분산 이동 시스템에서 인과적 메시지 전달을 위한 효율적인 프로토콜

(An Efficient Protocol for Causal Message Delivery in Distributed Mobile Systems)

노 성 주 † 정 광 식 † 이 화 민 †† 유 헌 창 ††† 황 종 선 ††††
(Sung-Ju Roh) (Kwang-Sik Chung) (Hwa-Min Lee) (Heon-Chang Yu) (Jong-Sun Hwang)

요 약 분산 이동 시스템은 단순한 통신 기능에서 작업 흐름 관리, 화상회의, 복제 데이타의 관리, 자원 할당 등의 서비스를 제공하는 시스템으로 급속히 확대·발전하고 있으며, 이러한 서비스를 제공하는 어플리케이션들은 사용자의 요구를 반영하기 위해 메시지를 인과적 순서로 전달해야 한다. 인과적 메시지 전달을 제공하는 기존의 방법들은 많은 피기백(piggyback) 정보로 인한 통신 오버헤드 혹은 어플리케이션으로 전달하는 메시지의 지연, 이동 호스트의 증가에 대한 확장성, 이동 호스트가 계산의 대부분을 수행하는 등의 문제점이 있다. 이 논문은 기지국과 이동 호스트 사이의 종속 정보 행렬을 기지국이 유지하며, 즉각 선행자 메시지(immediate predecessor message)에 대한 종속 정보만을 각 메시지에 피기백하는 방법을 통해 기존 기법의 문제점을 해결하는 효율적인 인과적 메시지 전달 기법을 제안한다. 제안하는 알고리즘은 이전의 알고리즘들과 비교해서 낮은 메시지 오버헤드를 가지며, 메시지를 전달할 때 불필요한 지연(inhibition)을 발생시키지 않는다. 또한 기지국에서 알고리즘의 대부분을 수행하도록 함으로써 이동 호스트의 자원제약과 무선 통신의 낮은 대역폭을 고려하고, 이동 호스트 단위로 인과적 메시지 전달을 이행함으로써 발생하는 처리 지연(processing delay)을 줄여준다.

키워드 : 분산 이동 시스템, 인과적 메시지 전달, 인과적 장벽 벡터

Abstract There is a growing trend in developing system for distributed mobile system that supports services - job flow management, video conference, replicated data management and resource allocation. Supporting these services, applications have to use causally ordered message delivery. Previous proposals that provide causally ordered message delivery have problems such as communication overhead, message delaying, scalability, computing overload of mobile host.

In this paper, we proposed efficient protocol for causally ordered message delivery using the methods that MSS maintains dependency information matrix between MSS and MH, piggybacking dependency information about each immediate predecessor message. Proposed algorithm, when compared with previous proposals, provides a low message overhead, and low probability of unnecessary inhibition in delivering messages. Also, it consider resource restriction of MH and low bandwidth of wireless communication by computing most of algorithm at MSS, and reduce processing delay by executing causally ordered message delivery a unit of MH.

Key words : distributed mobile system, causally ordered message delivery, causal barrier vector

† 비 회 원 : 고려대학교 컴퓨터학과
sjroh@disys.korea.ac.kr
cks@disys.korea.ac.kr
†† 학생회원 : 고려대학교 컴퓨터교육과
zelkova@comedu.korea.ac.kr
††† 정 회 원 : 고려대학교 컴퓨터교육과 교수
yuhc@comedu.korea.ac.kr
†††† 종신회원 : 고려대학교 컴퓨터학과 교수
hwang@disys.korea.ac.kr
논문접수 : 2000년 12월 19일
심사완료 : 2002년 12월 9일

1. 서 론

분산 시스템(distributed system)은 통신 네트워크에 의해서 서로 연결되고, 소프트웨어가 갖추어진 자율적인 고정 처리 단위(fixed processing unit)의 집합이다[1]. 통신 기술의 발전과 컴퓨터의 소형화로 이동 처리 단위(mobile processing unit)의 수는 점차로 증가 추세에 있다. 이러한 추세에 따라 분산 시스템 역시 이동 처리

단위를 포함하는 분산 이동 시스템(distributed mobile system)으로 확대되어 가고 있다. 분산 이동 시스템은 이동 처리 단위인 이동 호스트(MH:mobile host)의 이동성을 지원해 줌으로써 분산 컴퓨팅에 대한 지역적 제약을 제거한다. 그러나 분산 이동 시스템은 이동 호스트의 이동성, 무선 통신의 낮은 대역폭, 이동 호스트의 자원 제약성, 잦은 접속 단절 등과 같은 새로운 문제에 직면하였다[2, 3, 4, 5, 6, 7].

이동 호스트들 사이에는 메시지를 송/수신함으로써 종속 관계(dependency relation)가 발생한다. 복제 데이타의 관리, 자원 할당, 멀티미디어 데이타 제공, 화상 회의, 전자 메일 등의 서비스를 제공하는 다양한 어플리케이션들은 하나의 프로세스 내부 또는 여러 프로세스들 사이에서 발생하는 사건의 종속관계를 고려하여 메시지의 인과적 순서를 맞추도록 요구한다[2, 3, 4, 8, 9, 10, 11, 12, 13, 14, 15]. 또한, 최근에는 기존의 분산 시스템에서 구현된 어플리케이션들은 분산 이동 시스템에서도 구현되도록 요구되고 있다. 특히 분산 이동 시스템에서는 낮은 통신 오버헤드, 이동 호스트의 낮은 계산 능력과 메모리 제약을 고려한 메시지의 인과적 순서화(causal ordering)에 대한 연구가 진행중이다.

기존의 인과적 메시지 순서화 알고리즘들은 이동 호스트의 증가에 대한 확장성이 있으면 이동 호스트에서 계산의 대부분이 수행되거나[2], 메시지 오버헤드가 낮으면 메시지 전달의 불필요한 지연이 발생하고, 지연을 발생시키지 않으면 메시지 오버헤드가 높은 단점이 있다[4]. 이러한 문제점들을 해결하기 위하여, 이 논문에서는 기지국(MSS: mobile support station)과 이동 호스트 사이의 종속 정보를 유지하고, 인과적 장벽 벡터(causal barrier vector)를 사용함으로써 분산 이동 시스템에서 이동 호스트에게 인과적 메시지 전달(causal message delivery)을 이행하는 알고리즘을 제안한다. 제안하는 알고리즘은 이동 환경에서 인과적 메시지 순서화를 위해서 필요한 제어 정보의 크기를 줄이고 지연을 감소 시킨다. 또한 기존의 알고리즘보다 낮은 메시지 오버헤드를 가지며, 알고리즘의 대부분이 기지국에서 수행되기 때문에 이동 호스트의 자원 제약의 문제점을 해결한다.

이 논문의 구성은 다음과 같다. 2 장에서는 인과적 메시지 순서화에 대한 기존 연구와 제안하는 알고리즘의 동기에 대하여 살펴본다. 3 장에서는 분산 이동 시스템의 모델을 설명하고 인과적 메시지 순서화에 대해 정의한다. 4 장에서는 정적 모듈, 핸드오프 모듈, 단절/재접속 모듈에 대하여 기술한다. 5 장에서는 제안하는 알고리즘의 정당성을 증명하고, 6 장에서는 Prakash가 제안한 알고리즘, Alagar가 제안한 두 번째 알고리즘과 이 논문에서 제안하는 알고리즘을 실험을 통하여 성능을 평가한다. 마지막으로 7 장에서는 이 논문의 결론을 맺고 향후 연구 과제에 대하여 기술한다.

2. 관련 연구

분산 시스템에서 인과적 메시지 순서화가 처음으로 구현된 시스템은 ISIS 시스템이다[9]. ISIS 시스템은 메시지를 송신할 때, 수신측 프로세스에게 이전에 보낸 모든 메시지 정보를 피기백한다. 메시지 m이 수신측 프로세스 p_i에게 도착하고 프로세스 p_i로 향하는 과거의 모든 메시지가 이미 프로세스 p_i에게 전달되었다면 m도 프로세스 p_i에게 전달된다. 만약 이전의 메시지들이 전달되지 않았다면, m에 피기백된 이전의 모든 메시지를 전달한 후에 m을 프로세스 p_i로 전달한다.

Schiper는 벡터 클럭을 이용하여 분산 시스템에서 인과적 메시지 순서화를 이행하였다[10]. 이 방법은 ISIS 시스템의 첫 번째 버전과는 달리 각 메시지에 <수신측 사이트, 벡터 시간>으로 구성된 정보가 피기백된다. 각 메시지에는 최대 $n-1$(단, n은 시스템의 전체 프로세스의 수)개의 튜플(tuple)이 피기백될 수 있다. 수신측 프로세스가 메시지 m을 받으면 m의 인과적 선행자 메시지(causal predecessor message)가 수신측 프로세스에게 전달되었는지를 결정하기 위하여 메시지에 피기백된 종속정보를 사용한다. m의 인과적 선행자 메시지가 전달되었다면 m은 수신측 프로세스에게 전달되고 그렇지 않았다면 m은 자신의 인과적 선행자 메시지들이 모두 수신측 프로세스에게 전달될 때까지 수신측 프로세스에 버퍼링된다.

Raynal의 인과적 메시지 순서화 알고리즘[11]은 Schiper의 알고리즘과 유사하다. 그러나 벡터 시간을 사용하는 대신에 각 프로세스 p_i는 $n \times n$ 행렬 $SENT$를 유지한다. $SENT$는 각 프로세스가 모든 다른 프로세스에게 보낸 메시지의 수를 나타낸다. 또 프로세스 p_i가 유지하는 N개의 요소를 가진 정수 벡터 $DELIV$는 p_i가 모든 다른 프로세스로부터 받은 메시지의 수를 나타낸다. 각 메시지에는 $SENT$ 행렬이 피기백 된다. 그러므로 m의 인과적 선행자 메시지가 전달될 때까지 메시지 m은 프로세스 p_j에 버퍼링된다. Raynal의 알고리즘의 메시지 오버헤드는 $SENT$ 행렬이 $N \times N$ 행렬이므로 $O(N^2)$이다.

Alagar는 Raynal의 RST 알고리즘을 기반으로 하여 이동 시스템에서 메시지의 인과적 순서화를 이행하는 3

개의 알고리즘을 제안하였다[4]. 첫 번째 알고리즘에서는 각 메시지에 모든 이동 호스트에 대한 종속정보, 즉 $n_{mh} \times n_{mh}$(단, n_{mh}는 이동 호스트의 수) 행렬을 피기백한다. 이 알고리즘은 높은 통신 오버헤드 때문에 이동 호스트의 증가에 대한 확장성(scalability)을 갖지 못한다. 두 번째 알고리즘은 각 메시지에 이동 시스템의 기지국에 대한 종속 정보, 즉 $n_{mss} \times n_{mss}$(단, n_{mss}는 이동 호스트의 수)행렬을 피기백한다. 그러므로 Raynal의 논문에 비해 확장성은 좋아졌지만 기지국 사이에서 인과적 메시지 순서화를 이행함으로써 불필요한 지연이 발생한다. 세 번째 알고리즘은 두 번째 알고리즘에 발생하는 불필요한 지연을 줄이기 위해서 각 기지국을 k개의 논리적인 단위로 나누어서, 각 메시지에는 $n_{mss} \times k$개의 기지국 사이의 종속 정보, 즉 $(n_{mss} \times k) \times (n_{mss} \times k)$ 행렬이 피기백된다.

Prakash가 제안한 알고리즘은 각 메시지에 즉각 선행자 메시지에 관한 종속 정보만을 피기백한다[2]. 이 알고리즘에서는 이동 호스트에 $n_{mh} \times n_{mh}$ 행렬인 *Delivered*를 유지하고 각 메시지에는 크기 n_{mh}인 인과적 장벽 벡터를 피기백한다. 인과적 장벽 벡터의 요소는 (송신측 프로세스, 송신측 프로세스가 보낸 메시지 수)의 집합이다. 이 알고리즘은 최악의 경우에 메시지 오버헤드가 $O(n_{mh}^2)$이기 때문에 Alagar의 첫 번째 알고리즘과 같은 메시지 오버헤드를 가진다. 그러므로 완전한 확장성을 제공하지 못한다. 또한 이동 호스트에서 알고리즘의 대부분이 수행되기 때문에, 이동 호스트의 전력 사용의 제약성과 무선 통신의 낮은 대역폭이 고려되지 않았다. 이동 호스트 단위로 인과적 메시지 순서화를 이행함으로써 처리시간의 증가로 인하여 긴 지연이 발생하는 단점을 갖는다[4].

기존의 여러 알고리즘들은 분산 이동 시스템에서 이동 호스트 단위로 메시지 순서화를 이행하여 메시지 오버헤드가 커지고 확장성이 결여되었다. 또 기지국 단위로 인과적 메시지 순서화를 이행한다면 <그림 1>의 (b)와 같이 불필요한 지연(inhibition)이 생긴다. <그림 1>의 (a)의 경우, 메시지 m_1을 기지국 MSS_1의 이동 호스트 MH_1에서 기지국 MSS_3의 이동 호스트 MH_3에게 보내고, 메시지 m_2를 이동 호스트 MH_1에서 기지국 MSS_2의 이동 호스트 MH_2로 보내고, 마지막으로 이동 호스트 MH_3으로 이동 호스트 MH_2가 메시지 m_3을 보내는 경우, 메시지 m_3은 기지국 MSS_3에 버퍼링되어 지연된다. 이러한 경우가 발생하는 것은 메시지 m_1과 메시지 m_3처럼 수신측 이동 호스트가 같은 경우이다. <그림 1>의 (b)의 경우와 같이 메시지를 보내는 경우, Alagar의 두 번째 알고리즘처럼 기지국 단위로 인과적 메시지 순서화를 한다면 불필요한 지연이 발생한다.

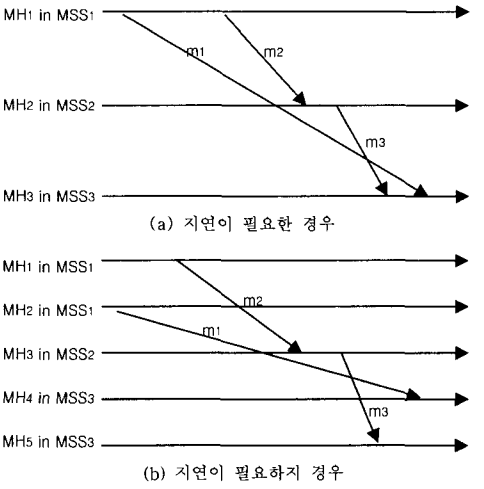


그림 1 메시지 전송 지연이 필요한 경우와 필요하지 않은 경우

이러한 지연을 제거하기 위하여, 이 논문에서는 송신측 기지국과 수신측 이동 호스트 사이의 인과적 메시지 순서화를 이행하는 알고리즘을 제안한다. 그림 1의 (b)는 메시지 m_1과 메시지 m_3의 수신측 기지국은 같지만, 수신측 이동 호스트는 다르기 때문에 지연이 필요하지 않다. 결국 같은 기지국에 있다하더라도 수신측 이동 호스트가 다르면 지연이 발생하지 않는다. 이것은 이동 호스트가 기지국으로부터 받은 메시지의 수를 나타내는 $n_{mss} \times n_{mh}$ 행렬을 이동 호스트가 머무르는 셀의 기지국에 유지함으로써 가능하다. 또한 이동 호스트에 모든 자료를 유지하는 Prakash의 방법과는 다르게, 크기 n_{mh}의 인과적 장벽 벡터를 이동 호스트가 자신이 머무르는 셀의 기지국에 유지하고 각 메시지에 이 벡터를 피기백하여 보내면, 이 메시지를 수신측 기지국은 $n_{mss} \times n_{mh}$ 행렬의 원소와 비교하여 이동 호스트에 전달할 것인지를 결정한다. 이때 이전에 보낸 인과적 종속 관계가 있는 모든 메시지가 수신측 이동 호스트에 전달되지 않았으면, 그 메시지는 수신측 이동 호스트로 전달되지 않고 이전에 보낸 인과적 종속관계에 있는 모든 메시지가 전달될 때까지 수신측 이동 호스트가 머무르는 셀의 기지국에서 지연되기 때문에 인과적 장벽 벡터(causal barrier vector)라고 한다[2].

3. 시스템 모델

분산 이동 시스템은 <그림 2>와 같이 유/무선 네트워크와 이동 호스트, 기지국, 고정 처리 단위인 고정 호스트로 구성된다. 이동 호스트는 네트워크에 접속된 도중에도 이동할 수 있으며, 이동 호스트의 종류로는 셀룰러 폰이나 무선 모뎀을 장착한 랩톱 컴퓨터, 휴대용 정보 단말기(PDA: Personal Digital Assistants) 등이 있다. 이동 호스트가 다른 이동 호스트나 고정 호스트와 통신을 하기 위해서는 반드시 자신이 머무르는 셀(cell)의 기지국과 무선 통신을 해야 한다. 하나의 셀은 기지국이 관리하는 논리적 또는 지리적 영역이다.

모든 기지국은 자신이 관리하는 셀에 정기적으로 자신의 식별자를 포함하는 탐지 신호(beacon signal)를 방송한다. 이동 호스트는 이 탐지 신호를 받자마자 자신이 통신해야 하는 기지국을 결정한다. 만약 이동 호스트가 두 개의 셀의 경계지역에 있으면 동시에 두 개의 기지국의 탐지 신호를 받지만, 이동 호스트는 탐지 신호의 품질을 측정하여 단지 하나의 기지국을 선택한다고 가정한다.

기지국이 관리하는 영역 내의 무선 통신은, 즉 기지국과 이동 호스트 사이의 통신은 FIFO(First-In First-Out)를 가정한다. 그리고 유선 네트워크에서 이루어지는 유선 통신은 FIFO를 가정하지 않는다. 하나의 이동 호스트에서는 하나의 프로세스가 실행된다고 가정하며, 프로세스를 이동 호스트로 표기한다.

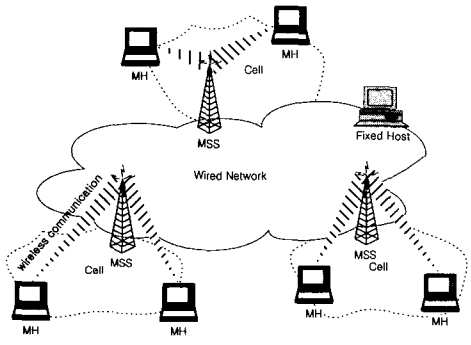


그림 2 분산 이동 시스템의 구조

분산 이동 시스템은 전역 시계(global clock)를 가지고 있지 않기 때문에, 서로 다른 이동 호스트 사이에서 발생하는 사건들의 순서는 절대적인 시간에 근거하여 결정할 수 없다. 두 사건이 논리적인 시간 순서가 정해지면 서로 종속 관계가 있다고 말한다. 종속 관계는 Lamport의 전후 관계(happened-before), 즉 $\rightarrow$를 사용해서 표현될 수 있다[16]. 메시지 m을 보내는 사건을 *Send(m)*, 메시지 m을 받는 사건을 *Deliver(m)*이라고 하면 사건 사이의 전후 관계는 다음과 같이 정의된다:

- 같은 호스트에서 발생하는 사건 e와 e'에 대하여, e 다음에 e'가 발생하면, $e \rightarrow e'$이다.
- 같은 메시지 m에 대하여 *Send(m)*을 e, *Deliver(m)*을 e'라고 하면 $e \rightarrow e'$이다.
- 사건 e, e', e''에 대하여, $e \rightarrow e''$이고 $e'' \rightarrow e'$이면 $e \rightarrow e'$이다.

두 개의 사건 사이에 종속관계가 없다면 상호 병행적(mutually concurrent)이라고 말한다[2, 7]. 즉, $e \nrightarrow e'$이고 $e' \nrightarrow e$이면 e와 e'는 병행적이라고 말한다. 모든 사건이 전 순서화(total ordering)되도록 분산 어플리케이션을 실행하면 비용이 매우 비싸며, 많은 지연이 발생하여 프로세스간의 병행성(concurrency)이 낮다. 그러므로 전 순서화보다 제약 사항을 완화한 인과적 메시지 순서화를 이행하면 프로세스 사이의 병행성이 높아진다. 주어진 시스템에서 공동의 작업을 수행하는 프로세스 사이에서 메시지를 교환할 때, 메시지간의 인과적 순서화는 다음과 같이 정의한다.

[정의 1] (인과적 순서화)
두 개의 메시지 m_1과 m_2가 같은 수신측 이동 호스트로 향하고 $send(m_1) \rightarrow send(m_2)$ 이면, $Deliver(m_1) \rightarrow Deliver(m_2)$이다. ■

<그림 3>과 같이, $Send(m_1) \rightarrow Send(m_2)$이고 $Send(m_2) \rightarrow Send(m_3)$이면, Lamport의 전후관계에 의해서 $Send(m_1) \rightarrow Send(m_3)$이다. 또 $Send(m_1) \rightarrow Send(m_3)$는 메시지를 송신측 이동 호스트는 다르지만 수신측 이동 호스트는 같으므로 인과적 메시지 순서화를 이행하려면 $Deliver(m_1) \rightarrow Deliver(m_3)$이어야 한다. 그러나 <그림 3>은 m_3이 m_1보다 먼저 수신되어서 인과적 순서를 위반한다.

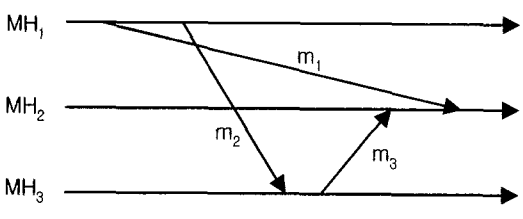


그림 3 인과적 메시지 전달을 위반하는 예

4. 인과적 메시지 전달 프로토콜

4.1 자료 구조

기지국은 s, 이동 호스트는 h에 아래첨자를 붙여 표기한다. 인과적 순서로 메시지를 전달하기 위해서 각 기지국 s_i가 유지하는 자료구조는 다음과 같다.

• $SSENT_i$: 자신이 보낸 메시지의 수를 의미하는 단조 증가 정수, $SSENT_i$를 유지한다. $SSENT_i$는 기지국 s_i가 메시지를 하나 보낼 때마다 1씩 증가한다. 단, 메시지 m에 피기백된 $SSENT_m$은 메시지 m을 보낸 기지국이 보낸 메시지의 수를 나타낸다.

• $STohCB_i$: $STohCB_i$는 메시지에 대한 전달 제약조건을 나타내는 직접 종속정보(direct dependency information)이다. 기지국 s_i는 벡터 $STohCB_i$를 유지한다. 벡터 $STohCB_i$의 각 요소는 (기지국 아이디, 기지국이 보낸 메시지의 수), 즉 $(SrcMSSID, SSENT_i)$의 집합으로 구성된다. 단 메시지 m에 피기백된 인과적 장벽 벡터 $STohCB_m$에 대하여, $(k, x) \in STohCB_m[j]$이면, 기지국 s_k가 보낸 메시지 번호가 x인 메시지를 이동 호스트 h_j가 받은 후에 메시지 m이 이동 호스트 h_j에게 전달될 수 있음을 의미한다. 이와 같이 메시지 m에 피기백된 $STohCB_m$이 만족될 때까지 메시지 m의 전달은 지연되므로 $STohCB_i$를 인과적 장벽 벡터라고 한다. 각 메시지에는 $STohCB_i$ 벡터가 피기백되고, 시작시에 공집합으로 초기화된다.

• $STohDeliver_i$: $n_{mss} \times n_{mh}$행렬로 $STohDeliver_i[j,]$는 기지국 s_j가 각 이동 호스트에게 보낸 메시지의 수를 나타낸다. 예를 들면, $STohDeliver_i[j,k]=x$이면 기지국 s_j로부터 이동 호스트 h_k로 메시지 번호가 x보다 작거나 같은 메시지는 모두 전달되었음을 기지국 s_i가 알고 있음을 의미한다. 처음에 $STohDeliver_i$의 모든 요소는 0으로 초기화된다.

• $WaitAck_i$: $STohCB_i$와 $STohDeliver_i$외에도 기지국 s_i는 메시지 큐인 $Suspend_i$와 $WaitAck_i$를 유지한다. 현재 수신된 메시지가 인과적 순서를 위반하지 않는다면 $WaitAck_i$ 메시지 큐에 그 메시지에 대한 응답이 수신측 이동 호스트로부터 올 때까지 일시적으로 저장하고 응답이 오면 그 메시지를 삭제한다.

• $Suspend_i$: 현재 수신된 메시지가 인과적 순서를 위반하면, $Suspend_i$ 큐에 일시적으로 저장된다. $Suspend_i$에 저장된 메시지들은 $WaitAck_i$ 큐에 저장된 메시지들에 대한 응답이 올 때마다, 인과적 순서를 위반하는지를 재검사하여 인과적 순서를 위반하지 않는 메시지는 이동 호스트에게 전달하고 $Suspend_i$ 큐에서 삭제한다.

4.2 프로토콜

인과적 메시지 순서화를 위해 제안하는 프로토콜은 정적 알고리즘, 핸드오프 알고리즘, 단절/재접속 알고리즘으로 나뉜다. 먼저 정적 알고리즘은 이동 호스트가 셀을 이동하지 않을 때 메시지를 주고받는 것을 다루는 모듈이고, 핸드오프 알고리즘은 이동 호스트가 셀을 이동할 때, 그 이동 호스트에 대한 책임을 바꾸는 모듈이며, 마지막으로 단절/재접속 알고리즘은 이동 호스트의 일시적인 단절과 재접속을 다루는 모듈이다.

4.2.1 정적 알고리즘(static algorithm)

A. 메시지 송신 사건

기지국 s_i가 메시지 m을 수신측 이동 호스트 h_j에게 보내는 알고리즘은 다음과 같다.

---

1. $SSENT_i = SSENT_i + 1$;
2. $Send(m, SrcMSSID, SSENT_i, STohCB_i)$ to h_j;
3. $STohCB_i[j] = (SrcMSSID, SSENT_i)$;

기지국 s_i가 이동 호스트 h_j에게 메시지를 보낼 때 자신이 송신한 메시지의 수를 의미하는 $SSENT_i$를 1 증가시키고(단계 1), 수신측 프로세스에게 메시지 식별자(ID)를 의미하는 $(SrcMSSID, SSENT_i)$와 $STohCB_i$를 피기백하여 보낸다(단계 2). $STohCB_i[j]$를 보낸 메시지의 식별자로 치환한다(단계 3).

B. 메시지 수신 사건

기지국 s_j로부터 메시지 m을 이동 호스트 h_i가 수신하는 알고리즘에서, h_i의 지역 기지국 s_i가 하는 일은 다음과 같다.

---

1. MSS s_i receives $(m, SrcMSSID, SSENT_m, STohCB_m)$
2. if $STohCB_m[i] \neq \phi$
   then
   for $\forall s_k :: (k,x) \in STohCB_m[i]$:
   if $STohDeliver_i[k,i] < x$
   then
   do *append* $(m, SrcMSSID, SSENT_m, STohCB_m)$ to $Suspend_i$ od;
   else
   do
   $Send(m)$ to h_i;

```
        append (m, SrcMSSID, SSENT_m,
                              STOHCB_m) to WaitAck_i;
        STOHDeliver_i[j,i] = SSENT_m;
      od
    fi
  else
    do
      Send(m) to h_i;
      append(m,SrcMSSID,SSENT_m, STOHCB_m)
                                    to WaitAck_i;
      STOHDeliver_i[j,i] =SSENT_m;
    od
  fi
3. When s_i receives ack(m) from h_i, it takes the
   following actions        in sequence
3.1 remove (m, STOHCB_m) from WaitAck_i;
3.2 for ∀k:
      if(k, y)∈STOHCB_m[i]
      then
        do STOHDeliver_i[k,j] =
                  max(STOHDeliver_i[k,j], y) od
      fi
3.3 STOHCB_i[i] = DiffMax(UnionMax(STOHCB_i[i],
            (SrcMHID,SSENT_m)),STOHCB_m[i])
3.4 for ∀k ∉ {s_i, s_j}:
      do STOHCB_i[k] =UnionMax(STOHCB_i[k],
                                  STOHCB_m[k]) od
3.5 STOHCB_i[j] = DiffMax(STOHCB_i[j],
                                  STOHCB_m[j])
3.6 for ∀k ≠ i:
      for ∀(l,x) ∈ STOHCB_i[k]:
        if STOHDeliver_i[l,k] ≥x
        then
          do remove (l,x) from STOHCB_i[k] od
        fi
3.7 if Suspend_i ≠ Φ
      then
        for ∀ h_k::(k,x)∈STOHCBSuspend_i[i]:
          if STOHDeliver_i[k,i] ≥x
          then
            do
              Send(m) to h_i;
              append (m,   SrcMSSID,SSENT_m,
STOHCB_m) to WaitAck_i;
              STOHDeliver_i[j,i] =SSENT_m;
              goto step 3;
            od
          fi
    fi
```

위의 알고리즘에서 *UnionMax* 함수는 송신측 기지국과 수신측 이동 호스트의 인과적 메시지 전달 제약 조건의 합집합을 구하여 반환한다. *DiffMax* 함수는 현재의 제약 조건으로부터 이미 만족되어 있는 제약 조건을 삭제한다.

기지국 s_j로부터 메시지를 수신한 이동 호스트 h_i의 지역 기지국 s_i는 인과적 장벽 벡터를 참고하여 수신된 메시지가 전달 가능한지 검사한 후에 전달 가능하지 않으면 $Suspend_i$에 추가하고, 전달 가능하면 메시지를 관련 이동 호스트에게 보낸 후에 $WaitAck_i$에 추가한다(단계 2). 메시지를 받은 이동 호스트로부터 *ack*를 받으면 $WaitAck_i$로부터 관련 메시지를 제거하고(단계 3.1) $STOHCB_m[i]$를 이용하여 $STOHDeliver_i$를 갱신한다(단계 3.2). 이 메시지 m을 받은 후에 s_i가 보내는 메시지들은 메시지 m에 인과적으로 종속된다. 즉각 종속정보인 $(j, SSENT_m)$을 추가하고, 또 이전의 전이적 종속정보를 삭제하여 $STOHCB_i[i]$를 갱신한다(단계 3.3).

수신측 이동 호스트가 아닌 경우에 해당하는 인과적 장벽에 대하여도 $STOHCB_m$를 이용하여 *UnionMax* 연산을 사용하여 불필요한 제약 조건을 제거하여 준다. 즉 가장 최근의 상호 병행적인 정보만을 포함하도록 한다(단계 3.4). $STOHCB_m$는 이동 호스트 h_j에게 이미 전달된 메시지들을 나타내므로 *DiffMax* 연산을 사용하여 $STOHCB_i[j]$를 갱신하여 준다(단계 3.5). 또 불필요한 정보를 $STOHCB_i$로부터 제거하여 주는 쓰레기 수집(garbage collection)을 한다(단계 3.6). 마지막으로 $Suspend_i$에 있는 항목 중에서 전달 가능한 메시지가 있는지 검사하여 전달 가능한 메시지 있으면 메시지 수신 사건의 단계 3부터 다시 수행한다(단계 3.7).

4.2.2 핸드오프 알고리즘(handoff algorithm)

이동 호스트 h_i가 기지국 s_i로부터 기지국 s_j로 이동한다고 가정하자. 먼저 이동 호스트 h_i가 기지국 s_j의 셀로 들어가서 기지국 s_j가 보내는 전파를 감지하여 *register* $MH(h_i, s_i)$ 메시지를 기지국 s_j에게 보낸다. 이 메시지는 이동 호스트 h_i가 기지국 s_j에게 자신의 도착을 알리는 역할을 한다. 이동 호스트 h_i로부터 $registerMH(h_i, s_i)$ 메시지를 받은 기지국 s_j는 이전의 기지국 s_i에게 $migratedMH(h_i)$ 메시지를 보낸다. 이전의 기지국 s_i는 $migratedMH(h_i)$ 메시지를 받자마자 인과적 메시지 순서화를 이행하기 위하여 이동 호스트 h_i와 관련된 정보인 $STOHCB_i$, $STOHDeliver_i$, $Suspend_i$, $WaitAck_i$를 기지국 s_j에게 보내준다. 이러한 정보를 받은 기지국 s_j는 이동 호스트 h_i의 관련 자료들을 갱신하고 새로 만든다. 만약 핸드오프가 끝나기 전에 기지국 s_i에게 보내진

메시지는 기지국 s_j에게 전파한다.

4.2.3 단절/재접속 알고리즘(disconnection/reconnection algorithm)

이동 호스트 h_i가 기지국 s_i가 담당하는 셀에 머무르는 동안이나, 또는 이동 호스트 h_i가 다른 셀로 이동하기 전에 네트워크로부터 단절되었다면, 이동 호스트 h_i가 단절된 동안에도 기지국 s_i가 인과적 메시지 순서화를 유지하는 역할을 수행한다. 이때 이동 호스트 h_i로 향하는 모든 메시지는 이동 호스트 h_i가 여전히 기지국 s_i에 연결된 것처럼 처리된다. 나중에 이동 호스트 h_i가 재연결된 후에 $WaitAck_i$에 저장된 메시지들은 이동 호스트 h_i에게 보내질 수 있다. 이동 호스트 h_i가 다른 기지국 s_j가 관리하는 셀로 이동했을 때, 기지국 s_i와 기지국 s_j사이에 핸드오프 절차가 끝난 후에 메시지를 받거나 보내도록 처리되어야 한다. 이동 호스트 h_i가 영구적으로 네트워크로부터 단절되었다면 이동 호스트 h_i를 마지막으로 담당하는 기지국은 이동 호스트 h_i에 관련된 모든 자료를 삭제한다. 그리고 이동 호스트 h_i가 삭제되었다는 메시지를 다른 기지국들에게 보내는 메시지에 피기백해서 비동기적으로 전파한다. 새로운 이동 호스트가 기지국 s_i에 추가된 경우에, 기지국 s_i는 $STOHCB_i$, $STOHDeliver_i$, $Suspend_i$, $WaitAck_i$를 새로 만든다. 또한 이동 호스트 h_i가 영구적으로 단절된 경우와 같은 방법으로, 다른 모든 기지국들에게 이동 호스트의 연결 정보를 전파한다.

5. 정당성 증명

[보조정리 1]과 [보조정리 2]는 인과적 장벽 벡터가 인과적 순서 관계의 메시지들 사이의 정보를 유지한다는 것을 보장한다. [보조정리 1]은 (k, x)$\in STOHCB_m$[j]이면 m_x와 m 사이에는 반드시 인과적 종속 관계가 있다는 것을 증명하고, [보조정리 2]는 (k, x)를 $STOHCB_m$으로부터 미리 제거하지 않기 때문에 인과적 장벽 벡터 $STOHCB_m$은 필요한 정보를 포함한다는 것을 보여준다. [정리 1]은 이 두 가지 보조정리를 이용해서 모든 메시지는 인과적 순서를 위반하지 않고 수신측 이동 호스트에게 전달된다 것을 증명한다. 이동 호스트 h_i, h_j, h_k, h_l이 머무르는 셀의 기지국을 각각 s_i, s_j, s_k, s_l이라 하자. 그리고 이행적 폐쇄(transitive closure)는 $\rightarrow^*$로 표기한다.

[보조정리 1]

기지국 s_i가 보내는 메시지 m에 (k, x)$\in STOHCB_m$[j]인 정보가 피기백된다면 $Send(m_x) \rightarrow Send(m)$이고 기지국 s_k로부터 이동 호스트 h_j로 보낸 메시지 m_x가 존재한다.

증명) 두 가지 경우가 존재한다. 첫째로 $k=i$일 때, $STOHCB_i$[j]는 메시지 송신 사건의 단계 3에서 갱신된다. 메시지 m_x를 기지국 s_i가 이동 호스트 h_j로 보낸 후에 기지국 s_i는 $STOHCB_i$[j]를 (i, x)로 치환한다. 메시지 m을 보낼 때 (i, x)$\in STOHCB_i$[j] 라면, 기지국 s_i는 반드시 메시지 m_x를 보낸 후에 메시지 m을 보낸다. 즉, $Send(m_x) \rightarrow Send(m)$이다[그림 4의 (a)].

(a) k=i *(b)k≠1*

그림 4 [보조 정리 1]의 예

둘째로 k≠i일 때, 이동 호스트 hi가 메시지 m′를 받으면, 기지국 si는 메시지 수신 사건의 단계 3.4에서 SToHCBi[j]에 (k, x)를 추가하므로 (k, x)$\in$ SToHCBm′[j]이다. 즉, 기지국 sk에서 이동 호스트 hi로의 인과적 종속정보 사슬DC1[그림 4의 (b)의 점선인 DC1]인 Sendsk(mx)→＊Deliverhi(m′)에 의해서 기지국 si는 (k, x)를 SToHCBi[j]에 추가시킨다. 그러므로 (k, x)$\in$SToHCBi[j]라면, 기지국 si는 메시지 m′가 이동 호스트 hi에게 전달된 후에 메시지 m을 보내기 때문에, Send(mx)→Send(m)이다[그림 4의 (b)]. ■

[보조정리 2]

기지국 s_i가 보낸 메시지 m과 이동 호스트 h_j에게 보내진 메시지 m_x가 다음의 3가지 조건을 만족하고,

(i) $Send(m_x) \rightarrow Send(m)$

(ii) $Deliver_{h_j}(m_x) \nrightarrow Send(m)$

(iii) $Send(m)$을 하기 전에 이동 호스트 h_j에게 마지막으로 보낸 메시지가 m_x일 경우, 이동 호스트 h_j에게 보낸 메시지 m_y는 존재하지 않는다.

기지국 s_i가 보낸 메시지 m과 임의의 기지국 s_k가 이동 호스트 h_j에게 보낸 메시지 m_x가 존재하면, (k, x)$\in STOHCB_m$[j]이다.

증명) 메시지 m_x를 보낼 때 (k, x)에 관련된 정보, 즉 $(SrcMSSID,\ SSENT_m)$이 메시지 m_x에 피기백된다. $Send(m_x)$의 인과적 후행자인 메시지 송신은 다른 이동 호스트들에게 (k, x)를 전파한다. 만약 $Send(m_x) \rightarrow^* Send(m)$의 종속정보 사슬이 존재한다면 (k, x)는 s_i로

의 사슬을 따라서 전파된다. s_i가 m을 보내기 이전의 사슬을 통해서 (k, x)를 받았다면 (k, x)는 $STOHCB_m$[j]에 추가된다. 그러므로 (k, x)는 $STOHCB_m$[j]의 요소가 된다. 그러나 $Send(m_x) \rightarrow^* Send(m)$의 종속정보 사슬인 인과적 장벽 벡터로부터 (k, x)가 삭제되는 두 가지의 경우가 있는데, 이는 메시지를 보낼 때와 메시지를 받을 때이다.

• 메시지를 보낼 때

a) 기지국 s_k가 이동 호스트 h_j에게 메시지 m_x를 보낸 후에, 메시지 m_y를 보냈을 때, 기지국 s_k는 메시지 m_y를 보낸 후에 (k, x)를 $STOHCB_k$[j]에서 삭제하고 다른 요소로 치환한다(메시지 송신 사건의 단계 3). 기지국 s_i가 메시지 m을 보낼 때 기지국 s_k가 이동 호스트 h_j에게 메시지 보낸 메시지중 메시지 m_x가 마지막 메시지라고 알고 있기 때문에, $Send(m_y) \nrightarrow Send(m)$이다. 그러므로 $Send(m_y)$는 이동 호스트 h_j로 피기백되는 종속정보 사슬에 존재하지 않는다. 이와 같이 그림 5 (a)에서 DC_1만 존재하고 DC_2는 존재하지 않는 경우에, 기지국 s_i가 메시지 m을 보낼 때 (k, x)$\in STOHCB_m$[j]이다. 그림 5 (a)의 DC_2가 존재하면 [보조정리 2]의 가정 (iii)을 위반한다.

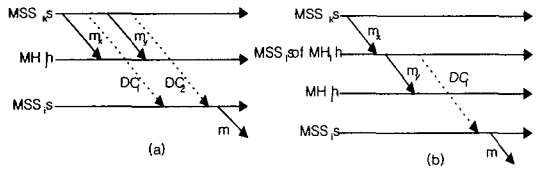


그림 5 메시지를 송신하는 동안 (k, x)가 삭제되는 경우

b) $Send(m_x) \rightarrow^* Send(m)$의 종속 정보 사슬 내에서 기지국 s_l의 지역 사건이 다음의 순서로 발생한다. 메시지 m_x를 이동 호스트 h_l에게 전달한 후에, 기지국 s_l은 메시지 m_y를 이동 호스트 h_j에게 보낸다. 기지국 s_l은 메시지 m_y를 보낸 후에 $STOHCB_l$[j]로부터 (k, x)를 삭제한다(메시지 송신 사건의 단계 3). 그러나 $Send(m_x) \rightarrow Deliver_{h_l}(m_x) \rightarrow Send(m_y) \rightarrow Send(m)$이고, 기지국 s_l에서 기지국 s_i로의 종속정보 사슬 DC_1[그림 5의 (b)의 종속정보 사슬 DC_1]이 존재하면, 메시지 m_x를 이동 호스트 h_j에게 보내는 사건이 $Send(m)$을 앞서는 것 중 가장 마지막 사건이라는 [보조정리 2]의 가정(iii)을 위반한다. 결국 $Send(m_x) \rightarrow Deliver_{h_l}(m_x) \rightarrow Send(m_y) \rightarrow Send(m)$은 발생하지 않는다. 그러므로 (k, x)는 이 사슬에서 삭제되지 않는다[그림 5의 (b)].

• 메시지를 전달할 때

$Send(m_x) \rightarrow^* Send(m)$ 종속정보 사슬 내에서 이동 호스트 h_l이 머무르는 셀의 기지국인 s_l이 (k, x)를 삭제하는 경우는 메시지 수신 사건의 단계 3.3, 3.4, 3.5에서 발생한다.

a) 이동 호스트 h_l이 머무르는 셀의 기지국인 s_l은 메시지 수신 사건의 단계 3.3에서 $STOHCB_l$[j]의 원소 (k, x)를 삭제할 수 있다. 기지국 s_k가 메시지 m_x를 이동 호스트 h_j에게 보낸 후에, 기지국 s_k에서 이동 호스트 h_l로의 종속정보 사슬 DC_1(즉, 그림 6의 (a)의 종속정보 사슬 DC_1)에 의해서 기지국 s_l은 (k, x)를 $STOHCB_l$[j]에 추가시킨다. DC_1이 기지국 s_l에게 (k, $SSENT_{m''}$)를 전파하기 전에 기지국 s_k는 이동 호스트 h_j에게 메시지 m''를 보낸다. 그림 6 (a)의 종속정보 사슬 DC_2에서 이동 호스트 h_l에게 가장 마지막 전달된 메시지는 m'이다. 이동 호스트 h_l에게 메시지 m'를 전달 하자마자 (k, x)는 기지국 s_l에 의해서 $STOHCB_l$[j]로부터 삭제되고 메시지 m''와 관련된 종속정보로 치환된다. 또한 기지국 s_l에서 이동 호스트 h_i로의 종속정보 사슬 DC_3이 시작하기 전에 메시지 m'는 이동 호스트 h_l에 전달된다. 그리고 기지국 s_l에서 기지국 s_i로의 종속정보 사슬 DC_3이 존재하므로 $Send(m_x) \rightarrow Send_{s_k}(m'') \rightarrow Deliver_{h_l}(m') \rightarrow Send(m)$이고, 메시지 m''의 수신측 이동 호스트는 h_j이므로 [보조정리 2]의 가정 (iii)을 위반한다[그림 6의 (a)].

b) 이동 호스트 h_l이 머무르는 셀의 기지국인 s_l은 메시지 수신 사건 단계 3.4에서 $STOHCB_l$[j]의 원소 (k, x)를 삭제할 수 있다. 기지국 s_k가 메시지 m_x를 이동 호스트 h_j에게 보낸 후에, 기지국 s_k에서 이동 호스트 h_l로의 종속정보 사슬 DC_1(즉, 그림 6의 (b)의 종속정보 사슬 DC_1)이 존재하면, 기지국 s_l은 DC_1에 의해서 $STOHCB_l$[j]에 (k, x)를 추가시킨다. 메시지 m_x가 이동 호스트 h_j에게 전달된 후에, 이동 호스트 h_j가 메시지 m'를 이동 호스트 h_l에게 보내면, (k, x)$\in STOHCB_{m'}$[j]가 된다. 메시지 m'가 이동 호스트 h_l에게 전달될 때, 이동 호스트 s_l에 의해서 (k, x)는 $STOHCB_l$[j]에서 삭제된다. 그리고 이동 호스트 s_l에서 이동 호스트 s_i로의 종속정보 사슬 DC_2(즉, 그림 6의 (b)의 종속정보 사슬 DC_2)가 존재하므로 $Deliver_{h_j}(m_x) \rightarrow Send(m)$이다. $Deliver_{h_j}(m_x) \rightarrow Send(m)$은 [보조정리 2]의 가정 (ii)를 위반한다[그림 6의 (b)].

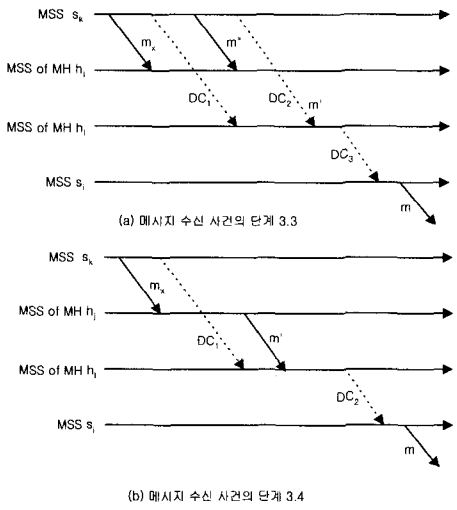


그림 6 메시지를 수신하는 동안 (k, x)가 삭제되는 경우

c) 이동 호스트 h_l이 머무르는 셀의 기지국인 s_l이 메시지 수신 사건의 단계 3.5에서 $STOHCB_l$[j]로부터 (k, x)를 삭제한다면, 기지국 s_l은 종속정보 사슬에 의해 메시지 m_x가 이동 호스트 h_j에게 전달되었다는 것을 안다. 이것은 $Deliver h_j(m_x) \rightarrow Send(m)$을 의미하므로 [보조정리 2]의 가정 (ii)를 위반한다.

위의 a), b), c)에 의하여 (k, x)를 $STOHCB_l$[j]로부터 삭제한 후에, 기지국 s_l의 $Send(m_y)$ 사건이 $Send(m_x) \rightarrow {}^*Send(m)$ 종속정보 사슬에 있다면, [보조정리 2]의 가정 (ii), (iii)중에 하나를 위반한다. 그러므로 모순에 의해서 임의의 기지국 s_k가 이동 호스트 h_j에게 보낸 메시지 m_x와 기지국 s_i가 보낸 메시지 m이 존재하면, (k, x)$\in STOHCB_m$[j]이다. ■

[정리 1]

제안하는 알고리즘은 인과적 순서화를 보장한다.

증명) 메시지 m_x와 메시지 m의 수신측 이동 호스트는 h_j이고 메시지 m_x는 기지국 s_k가 이동 호스트 h_j에게 보낸 마지막 메시지이며, $Deliver h_j(m_x) \nrightarrow Send(m)$이라 하자. [보조정리 1]과 [보조정리 2]에 의하여 메시지 m에 피기백되는 $STOHCB_m$[j]는 (k, x)를 원소로 갖는다. 즉, $Send(m_x) \rightarrow Send(m)$이고 수신측 이동 호스트가 h_j이며, $Send(m_x) \rightarrow Send(m_y) \rightarrow Send(m)$인 메시지 m_y가 존재하지 않는다면 (k, x)$\in STOHCB_m$[j]인 $STOHCB_m$[j]가 메시지 m에 피기백 된다. 메시지 수신 사건의 단계 2는 메시지 m_x가 이동 호스트 h_j에게 전달된 후에 메시지 m이 이동 호스트 h_j에게 전달되도록 한다. 그러므로 인과적 선행자 메시지와 즉각 후행자 메시지(immediate successor message) 쌍간의 인과적 메시지 순서화가 이루어진다. 인과적 종속 관계의 이행성(transitivity)은 분산 이동 시스템에서 모든 메시지들에 대한 인과적 순서화가 이루어지도록 보장한다. ■

6. 성능 평가 및 분석

이 장에서는 인과적 메시지 순서화를 제공하는 알고리즘인 Prakash의 알고리즘[2](이하 PCMD: Prakash's Causal Message Delivery algorithm)과 Alagar의 두 번째 알고리즘[4](이하 ACMD: Alagar's Causal Message Delivery algorithm)을 이 논문에서 제안하는 알고리즘(이하 RCMD: Roh's Causal Message Delivery algorithm)과 함께 실험을 통해서 성능을 평가한다. 6.1절에서는 실험 환경에 관하여 설명하고, 6.2 절에서는 3가지 알고리즘의 실험결과를 비교 분석한다.

6.1 실험 환경

이 논문에서는 UCLA 병렬 컴퓨팅 연구실에서 만든 PARSEC 시뮬레이터를 이용하여 실험을 수행하였다. PARSEC은 C++ 언어를 기반으로 만들어진 메시지 교환 실험 언어로 Maisie[11] 실험 언어에서 파생되어 만들어졌고, 이산 사건 실험 모델(discrete-event simulation model)을 사용한다.

실험을 하고자 하는 분산 이동 시스템은 3 장에서와 같이 이동 호스트의 총 수가 n_{mh}이고 기지국의 총수는 n_{mss}이다. 각 이동 호스트에 하나의 프로세스가 수행된다고 가정한다. 이 프로세스들 사이에는 메시지만을 교환함으로써 통신한다. 하나의 프로세스에서 연속적인 메시지 생성 시간 간격은 지수 분포를 따른다. 또 모든 이동 호스트의 생성이 완료된 상태에서 메시지 교환이 시작되고 각 이동 호스트는 기지국에게 임의로 배정되고, 모든 기지국이 관리하는 이동 호스트의 수는 동일한 것으로 가정한다.

알고리즘에 대한 실험 평가 요소로는 이동 호스트 수의 증가에 따라 메시지에 피기백되는 인과적 장벽 벡터에 포함되는 튜플의 수와 메시지 생성 시간 간격에 따른 메시지에 피기백되는 인과적 장벽 벡터에 포함되는 평균 튜플의 수, 그리고 메시지 전달의 지연율이다.

먼저, PCMD 알고리즘에서 n_{mh}의 증가에 따른 평균 튜플의 수와 RCMD 알고리즘에서 n_{mh}의 증가에 따른 평균 튜플의 수를 비교한다. 두 번째로 메시지의 생성 시간 간격의 증가에 따른 평균 튜플의 수는, 메시지를 이동 호스트에 전달하기 위한 조건을 만족하는 시간적

여유가 많아져서 인과적 순서로 메시지를 전달하는 시스템의 중요한 요소이므로 PCMD 알고리즘과 RCMD 알고리즘을 비교한다. 세 번째로 이동 분산 시스템에서 기지국의 수가 고정된 상태에서 이동 호스트의 수의 증가에 대해 ACMD 알고리즘과 RCMD 알고리즘을 비교한다. 실제로 현재 구현된 시스템에서도 기지국의 수는 고정되어 있고 이동 호스트의 수는 증가하는 추세이다. 따라서 이동 호스트의 수의 증가에 따른 지연율의 측정은 필요하다. 실험 결과는 모든 실험 평가 요소에 대하여 10번 수행한 결과의 평균값을 취하였다.

6.2 실험 결과 및 분석

이 실험을 통해서 PCMD 알고리즘과 ACMD 알고리즘, 그리고 이 논문에서 제안한 RCMD 알고리즘의 성능을 비교한다.

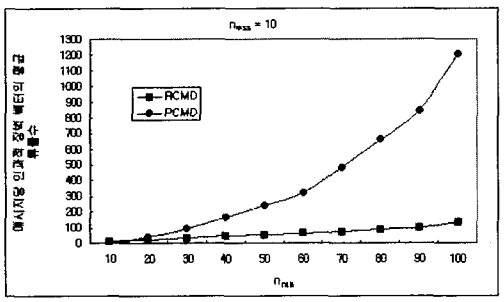


그림 7 n_{mh}의 증가에 대해 각 메시지에 피기백되는 인과적 장벽 벡터의 평균 튜플수

그림 7은 n_{mh}의 증가에 따라서 각 메시지에 피기백되는 인과적 장벽 벡터에 포함되는 RCMD 알고리즘의 평균 튜플의 수와 PCMD 알고리즘의 평균 튜플의 수를 비교한 것이다. 그림 7은 n_{mss}가 10일 때, n_{mh}가 증가함에 따라서 각 메시지에 피기백되는 인과적 장벽 벡터의 튜플율이 PCMD 알고리즘은 최소 $0.12\times n_{mh}\times n_{mh}$에서 최대 $0.14\times n_{mh}\times n_{mh}$이고, RCMD 알고리즘은 최소 $0.11\times n_{mss}\times n_{mh}$에서 최대 $0.13\times n_{mss}\times n_{mh}$ 임을 보여준다. 실험 결과에서 튜플의 수를 계산하면 n_{mh}가 100이고 n_{mss}가 10인 경우에 PCMD 알고리즘은 인과적 장벽 벡터가 약 1207 개의 튜플을 포함하고, RCMD 알고리즘은 인과적 장벽 벡터가 약 132 개의 튜플을 포함한다. 그러므로 RCMD 알고리즘은 PCMD 알고리즘보다 메시지 오버헤드가 낮다. 이처럼 RCMD 알고리즘이 PCMD 알고리즘보다 낮은 메시지를 오버헤드를 발생시키는 이유는 기지국과 이동 호스트 사이에 인과적 순서화를 이행하기 때문이다.

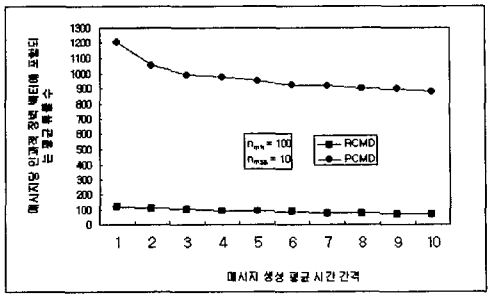


그림 8 메시지 생성 시간 간격의 증가에 대해 각 메시지에 피기백되는 인과적 장벽 벡터의 평균 튜플수

그림 8은 메시지 생성 시간 간격의 증가에 대해 각 메시지에 피기백되는 인과적 장벽 벡터에 포함되는 튜플의 수를 RCMD 알고리즘과 PCMD 알고리즘을 비교한 것이다. 그림 8은 n_{mh} 가 100이고 n_{mss}가 10일 때 메시지 생성 시간 간격의 증가에 대한 각 메시지에 피기백되는 인과적 장벽 벡터에 포함되는 평균 튜플수이다. 메시지 생성 시간 간격이 증가에 대한 메시지에 피기백되는 인과적 장벽 벡터에 포함되는 튜플의 수는 PCMD 알고리즘과 RCMD 알고리즘 모두 조금씩 감소한다. 메시지 생성 시간 간격이 증가하면 PCMD 알고리즘에서는 이동 호스트의 처리 지연(processing delay)의 발생이 줄어들고, RCMD 알고리즘에서는 기지국의 처리 지연의 발생이 줄어들기 때문에 인과적 순서를 위반하는 메시지의 수가 줄어든다. 따라서 메시지에 피기백되는 인과적 장벽 벡터에 포함되는 튜플의 수는 메시지 생성 시간 간격이 증가함에 따라서 감소한다. 또 PCMD 알고리즘과 RCMD 알고리즘에 포함되는 튜플의 비는 10 : 1 정도이다. 그러므로 RCMD 알고리즘의 메시지 오버헤드가 PCMD 알고리즘의 오버헤드보다 작다.

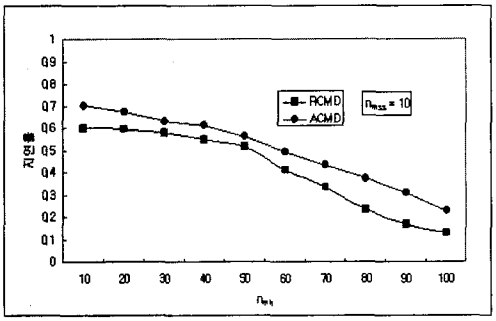


그림 9 n_{mh}의 증가에 대한 메시지 전송 지연율의 비교

그림 9는 n_{mss}가 10일 때 n_{mh}의 증가에 대한 메시지의 지연율을 RCMD 알고리즘과 ACMD 알고리즘을 비교한 것이다. n_{mh}가 10일 때는 RCMD 알고리즘과 ACMD 알고리즘 모두 지연율이 매우 높다. 실제로 n_{mh}가 작으면 이동 호스트 사이에 인과적 종속관계가 많이 발생하여 높은 지연율이 발생한다. 그러나 n_{mh}가 증가함에 따라서 이동 호스트 사이에 인과적 종속관계의 발생 횟수가 작아 지연율은 점점 낮아진다. 그림 9는 RCMD 알고리즘이 ACMD 알고리즘보다 지연율이 낮음을 보여준다. 이것은 수신측 이동 호스트가 다르지만 기지국은 같은 경우에 불필요한 지연이 발생하지 않기 때문이다.

지금까지 이 논문에서 수행한 시뮬레이션의 결과는 RCMD 알고리즘이 2 절에서 언급된 문제점들을 해결함으로써 ACMD 알고리즘보다는 지연이 낮게 발생하고 PCMD보다 메시지 오버헤드를 줄여 줌을 보여 주었다. 이동 호스트와 이동 호스트 사이에 인과적 메시지 순서화를 이행하는 PCMD 알고리즘은 계산의 대부분이 이동 호스트에서 수행되고, 자료의 처리 단위는 $O(n_{mh}^2)$이기 때문에, PCMD 알고리즘보다 RCMD 알고리즘이 메시지 오버헤드가 낮고 메시지 생성 시간 간격의 증가에 따른 메시지 오버헤드도 점차로 낮아진다. 기지국과 기지국 사이에 메시지 전달의 인과적 순서화를 이행하는 ACMD는 불필요한 지연으로 인하여 제안한 RCMD보다 지연율이 높다.

7. 결론 및 향후 연구 과제

복제 데이타의 관리, 자원 할당, 멀티미디어 데이타 제공, 화상 회의 등의 다양한 어플리케이션들은 분산 이동 시스템에서 메시지의 인과적 순서화를 이행하도록 요구한다. 메시지의 인과적 순서화를 제공하는 이전의 알고리즘들은 메시지 오버헤드가 높거나 이동 호스트가 알고리즘을 수행하고, 불필요한 지연이 발생하는 단점을 가지고 있다. 이 논문에서는 알고리즘의 대부분을 기지국에서 수행하도록 함으로써 이동 호스트의 수행 계산량을 줄이고, 인과적 종속정보를 기지국과 이동 호스트 단위로 유지시킴으로써 불필요한 지연을 줄이는, 분산 이동 시스템에서의 인과적 메시지 전달을 제공하는 효과적인 프로토콜을 제안하였다. 제안된 알고리즘은 메시지의 즉각 선행자 메시지에 관한 종속 정보만을 유지함으로써 종속 정보의 크기를 줄여서 낮은 대역폭을 갖는 무선 통신의 통신 오버헤드를 줄였다. 또 기지국과 이동 호스트 사이에 인과적 종속정보를 유지함으로써 이동 호스트와 이동 호스트 사이에 종속정보를 유지함으로 인해서 발생하는 처리 지연을 줄였다. 제안한 알고리즘은 최악의 경우에 메시지 오버헤드가 $O(n_{mss} \times n_{mh})$로 ACMD 알고리즘의 $O(n_{mss} \times n_{mss})$보다는 높지만 PCMD의 $O(n_{mh} \times n_{mh})$보다는 낮다.

앞으로는 하나의 기지국을 여러 개의 논리적인 기지국으로 나누고, 각 논리적인 기지국과 이동 호스트 단위로 인과적 장벽 벡터를 유지하는 알고리즘을 제안하고, 이 알고리즘을 Alagar의 세 번째 알고리즘과 비교하는 성능평가가 이루어 질 것이다.

참 고 문 헌

[1] George Coulouris, Jean Dollimore, and Tim Kindberg, *Distributed Systems*, 2nd ed. New York: Addison-Wesley, 1994.

[2] R. Prakash, M. Raynal, and M. Singhal, "An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments," *Journal of Parallel and Distributed Computing*, pp. 190-204, Vol. 42, No. 2, March 1997.

[3] A. Acharya and B.R. Badrinath, "A Framework for Delivering Multicast Messages in Networks with Mobile Hosts," *ACM-Baltzer Journal on Mobile Networks and Applications*, pp. 199-219, Vol. 1, No. II, 1996.

[4] Sridhar Alagar and S. Venkatesan, "Causal Ordering in Distributed Mobile Systems," *IEEE Transactions on Computers*, Vol. 46, No. 3, March 1997.

[5] B.R. Badrinath, A. Acharya, and T. Imielinski, "Impact of Mobility on Distributed Computations," *Operating Systems Review*, pp. 15-20, Vol. 27, No. 2, April 1993.

[6] B. R. Badrinath, A. Acharya, and T. Imielinski, "Structuring Distributed Algorithms for Mobile Hosts," *Proceedings of the 14th International Conference on Distributed Computing Systems*, pp. 21-28, June 1994.

[7] R. Prakash and M. Singhal, "Dependency Sequences and Hierarchical Clocks: Efficient Alternatives to Vector Clocks for Mobile Computing Systems," *ACM/Baltzer Journal on Wireless Networks*, pp. 349-360, 1997.

[8] Rosario Aiello, Elena Pagani, and Gian Paolo Rossi, "Causal Ordering in Reliable Group Communications," *Proceedings ACM SIGCOMM '93 Conference*, In *Computer Communication Review*, pp. 106-115, Vol. 23, No. 4, October 1993.

[9] Kenneth P. Birman and Thomas A. Joseph, "Reliable Communication in the Presence of Failures," *ACM Transactions on Computer*

Systems, pp. 47-76, Vol. 5, No. 1, February 1987.

[10] A. Schiper, J. Eggli, and A. Sandoz, "A New Algorithm To Implement Causal Ordering," *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*, pp. 83-91, June 1995.

[11] M. Raynal, A. Schiper, and S. Toueg, "The causal ordering abstraction and a simple way to implement it," *Information Processing Letters*, pp. 343-350, Vol. 39, No. 6, 1991.

[12] Khawar M. Zuberi and Kang G. Shin, "A Causal Message Ordering Scheme for Distributed Embedded Real-time Systems," *Proceedings of Symposium on Reliable and Distributed Systems*, pp. 210-219, October 1996.

[13] Frank Adelstein and Mukesh Singhal, "Real-Time Causal Message Ordering in Multimedia Systems," *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*, pp. 36-43, June 1995.

[14] Luis Rodrigues and Paulo Verissimo, "How to Avoid the Cost of Causal Communication in Large-scale Systems," *Proceedings of the 6th SIGOPS European Workshop*, September 1994.

[15] Friedemann Mattern and Stefan Funfrocken, "A Non-Blocking Lightweight Implementation of Causal Order Message Delivery," *Dagstuhl Seminar on Distributed Systems*, pp. 197-213, 1994.

[16] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System," *Communications of the ACM*, pp. 558-565, Vol. 21, No. 7, July 1978.

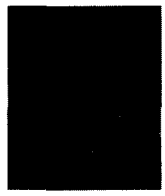
[17] R. Bagrodia and W-L. Liao. "Maisie: A Language for Design of Efficient Discrete-Event Simulations," *IEEE Transactions on Software Engineering*, April 1994.

노 성 주

1988년 8월 고려대학교 생물학과 학사 (컴퓨터학 부전공). 200년 8월 고려대학교 컴퓨터학사 석사. 200년 9월~현재 고려대학교 컴퓨터학과 박사과정, LG텔레콤 기술연구소 근무. 관심분야는 Distributed systems, Hierachical Mobile IP, IP routing for wireless mobile hosts

정 광 식

1993년 고려대학교 컴퓨터학과 학사 1995년 고려대학교 컴퓨터학과 석사 1995년~현재 고려대학교 컴퓨터학과 박사과정. 관심분야는 분산시스템, 이동컴퓨팅 시스템, 결함포용시스템



이 화 민

2000년 고려대학교 컴퓨터교육과 학사 2002년 고려대학교 컴퓨터교육과 석사 2002년~현재 고려대학교 컴퓨터교육과 박사과정. 관심분야는 분산 시스템, 그리드 컴퓨팅, 에이전트 시스템

유 헌 창

1989년 고려대학교 이과대학 컴퓨터학과 졸업. 1991년 고려대학교 대학원 컴퓨터학과 졸업(이학석사). 1994년 고려대학교 대학원 컴퓨터학과 졸업(이학박사). 1995년~1997년 서경대학교 이공대학 컴퓨터공학과 조교수. 1998년~현재 고려대학교 사범대학 컴퓨터교육과 조교수. 관심분야는 분산 시스템 이동 컴퓨팅 시스템, 결함 포용 시스템, 웹기반교육

황 종 선

정보과학회논문지 : 정보통신 제 30 권 제 1 호 참조