즘에서 발생하는 이벤트는 request와 receive, release 그리고 move가 있다. 각 입력 이벤트에 따라 이동호스트와 고정호스트의 상태 변화에 대한 알고리즘 설계를 3.3절에서 제시한다.

3.3 알고리즘

3.3.1 토큰 기반 알고리즘

이동시 발생하는 토큰기반 알고리즘은 다음과 같은 순서로 진행된다.

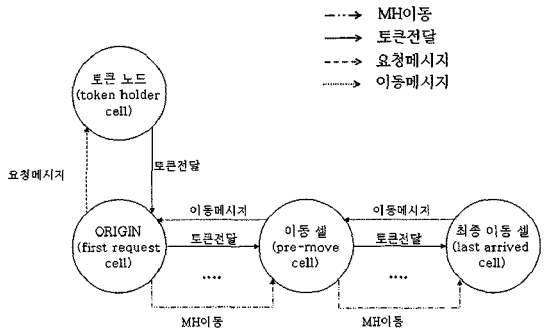

그림 10 토큰기반 알고리즘 흐름도

가) MH가 임계구역에 들어가려 할 때 토큰 메시지를 보낸다.

나) 토큰 요청 메시지를 처음 받은 FH(origin)은 자신의 큐에 저장하고 FH가 토큰 요청을 안했으면 토큰을 요청한다.

다) MH가 이동한 새로운 셀에서 이동(move) 메시지를 보낸다.

라) 이동 메시지를 받은 FH는 자신이 최초에 요청을 받은 FH(origin)이면 큐에서 해당 MH가 있는 현재 셀(new_id)정보를 업데이트 한다. 만약 자신이 origin이 아니면 이동호스트를 관리하는 list에 저장하고 origin에게 move메시지를 전달한다.

마) MH가 있었던 직전 FH는 자신의 list에서 MH정보를 삭제한다.

바) FH는 토큰을 받으면 req_id의 origin_id와 new_id값이 같으면 req_id에게 토큰을 전달하고 다르면 FH가 알고 있는 MH의 new_id에게 MH가 있는지 확인 메시지를 보낸다.

사) 확인메시지를 받은 new_id FH는 자신의 이동 list에서 해당 MH가 있는지 확인하고 있으면 ACK를 보내고 없으면 NACK을 보낸다.

아) origin FH는 new_id FH로 부터 응답메시지가 ACK이면 토큰을 보내주고, NACK이면 토큰보내는 것을 잠시 보류하며 MH로 부터 이동 정보가 도착할 때까지 기다린다.

자) FH는 MH의 이동 정보를 받으면 해당 MH를 관할하는 FH에게 토큰을 보낸다. 이때 목적지 req_id도 함께 알려준다.

차) MH는 FH로부터 토큰을 받고 임계구역에 들어간다.

카) 토큰사용을 마치면 현재 자신이 속해 있는 FH에게 토큰을 반납한다. 이때 origin FH정보를 알려주어 토큰이 origin FH로 가도록 한다.

타) origin FH는 토큰을 받아 큐의 다음 순서에게 토큰을 보낸다.

본 논문에서 제안하는 알고리즘은 그림 10에서 보여주는 것처럼 토큰 요청 전달, 토큰을 요청 후 이동하는 MH의 이동관리, 토큰 전달, 토큰을 소유한 후 이동한 MH관리 등 4가지 관점에서 바라본 알고리즘이다. 이동호스트가 토큰을 요청하면 이 요청사항이 토큰 이동호스트를 관리하는 고정호스트에게 도달한다. 토큰 호스트는 토큰을 전달하여 결국 토큰을 요청한 최초의 FH호스트(origin)가 받는다. 토큰을 받은 FH는 MH가 이동하지 않았으면 MH에게 토큰을 보내고, 이동하였으면 MH가 현재 있는 FH에 MH가 있는지 확인 메시지를 보내고, ACK메시지 응답을 받으면 토큰을 전달해 준다. new_id값에 있는 FH는 토큰을 전달받아 자신의 MH이동관리 리스트에 있는 이동호스트에게 토큰을 전달하고 해당 MH를 리스트에서 삭제한다.

3.3.2 이동호스트 알고리즘

```
MH(i) ::
변수정의
holder varchar //토큰을 갖은 host를 가리킴
asked boolean initially false;
using boolean initially false;
new_id varchar initially null; //이동한 셀 id
origin_id varchar initially null; //최초 request를 보낸 셀 id

1.  To request //request를 보낸다.
2.    origin_id = cell_id;
3.    if holder ≠ self ∧ ¬asked then
4.      asked := true;
```

```
5.    send request(req_id, origin_id, new_id, stat)
      to cell_id;
6.  endif

7.  To release //토큰을 반환한다.
8.    using := false;
9.    send token(origin_id) to cell_id;
10.   origin_id = null;

11. Upon Receiving token
12.   holder := self;
13.   if holder = self ∧ ¬using then
14.     asked := false;
15.     using := true;
16.     <initiate entry into critical section>
17.   endif

18. Upon arriving
19.   new_id := cell_id;
20.   if asked = true then
21.     send move(req_id, origin_id, new_id) to
        cell_id;
22.   endif
```

3.3.3 고정호스트 알고리즘

```
FH(j) ::
변수 정의
REQUEST_Q FIFO Queue;
list MRequest; //이동 request를 저장하는 list로 구
조는 REQUEST_Q와 동일한 구조체
holder varchar initially indicate to token node; //
토큰을 갖은 host를 가리킴
new_id varchar initially null; //이동한 셀 id
origin_id varchar initially null; //최초request를 보
낸 셀 id
asked boolean initially false; //request를 보내면
true, 안보내면 false
wait boolean initially false;

1.  Upon receiving request
2.    addqueue(REQUEST_Q, id, origin_id, new_id);
3.    make_req();

4.  Upon receiving move
5.    if my_id = move.origin_id ∧ wait = true
      then //MH move 메시지 처리
6.      send token(req_id) to move.new_id;
7.    wait = false;
8.  else if my_id = move.origin_id ∧ wait = false
    then
9.    update(REQUEST_Q, req_id); //req_id의
      new_id정보를 업데이트 한다.
10.   else if my_id = new_id then
11.     send move to move.origin_id;
12.     add(Mrequest[i], move); //move메시지 저장
13.   endif

14. Upon receiving token
15.   if token_id = Mrequest[i].req_id then
16.     send token to Mrequest[i].req_id;
17.   else
18.     holder := self;
19.     assign_token();
20.     if wait ≠ true then
21.     make_req();
22.     end if
23.   end if

24. Upon receiving ACK
25.   send token(req_id) to new_id;
26.   wait = false;

27. Upon receiving search
28.   if req_id ∈ Mrequest[i] then
29.     send ACK to sender;
30.   else
31.     send NACK to sender;
32.   endif

33. assign_token():
34.   if holder = self ∧ REQUEST_Q ≠ empty then
35.     holder := delqueue(REQUEST_Q. req_id);
36.     asked := false;
37.     if new_id ≠ my_id then
38.       send search(req_id) to new_id;
39.       wait = true;
40.     else
41.       send token(req_id) to new_id;
42.     end if
43.   endif

44. make_req():
```

45. if holder ≠ self ∧ REQUEST_Q ≠ empty ∧ ¬asked then
46. send request to holder;
47. asked := true;
48. endif

위 알고리즘에서 한 FH가 move 메시지를 받았을 때, move 메시지내의 origin_id 필드 값이 FH 자신의 id와 동일하고 현재 ACK신호를 기다리는 중이면 토큰을 보내주고, 그렇지 않으면 Request_Q list에서 해당 req_id의 new_id를 업데이트한다. 만약 move메시지의 origin_id가 FH자신의 id가 아닌 경우는 origin_id에게 move메시지를 전달한다. 그리고 search메시지를 받고 해당 req_id가 자신의 리스트에 있는지 확인하여 있으면 ACK를 보내주고 없으면 NACK를 보내준다. search메시지를 보낸 FH는 응답이 올 때까지 기다린다. 다음 4장에서 본 알고리즘에 대한 증명을 진행 한다.

4. 증 명

본 장은 위 3장에서 제시한 모바일 상호배제 알고리즘이 상호배제를 보장하고 교착상태가 발생하지 않으며 기근상태가 발생하지 않는 것을 증명한다.

4.1 상호배제 보장

본 논문의 알고리즘은 이동호스트가 토큰 메시지를 요청하고 받았을 때만 임계구역(CS)에 들어갈 수 있다. 따라서 상호배제를 보장하기 위해서는 임의의 순간에 특정 노드만이 토큰 메시지를 갖도록 보장해야 한다. 본 논문은 고정호스트뿐만 아니라 이동호스트가 이동하는 경우에도, 항상 최대 하나의 노드만이 토큰을 갖도록 설계하여 상호배제를 보장한다. 다음은 노드의 프로세스 진행 상태에 기초한 상호배제에 대한 정리이다.

Theorem 1. 두개의 다른 프로세스가 동시에 임계구역에 들어 갈 수 없다(Safety property).

임의의 두 노드가 존재하고 각 노드마다 프로세스가 한 개씩 있다고 할 때, 두개의 각 노드의 프로세스는 독립적이고 동시에 존재하는 상태 s와 t가 있다. 상태 s와 t가 같지 않다면 임계구역에는 두개의 상태에서 요청한 상태가 동시에 존재할 수 없다.

가정 1. 두개의 다른 프로세스가 동시에 임계구역에 들어 갈 수 있다.

증명. 가정1에 대한 증명은 다음과 같다. 토큰을 가지고 있는 이동호스트는 임의의 호스트이며 편의상 특정 이동호스트로 표현한다. 이동호스트의 프로세스는 토큰을 요청하고, 받고, 전달하고, 이동한다고 할 때 두개의 이동호스트가 동시에 토큰을 요청하면 다음과 같은 절차로 진행된다.

1. 임의의 이동호스트 MH(z)가 토큰을 가지고 있고, 임의의 이동호스트 MH(x), MH(y)가 각각 동시에 토큰을 요청 한다. 이동호스트 MH(z)의 고정호스트는 FH(z)라 하고, 이동호스트 MH(x)와 MH(y)의 고정호스트는 FH(x), FH(y)라고 한다.
2. 고정호스트 FH(z)는 고정호스트 알고리즘 1번째 줄에서 FH(x), FH(y)가 보낸 MH(x)와 MH(y)의 토큰 요청메시지를 FIFO형태의 큐에 보관한다.
3. 고정호스트 FH(z)는 고정호스트 알고리즘 33번째 줄에서 이동호스트 MH(z)로 부터 토큰을 받으면 자신 큐의 top 노드에게 토큰을 보낸다.
4. 위(3)과정에서 전달된 토큰메시지는 FH(x) 또는 FH(y)가운데 하나의 고정호스트만이 수신한다. 만일 두 이동호스트가 하나의 고정호스트에 존재 한다면 즉 고정호스트 FH(x)의 셀에 두개의 이동호스트가 토큰을 동시에 요청하였으면 고정호스트 FH(x)가 수신한다.
5. 모든 이동호스트의 토큰요청은 항상 고정호스트의 REQUEST_Q안에 FIFO형태로 관리된다. 그러므로 위(4)에서 토큰을 수신한 고정호스트는 항상 하나의 토큰요청 이동호스트에게만 토큰을 전달한다.

따라서 위(5)의 결론은 두개의 토큰 요청상태 s, t가 동시에 임계구역에 들어갈 수 있다는 가정에 위배된다.

인근에 있는 이동호스트와 고정호스트의 토큰 요청 상태를 고정호스트의 FIFO형태의 REQUEST_Q에서 관리하고 있으므로 REQUEST_Q안에 있는 순서가 유지되는 동안 항상 상호배제가 유지된다. 만일 토큰을 요청한 이동호스트가 이동하면 이동호스트 알고리즘 18번째에서 이동(move)메시지를 보내준다. 이동메시지를 받은 고정호스트는 고정호스트알고리즘 4번째 줄에서 최초에 토큰을 요청했던 고정호스트(origin)에게 이동호스트의 위치를 알려준다. 이동호스트가 토큰을 처음 요청했던 고정호스트는 이동 정보를 받고 이동호스트의 위치를 업데이트한다. 토큰요청을 처음 받은 고정호스트는 마찬가지로 REQUEST_Q에서 모든 토큰요청을 항상 관리하므로 항상 하나의 프로세스만 토큰을 수신할 수 있으므로 상호배제가 항상 이루어진다.

4.2 교착 상태 자유(Deadlock free)

Theorem 2. 제안 알고리즘은 교착상태가 발생하지 않는다.

본 논문의 알고리즘은 기존 알고리즘과 달리 다음의 경우에서도 교착상태가 발생하지 않는다.

(1) 어느 노드도 토큰을 가지고 있지 않으므로 토큰 메시지를 다른 노드에 전송할 수 없는 경우
(2) 토큰을 가진 노드가 다른 노드들의 토큰 요청 메시지를 전달받지 못할 경우

(3) 네트워크 장애나 노드의 장애가 존재하지 않는데도 토큰메시지가 토큰을 요청한 노드까지 도달할 가능성이 없는 경우

증명. (1)의 경우는 장애가 발생하여 토큰 노드가 없어지거나 토큰이 전달되는 과정에서 네트워크 장애로 없어지는 경우에 발생할 수 있다. 그러나 본 논문은 FAULT FREE이고 장애가 발생하지 않는 네트워크를 가정하기 때문에 발생 불가능하다. 그러므로 본 논문은 네트워크 장애나 노드의 장애가 존재하지 않는 (2)와 (3)이 발생하는 경우에 대하여 고찰한다. 증명을 위하여 다음과 같이 두 가지를 가정할 수 있다.

가정 1. 토큰을 가진 노드가 다른 노드들의 토큰 요청 메시지를 전달받지 못한 경우가 발생한다.

가정 2. 토큰메시지가 토큰을 요청한 노드까지 도달하지 못한다.

증명. 가정1에 대한 증명은 다음과 같다.

임의의 이동호스트가 토큰을 가지고 이동하다가 토큰을 반환할 때 REQUEST_Q에 전혀 토큰요청이 없는 고정호스트에 토큰을 전달하면 발생할 수 있다. 가정1의 증명을 위해 임의의 이동호스트 MH(x)가 토큰을 가지고 있고, 임의의 이동호스트 MH(y), MH(z)가 토큰을 요청 한다고 가정하면 이동호스트 MH(z)를 관리하는 고정호스트는 FH(z), 이동호스트 MH(x)와 MH(y)의 고정호스트는 FH(x), FH(y)라고 한다.

1. 이동호스트 MH(x)가 임계구역에 들어가기 위해 요청을 보낸다. 고정호스트 FH(y)는 이동호스트의 요청을 자신의 REQUEST_Q에 저장하고, 이동호스트를 대신하여 토큰 요청을 FH(z)에게 보낸다.
2. 알고리즘에 의하여 이동호스트 MH(x)가 토큰을 받는다. MH(x)는 자신의 상태변수 holder=self로 설정하고 토큰을 사용한다.
3. MH(x)는 토큰을 가지고 FH(y)로 이동한다. 이동 후 MH(x)는 토큰을 반납할 때 자기가 현재 속해 있는 셀의 고정호스트에게 전달하면 고정호스트는 항상 토큰을 MH(x)에 전달했던 고정호스트에게 보내준다. 그렇지 않고 계속 사용할 때는 고정호스트에게 특정 메시지를 보내지 않고 고정호스트와 일반적인 통신을 통해 연결하며 토큰을 계속 사용한다.
4. MH(x)가 토큰을 FH(y)나 다른 고정호스트에 반환한다.
5. FH(z)나 다른 고정호스트는 토큰을 FH(x)에 전달한다. FH(x)는 자신의 REQUEST_Q의 top에 있는 이동호스트에게 토큰을 전달한다.

위의 알고리즘에서와 같이 MH(x)가 이동할 때 토큰 요청이 없었던 셀로 이동한다 할지라도 본 논문의 알고리즘에 의하여 토큰은 항상 고정호스트의 REQUEST_Q에 있는 다음순서로 전달되므로 교착상태가 발생한다는 가정 1은 거짓이 된다. 이동호스트에게 토큰을 보낸 고정호스트는 이동호스트가 다른 셀로 이동을 해도 항상 토큰을 반납 받게 된다. 본 논문은 고정호스트가 토큰관리를 하기 때문에 이동호스트가 다른 이동호스트의 토큰 요청을 직접 알지 못해도 전혀 문제가 발생하지 않는다.

다음은 가정 2에 대한 증명이다. 가정 2는 임의의 이동호스트가 토큰을 요청하였지만 토큰을 가진 이동호스트의 잦은 이동으로 인하여 특정 토큰 요청에 대하여 처리하지 못하는 경우 발생할 수 있다. 가정 2의 증명을 위하여 임의의 이동호스트 MH(z)가 토큰을 가지고 있고, 임의의 이동호스트 MH(x), MH(y)가 토큰 사용을 요청 한다고 가정하면 이동호스트 MH(z)의 고정호스트는 FH(z), 이동호스트 MH(x)와 MH(y)의 고정호스트는 FH(x), FH(y)라고 한다.

1. 이동호스트 MH(x)가 토큰 요청 메시지를 보낸다. 고정호스트 FH(y)는 자신의 REQUEST_Q에 저장하고, 요청사항을 FH(z)에게 보낸다.
2. 이동호스트 MH(x)가 셀을 이동할 때 이동 후 새로운 셀에서 MH(x)는 자신이 토큰을 요청한 상태라면 이동 메시지를 자신이 도착한 셀의 고정호스트 FH(y)에게 보낸다.
3. 이동 메시지를 받은 고정호스트 FH(y)는 자신의 리스트(list)에 이동 메시지를 넣고 고정호스트 FH(x)에게 이동호스트 MH(x)의 이동 메시지를 전달해 준다.
4. 이동호스트 MH(x)가 다시 다른 셀로 이동할 때 이동호스트 MH(x)는 이동 후 새로운 셀의 고정호스트 FH(z)에게 이동 메시지를 보낸다. 이동호스트 MH(x)가 셀을 나가면 고정호스트 FH(y)는 자신의 리스트에서 이동호스트 MH(x)를 삭제한다.
5. 고정호스트 FH(z)는 이동호스트 MH(x)가 처음 토큰 사용 요청을 했던 고정호스트 FH(y)에게 이동호스트 MH(x)의 이동 메시지를 전달하고, 이동호스트 MH(x)를 자신의 리스트에 입력한다.
6. 고정호스트 FH(x)는 이동호스트 MH(x)의 현재 셀 정보를 업데이트 한다.
7. 고정호스트 FH(x)가 토큰을 받으면 고정호스트 FH(x)는 자신이 알고 있는 이동호스트 MH(x)의 현재 고정호스트에게 MH(x)가 있는지 확인 요청메시지를 보낸다.
8. 고정호스트 FH(z)는 이동호스트 MH(x)가 자신의 리스트에 있으면 응답으로 ACK메시지를 보내주고 토큰을 받으면 해당 이동호스트 MH(x)에게 토큰을 전달한다.

위의 과정으로 이동호스트의 이동에 따른 토큰요청과 토큰전달이 이루어지므로 가정 2는 우리가 제시한 알고

리즘에서는 발생하지 않으므로 거짓이 된다.

본 논문은 이동호스트가 토큰을 요청하고 이동할 때 자신의 현재 고정호스트 정보를 토큰을 처음 요청한 고정호스트에게 항상 보내주어 고정호스트들 간에 동기화를 시킨다.

가정(2), 가정(3)이 거짓이므로 본 논문에서는 순환대기 상태가 발생하지 않고 변수 이동호스트의 상태변수 asked와 holder를 이용하므로 반복적인 토큰요청 메시지를 방지한다. 그러므로 가정(2)와 가정(3)은 발생하지 않으므로 교착상태가 발생하지 않는다.

4.3 기근상태 자유

본 논문은 이동호스트의 이동성을 고려하기 때문에 이동호스트의 이동이 발생할 때에도 고정호스트에서 관리하는 큐가 FIFO상태를 유지하는 것이 중요하다.

Theorem 3. 임계구역에 들어가고자 토큰을 요청하는 모든 이동호스트는 언젠가는 토큰을 받는다(Liveness property).

가정. 고정호스트의 REQUEST_Q는 FIFO가 유지되지 않는다.

임의의 이동호스트 MH(z)가 토큰을 가지고 있고, 임의의 이동호스트 MH(x), MH(y)가 임계구역 사용을 요청 한다고 가정하면 이동호스트 MH(z)의 고정호스트는 FH(z), 이동호스트 MH(x)와 MH(y)의 고정호스트는 FH(x), FH(y)라고 하자.

1. 이동호스트 MH(x)와 MH(y)가 토큰을 요청하면 고정호스트 알고리즘 1번째줄에서 고정호스트 FH(x)는 MH(x)의 요청사항을, 고정호스트 FH(y)는 MH(x)의 요청사항을 각각 자신의 REQUEST_Q에 저장한다.
2. 이동호스트 MH(x)가 고정호스트 FH(y)의 셀로 이동한 후 토큰을 요청했으면 이동호스트 MH(x)는 고정호스트 FH(y)에게 이동 메시지를 보낸다.
3. 고정호스트 FH(y)는 이동 메시지를 처음 토큰을 요청했던 고정호스트 FH(x)에 전달한다.
4. 이동 메시지를 받은 고정호스트 FH(x)는 자신의 큐에 이동호스트 MH(x)의 현재 셀 정보(new_id)를 업데이트 한다.
5. 토큰이 고정호스트 FH(x)에 도착하면 REQUEST_Q의 순서대로 토큰을 전달한다. 고정호스트는 토큰요청메시지를 항상 처음 순서를 유지하면서 REQUEST_Q를 관리하므로 각 노드의 큐는 FIFO를 유지하지 못한다는 가정에 위배된다.

모든 고정호스트는 이동호스트의 토큰요청을 큐에 저장하고 항상 FIFO상태를 유지시키므로 토큰 사용을 요청한 이동 호스트는 언젠가는 토큰을 사용하게 된다.

5. 성능분석

본 장에서는 4장에서 제시한 알고리즘에 대한 성능분석을 실시한다. 본 논문에서 제시한 알고리즘과 Raymond알고리즘을 이동환경에 적용하였을 경우 발생하는 메시지 수 분석을 통하여 성능분석을 실시하고, 관련 연구에서 제시한 모바일 환경에서의 Ricart-Agrawala 알고리즘과 Raymond 알고리즘의 성능을 비교 한다. 여기서 Ricart-Agrawala 알고리즘을 RAMX, Raymond 알고리즘을 RMX, 본 논문에서 제시한 알고리즘을 MMX로 간단히 표시 한다.

본 논문에서 제시한 논문의 전체 노드를 N이라 하고 최대값을 n이라 할 때 N={1, 2, 3, ..., n}이 된다.

제안 알고리즘에서의 전체 노드는 고정호스트와 이동호스트로 구성되어 있다. 전체 노드수를 N이라 하고, 고정호스트수를 FN, 이동호스트수를 MN이라 할 때

$$N = FN + MN \quad (1)$$

이 된다. 위의 노드들에 대한 성능 분석은 다음과 같다.

5.1 MMX의 알고리즘 비용

MMX의 알고리즘 비용은 이동호스트가 이동하지 않은 경우와 이동한 경우로 구분한다.

5.1.1 이동호스트가 이동하지 않은 경우

이동호스트가 이동하지 않은 경우 메시지 복잡도는 네트워크의 위상(topology)에 따라 달라지는데 각 위상에 따라 알고리즘 비용은 다음과 같다. 알고리즘 비용은 CA로 표시한다.

• Radiating star topology인 경우(best case)

$$CA = 2*(2[\log K-1(((N-1)(K-2)/K) + 1)]) \quad (2)$$

여기서 K는 고정호스트들 간의 결합가(degree) 즉 이웃하고 있는 고정호스트들의 수이다. 이 경우의 RMX와 MMX의 알고리즘 비용은 동일하다.

• Straight line topology인 경우(worst case)

$$CA = 2*((TN+1)/3) \quad (3)$$

또한 이 경우에 트리구조와 상관없이 RMX와 MMX의 알고리즘 비용은 동일하다.

일반적으로 알고리즘 비용은 고정호스트와 이동호스트를 구분하여 고정호스트 수는 FN, 이동호스트 수는 MN이므로 전체 노드에 대한 알고리즘 비용은 logN= log(FN+MN)이 된다.

5.1.2 이동호스트가 이동한 경우

여기서는 전체노드를 고정호스트와 이동호스트로 구분 한다. 알고리즘 전체 비용은 전체 노드에 대한 소요비용 logN과 이동호스트의 이동 비용을 더한다. 이동호스트의 이동시 모든 노드에 영향을 주지 않고 이동호스트가 속해 있는 고정호스트만 영향을 받으므로 이동수 만큼 메시지를 전달하게 된다. 그러므로 이동비용 또는 이동 횟수를 MC라 할 때 전체 소요비용은 logN + MC 즉 $O(\log N + MC)$가 된다.

5.2 모바일 환경에서의 메시지 복잡도 비교

다음은 RMAX, RMX, MMX를 모바일 환경에 적용하였을 때 메시지가 발생한 복잡도를 계산하여 비교하였다. 메시지 종류는 요청(request) 메시지, 토큰(token) 메시지, 이동(move) 및 업데이트 메시지, 확인(search) 메시지가 있다. 이러한 메시지들의 발생 횟수를 계산하여 성능을 분석하였다. 이동호스트가 이동하지 않은 경우는 다음 표 1과 같이 RMX와 MMX의 메시지 복잡도는 동일하다.

이동호스트가 이동한 경우는 요청 메시지를 보내고 이동하는 경우, 토큰을 소유하고 이동하는 경우로 나누어 발생하는 메시지 수를 계산하였다.

이동호스트 이동에 대한 각각의 경우에 대해서는 표 2, 표 3에 나타내었다.

표 1, 표 2, 표 3에서 알 수 있는 것은 이동 컴퓨팅 환경이 아닌 경우 표 1과 같이 RMX와 MMX의 메시지 발생 횟수가 동일하지만 모바일 환경인 경우 표 2, 표 3에서 볼 수 있는 것처럼, RAMX나 RMX 보다 MMX가 메시지 발생횟수가 적음을 보여준다. 이러한 결과를 토대로 표 4는 제안 알고리즘과 기존의 알고리즘을 모바일 환경에 적용할 경우의 성능을 비교한 결과

표 1 이동이 없는 경우 발생 메시지 수

메시지 종류	RAMX	RMX	MMX
요청	Φ	O(logN)	O(logN)
토큰	Φ	O(logN)	O(logN)
확인	0	0	0
이동 및 업데이트	0	0	0

N : 전체 노드 수, MC : 셀을 이동한 횟수, 파이(Φ) : RAMX에서 사용하는 집합, PN : 이동호스트의 이동전 고정호스트와 ROOT 노드사이의 요청 패스(ORP : Old Request Path)와 이동후의 고정호스트와 ROOT노드사이의 새로운 요청 패스(NRP : New Request Path)상에 있는 고정호스트 수를 의미한다. PN = ORP + NRP이다.

표 2 요청 후 이동한 경우 발생 메시지 수

메시지 종류	RAMX	RMX	MMX
요청	Φ	O(logN)	O(logN)
토큰	Φ	0	0
확인	0	0	1
이동 및 업데이트	Φ	MC*PN	MC

표 3 토큰소유 후 이동한 경우 발생 메시지 수

메시지 종류	RAMX	RMX	MMX
요청	Φ	0	0
토큰	Φ	O(logN)	O((logN)
확인	0	0	0
이동 및 업데이트	N	MC*N	0

표 4 알고리즘 성능 분석

알고리즘	RAMX	RMX	MMX
이동성	O	×	O
기근상태	×	O	×
교착상태	×	O	×

를 나타낸 것이다.

본 논문에서 제시한 알고리즘은 이동호스트의 이동이 없는 경우 RMX와 복잡도가 동일하고 RAMX보다는 더 좋다. RAMX는 최악의 경우에는 전체 메시지 전달이 발생할 수 있다. 이동고정호스트가 이동하는 경우 RMX는 이동성을 고려하지 않았다, 그러나 만일 이동환경에 적용한다고 가정하면 표 1, 표 2, 표 3에서 보는 바와 같이 MMX의 성능이 우수하다. 또한 RAMX에 비해서도 메시지 복잡도가 O(logN+MC)이므로 알고리즘의 성능이 좋다.

본 논문의 제안 알고리즘은 표4에서 보여주는 것과 같이 기존의 RMX 알고리즘에 이동성을 부여하였을 때 발생할 수 있는 기근상태와 교착상태가 발생하지 않는다. 또한 이동환경에서의 메시지 복잡도를 계산하면 위 표 2, 표 3에서 보여주는 것처럼 적은 비용이 발생한다.

본 논문에서 제시한 MMX알고리즘은 이동성을 고려하여 모바일 환경에 적용가능하고, 기존 알고리즘보다 더 좋은 성능을 보여주고 있다.

6. 결 론

본 논문은 모바일 환경에서의 상호배제문제를 다루었다. 컴퓨팅에 참여하는 호스트는 고정호스트와 이동호스트를 구분하였고 각각의 알고리즘을 제시하였다. 고정호스트는 유선환경에 기초하였기 때문에 미세한 알고리즘의 복잡도와 데이터의 양에 영향을 받지 않는다. 그러나 이동호스트인 경우에는 저 전력 메모리, CPU를 사용하기 때문에 간편한 알고리즘을 가지고 운행이 되어야만 효과를 낼 수 있다. 이런 의미에서 본 논문은 이동호스트 측면에서의 복잡도를 O(logN+MC)로 단순화하였고 저장 공간과 처리되는 데이터양도 최소화하였다.

향후 연구과제는 이동호스트간의 애드혹(ad hoc)통신에 기초한 상호배제 알고리즘을 제안하기위해 본 알고리즘의 변형 및 확장이 요구된다. 또한 본 알고리즘은 이동호스트의 이동에 대한 correctness를 보장하기 위한 방법 중 메시지 복잡도를 늘리면서 시간 지연을 적게 하는 방법과 시간지연(delay)을 늘리면서 메시지 복잡도를 작게 하는 것에 대한 trade-off가 발생한다. 본 논문에서는 토큰전달 시간을 어느 정도 지연하면서 메시지 복잡도를 줄이는 방향으로 본 알고리즘을 제안하였다. 하지만 이에 대한 미세한 연구를 통하여 좀 더 시간지

연 증가와 메시지 증가에 따른 trade-off개선을 위한 연구가 필요하다.

참 고 문 헌

[1] M. A. Maekawa. "A rootN Algorithm for Mutual Exclusion in Decentralized Systems," *ACM Trans. Comput. Systems*, vol.3, pp.145-159, 1985.

[2] K. Raymond. "A Tree-based Algorithm for Distributed Mutual Exclusion," *ACM Bans. Comput. Systems*, vol.7, pp.61-77, 1989.

[3] G. Ricart and A. K.. Agrawala. "An Optimal Algorithm for Mutual Exclusion in Computer Networks," *Communications of the ACM*, vol.24, pp.9-17, 1981.

[4] Mukesh Singhal and D. Manivannan, "A Distributed Mutual Exclusion Algorithm for Mobile Computing," *Proc. of the 1997 IASTED International Conference on Intelligent Information Systems (IIS '97)*, pp.557, 1997.

[5] Ye-In Chang, "A simulation study on distributed mutual exclusion," *Source Journal of Parallel and Distributed Computing*, vol.3, pp.107-121, 1996.

[6] MUKESH SINGHAL, "A taxonomy of distributed mutual exclusion," *Journal of parallel and distributed computing*, vol.18, pp.94-101, 1993.

[7] Michel Raynal, A simple taxonomy for distributed mutual exclusion algorithms, *ACM SIGOPS Operating Systems Review*, vol.25, pp.47-50, 1991.

[8] B. R. Badrinath, Arup Acharya, Tomasz Imielinski, "Designing Distributed Algorithms for Mobile Computing Networks," *COMPUTER COMMUNICATIONS*, vol.19, 1992.

[9] Ichiro Suzuki and Tadao Kasami, "A distributed mutual exclusion algorithm," *ACM Transactions on Computer Systems (TOCS)*, Vol.3, pp.344-349, 1985.

[10] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol.21, pp.558-565, 1978.

[11] B. A. Sanders, "The information structure of distributed mutual exclusion algorithms," *ACM Transactions on Computer Systems(TOCS)*, vol. 5, pp.284-299, 1987.

[12] R. Franklin, "On an improved algorithm for decentralized extrema finding in circular configurations of processors," *Communications of the ACM*, vol. 25, pp.336-337, 1982.

[13] M. sinhal, "A heuristically-aided algorithm for mutual exclusion in distributed systems," *IEEE Trans. comput.*, vol.38, pp.5, 1989.

[14] Y. Chang, M. Singhal, M. Liu, "A hybrid algorithm for distributed mutual exclusion," *proc. of comPSAC'90*, 1990.

양 승 일

1996년 2월 호서대학교 정보통신공학과(공학사). 2004년 8월 중앙대학교 컴퓨터소프트웨어학과(공학석사). 2005년 3월~현재 충북대학교 컴퓨터공학과 박사과정 관심분야는 미들웨어, 유비쿼터스, 분산 알고리즘, 모바일컴퓨팅, 상호배제

이 태 규

1992년 2월 군산대학교 전자계산학과(이학사). 1996년 8월 숭실대학교 컴퓨터공학(공학석사). 2006년 2월 고려대학교 컴퓨터학과(이학박사). 2007년 3월~현재 한국산업기술대학교 겸임교수. 관심분야는 분산시스템, 네트워크, 미들웨어, 유비쿼터스, 로봇컴퓨팅

박 성 훈

1982년 2월 고려대학교 정경대학 통계학과(경제학사). 1993년 2월 인디애나대학교 대학원 컴퓨터학과(공학석사). 1997년 2월 고려대학교 컴퓨터학과(공학박사) 2004년 9월~현재 충북대학교 전기전자컴퓨터공학부 교수. 관심분야는 분산, 모바일, 유비쿼터스 시스템, 알고리즘