분산 이동 시스템에서 인과적 메시지 전달을 위한 효율적인 알고리즘*

노성주[U] 정광식 김기범 안진호 황종선
고려대학교 컴퓨터학과 분산시스템연구실
{sjroh, cks, kibom, jhahn, hwang}@disys.korea.ac.kr

An Efficient Algorithm for Causal Message Delivery in Distributed Mobile Systems

SungJu Roh[U] KwangSik Chung Kibom Kim JinHo Ahn ChongSun Hwang
Dept. of Computer Science & Engineering, Korea University

요 약

분산 이동 시스템은 단순한 통신 기능에서 작업 흐름 관리, 화상회의, 전자 메일과 같은 서비스를 제공하는 시스템으로 급속히 확대·발전하고 있으며, 이들 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 비확장성 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 통하여 기존 기법의 문제점을 해결하는 효율적인 인과적 메시지 전달 기법을 제안한다. 제안하는 알고리즘은 이전의 알고리즘들과 비교해서 낮은 메시지 오버헤드를 가지며, 메시지를 전달할 때의 불필요한 지연(inhibition)이 발생하지 않는다. 또한 제안된 알고리즘은 이동 호스트의 에너지 사용에 대한 제약성, 무선 통신의 낮은 대역폭, 이동 호스트 수의 증가에 대한 확장성, 무선 통신의 잦은 접속 단절과 같은 요소를 고려한다.

1. 서론

분산 시스템은 통신 네트워크에 의해서 서로 연결되고, 소프트웨어가 갖추어진 자율적인 컴퓨터들의 집합이다[4]. 통신 기술의 발전과 컴퓨터의 소형화로 이동 처리 단위(mobile processing unit)의 수는 점차로 증가 추세에 있다. 이러한 추세에 힘입어 분산 컴퓨팅 환경 역시 이동 처리 단위를 포함하는 분산 이동 환경으로 확대되어 가고 있다. 분산 이동 시스템은 이동 호스트(MH: mobile host)의 이동성을 지원해 줌으로써 컴퓨팅에 대한 지역적 제약을 제거한다. 그러나 분산 이동 시스템은 MH의 이동성, 낮은 대역폭, MH의 자원 제약성, 잦은 접속 단절 등과 같은 새로운 문제에 직면하였다[1, 2, 9, 10].

시스템 내의 MH들 사이에는 메시지를 주고 받음으로써 종속 관계(dependency)가 발생한다. 복제 데이터의 관리, 자원 할당, 멀티미디어 데이터 제공, 화상 회의, 전자 메일 등의 서비스를 제공하는 다양한 어플리케이션들은 하나의 프로세스 내부 또는 여러 프로세스들 사이에서 발생하는 사건의 종속관계를 고려하여 순서를 맞추도록 요구된다[3]. 최근에 분산 이동 시스템이 보편화됨에 따라 기존의 분산 시스템에서 구현된 어플리케이션들이 분산 이동 시스템에서도 구현되도록 요구되고 있다. 특히 분산 이동 시스템에서는 낮은 통신 오버헤드, MH의 낮은 계산 오버헤드와 메모리 오버헤드를 고려한 메시지의 인과적 순서화(causal ordering)에 대한 연구가 활발히 진행중이다.

기존의 인과적 순서화 알고리즘들은 MH의 증가에 대한 확장성이 있으면 MH에서 계산의 대부분이 수행되거나[2], 낮은 메시지 오버헤드를 발생시키면 메시지 전달의 불필요한 지연이 발생하고, 지연을 발생하지 않으면 메시지 오버헤드가 높은 단점이 있다[9]. 이러한 문제점들을 해결하기 위하여, 우리는 기지국(MSS: moblie support station)과 MH 사이의 종속 정보를 유지하고 또 인과적 장벽 벡터(causal barrier vector)를 사용함으로써 분산 이동 시스템에서 MH에게 인과적 메시지 전달(causal message delivery) 알고리즘을 제안한다. 제안된 알고리즘은 MH의 에너지 사용의 제약성, 무선 통신의 낮은 대역폭, 확장성, 접속 단절 등의 요소를 고려하여 인과적 순서화를 위해서 필요한 제어 정보의 크기를 줄이고 불필요한 지연이 발생하지 않는다. 제안한 알고리즘은 기존의 알고리즘보다 낮은 메시지 오버헤드를 가지며 낮은 핸드오프 비용을 갖는다.

2. 관련 연구

분산 시스템에서 인과적 순서화가 처음으로 구현된 시스템은 ISIS 시스템이다[6]. ISIS 시스템은 메시지를 보낼 때, 목적지 프로세스에게 이전에 보낸 모든 메시지를 보내는 메시지에 피기백한다. Schiper는 벡터 클럭을 이용하여 분산 시스템에서 인과적 순서화를 하였다[7]. 이 방법은 각 메시지에 (목적지 사이트, 벡터시간)의 쌍으로 구성된 정보가 피기백된다. Raynal의 인과적 순서화 알고리즘[8]은 Schiper의 알고리즘과 유사하다. 그러나 벡터 시간을 사용하는 대신에 각각의 프로세스 P_i는 $N \times N$ 행렬 $SENT$를 유지한다. $SENT$는 모든 다른 프로세스에게 보내진 메시지의 수를 나타낸다. 또 N개의 요소를 가진 정수 벡터 $DELIV$는 P_i가 모든 다른 프로세스로부터 받은 메시지의 수를 나타낸다. 각각의 메시지에 $SENT$ 행렬이 피기백 된다. 그러므로 메시지 오버헤드는 $O(N^2)$이다.

Alagar는 Raynal의 RST 알고리즘을 기반으로 하여 이동 시스템에서 메시지의 인과적 순서화를 이행하는 3개의 알고리즘을 제안하였다[9]. 첫 번째 알고리즘에서는 각각의 메시지에 모든 MH에 대한 종속 정보, 즉 $n_{MH} \times n_{MH}$(단, n_{MH}는 MH의 수) 행렬을 피기백한다. 이 알고리즘은 높은 통신 오버헤드 때문에 MH의 증가에 대한 확장성(scalability)이 좋지 않다. 두 번째 알고리즘은 각각의 메시지에 이동 시스템의 MSS에 대한 종속 정보, 즉 $n_{MSS} \times n_{MSS}$ (단, n_{MSS}는 MH의 수)행렬을 피기백한다. 그러므로 확장성은 좋지만 MSS 사이에서 인과적 순서화를 이행함으로써 불필요한 지연이 발생한다. 세 번째 알고리즘은 두 번째 알고리즘에 발생하는 불필요한 지연을 줄이기 위해서 각각의 MSS를 k개의 논리적인 단위로 나누어서, 각 메시지에는 $n_{MSS} \times k$개의 MSS 사이의 종속 정보, 즉 $(n_{MSS} \times k) \times (n_{MSS} \times k)$ 행렬이 피기백된다.

Prakash가 제안한 알고리즘은 각각의 메시지에 즉각 선행자 메시지에 관한 종속 정보만을 피기백한다[2]. 이 알고리즘은 메시지 오버헤드를 줄여줄 수 있지만 최악의 경우에 메시지 오버헤드가 $O(n^2_{mh})$이기 때문에 Alagar의 첫 번째 알고리즘과 같은 메시지 오버헤드를 가진다. 그러므로 완전한 확장성을 제공하지 못한다. 더구나 MH 상에서 알고리즘의 대부분이 수행되기 때문에, MH의 에너지 사용의 제약성과 무선 통신의 낮은 대역폭이 고려되지 않았다.

3. 시스템 모델

분산 이동 시스템은 유선 네트워크와 MH, 고정 호스트, MSS로 구성된다. MH는 네트워크에 접속된 도중에도 이동할 수 있다. MH가 다른 MH나 고정 호스트와 통신을 하기 위해서는 반드시 자신이 상주하는 셀(cell)의 MSS와 무선 통신을 해야 한다. 하나의 셀은 MSS가 관리하는 논리적 또는 지리적 영역이다. MH가 자신과 통신하는 MSS가 바뀔 때, 핸드오프(handoff) 과정이 수행된다.

* 이 연구는 한국과학재단 핵심전문연구(과제번호:981-0924-126-2)인 "분산시스템에서 비동기적 회복 기법의 개발"의 지원으로 수행되었음.

MSS가 관리하는 영역 내의 무선 통신은, 즉 MSS와 MH간의 통신은 FIFO(First-In First-Out)를 가정한다. 그리고 고정 네트워크에서 이루어지는 유선 통신은 FIFO를 가정하지 않는다.

시스템의 MH들 사이에서 발생하는 사건들은 종속 관계가 발생한다. 이러한 종속 관계는 Lamport의 전후 관계(happened-before), 즉 →를 사용해서 표현될 수 있다[5]. 메시지 M을 보내는 사건을 $Send(M)$, 메시지 M을 받는 사건을 $Deliver(M)$이라고 하자.

- 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.

그림 1과 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 보내는 MH는 다르지만 목적지 MH는 같으므로 인과적 순서화를 이행하려면 $Deliver(M_1) \rightarrow Deliver(M_3)$이어야 한다.

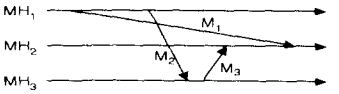


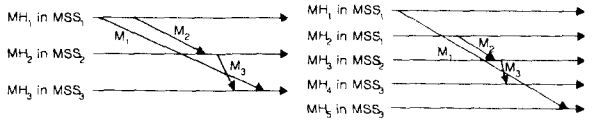
그림 1 인과적 순서를 위반하는 예

그러나 그림 1은 M_3이 M_1보다 먼저 수신되어서 인과적 순서를 위반한다.

4. 제안된 인과적 메시지 전달 알고리즘

4.1 동기

기존의 여러 알고리즘들은 분산 이동 시스템에서 MH 단위로 메시지 순서화를 이행하여 메시지 오버헤드가 커지고 확장성이 결여되었다. 또 MSS 단위로 인과적 순서화를 한다면 그림 2의 (b)와 같이 불필요한 지연(inhibition)이 생긴다. 그림 2의 (a)의 경우, 메시지 M_1을 MSS_1의 MH_1에서 MSS_3의 MH_3에게 보내고, 메시지 M_2를 MH_1에서 MSS_2의 MH_2로 보내고, 마지막으로 MH_2로 MH_2가 메시지 M_3을 보내는 경우, 메시지 M_3은 MSS_2에 버퍼되어 지연된다. 이러한 경우가 발생하는 것은 M_1과 M_2처럼 목적지 MH가 같은 경우이다. 그림 2의 (b)의 경우와 같이 메시지를 보내는 경우, Alagar의 두 번째 알고리즘처럼 MSS 단위로 인과적 순서화를 한다면 지연이 발생하지 않아도 될 상황에서 지연이 발생한다. 이러한 지연을 제거하기 위하여, 우리는 보내는 측의 MSS와 받는 측의 MH 사이의 인과적 순서화를 이행한다. 그림 2의 (b)는 M_1과 M_2의 수신자 MSS는 같지만, 수신자 MH는 다르기 때문에 지연이 불필요하다. 결국 같은 MSS에 있다하더라도 목적지 MH가 다르면 지연이 발생하지 않는다. 이것은 MH가 MSS로부터 받은 메시지의 수를 나타내는 $n_{mh} \times n_{mss}$ 행렬을 MH가 상주하는 셀의 MSS에 유지함으로써 가능하다. 또한 MH에 모든 자료를 유지하는 Prakash의 방법과는 다르게, 크기 n_{mh}의 인과적 장벽 벡터를 MH가 상주하는 셀의 MSS에 유지하고 각 메시지에 이 벡터를 피기백하여 보내면, 이 메시지를 받는 측의 MSS는 $n_{mh} \times n_{mss}$ 행렬의 원소와 비교하여 MH에 전달할 것인지를 결정한다. 이때 이전에 보낸 인과적 관계가 있는 모든 메시지가 목적지 MH에 전달되지 않았으면, 그 메시지는 목적지 MH로 전달되지 않고 이전에 보낸 인과적 종속관계에 있는 모든 메시지가 전달될 때까지 목적지 MH가 상주하는 MSS에서 지연되기 때문에 인과적 장벽 벡터라고 한다.



(a) 지연이 필요한 경우 (b) 지연이 불필요한 경우

그림 2 지연이 필요한 경우와 불필요한 경우

제안된 알고리즘은 Alagar의 첫 번째 알고리즘보다 메시지 오버헤드는 훨씬 작고, 두 번째 알고리즘의 단점인 메시지 전달의 불필요한 지연은 발생하지 않는다.

4.2 자료 구조

MSS는 s, MH는 h에 아래첨자를 붙여 표기한다. 인과적 순서로 메시지를 전달하기 위해서 각 MSS s_i가 유지하는 자료구조는 다음과 같다.

- $SSENT_i$: 자신이 보낸 메시지의 수를 의미하는 정수, $SSENT_i$를 유지한다. $SSENT_i$는 s_i가 메시지를 하나 보낼 때마다 1씩 증가한다.
- $SToHCB_i$: $SToHCB_i$는 메시지에 대한 전달 제약조건을 나타내는 직접 종속정보(direct dependency)이다. s_i는 벡터 $SToHCB_i$를 유지한다. 벡터 $SToHCB_i$의 각 요소는 (MSS 아이디, MSS가 보낸 메시지의 수), 즉 $(SrcMSSID,\ SSENT_i)$ 쌍의 집합으로 구성된다. 예를 들어 메시지 M에 피기백된 인과적 장벽 벡터 $SToHCB_M$에 대하여, $(k,\ x) \in SToHCB_M[j]$이면, MSS s_k가 보낸 메시지 번호가 x인 메시지를 MH h_j가 받은 후에 메시지 M이 MH h_j에게 전달될 수 있음을 의미한다. 이와 같이 메시지가 전달되기 위해서 메시지 M에 피기백된 $SToHCB_M$이 의미하는 제약 조건이 만족될 때까지 그 메시지는 지연되므로 $SToHCB_i$를 인과적 장벽 벡터라고 한다. 각 메시지에는 $SToHCB_i$ 벡터가 피기백되고, 시작시에 공집합으로 초기화된다.
- $SToHDeliver_i$: $n_{mss} \times n_{mh}$행렬이고 $SToHDeliver_i[j,*]$, 즉 $SToHDeliver_i$의 j 번째 열은 s_i가 각각의 MH에게 보낸 메시지의 수를 나타낸다. 예를 들면, $SToHDeliver_i[j,k]=x$이면 MSS s_j로부터 MH h_k로 메시지 번호가 x보다 작거나 같은 메시지는 모두 전달되었음을 MSS s_i가 알고 있음을 의미한다. 처음에 $SToHDeliver_i$의 모든 요소는 0으로 초기화된다.
- $Suspend_i$와 $WaitAck_i$: $SToHCB_i$와 $SToHDeliver_i$외에도 s_i는 메시지 큐인 $Suspend_i$와 $WaitAck_i$를 유지한다. 현재 받은 메시지가 전달 가능하지 않으면, 즉 인과적 순서를 위반하면, $Suspend_i$ 큐에 일시적으로 저장된다. 만약 현재 받은 메시지가 인과적 순서를 위반하지 않는다면 $WaitAck_i$ 메시지 큐에 그 메시지에 대한 응답이 올 때까지 일시적으로 저장한다. 또 $Suspend_i$에 저장된 메시지들은 $WaitAck_i$ 큐에 저장된 메시지들에 대한 응답이 올 때마다, 인과적 순서를 위반하는지를 재검사하여 인과적 순서를 위반하지 않으면 MH에게 전달한다.

4.3 알고리즘

4.3.1 정적 모듈

A. 메시지 송신 사건

MSS s_i가 메시지 M을 목적지 MH 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);
```

B. 메시지 수신 사건

MSS s_j로부터 메시지 M을 MH h_i가 받는 알고리즘에서, h_i의 지역 MSS s_i가 하는 일은 다음과 같다.

```
1. MSS s_i receives (M, SrcMSSID, SSENT_M, SToHCB_M)
2. if SToHCB_M[i] ≠ ∅
   then
     for ∀s_k : (k,x) ∈ 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], (SrcMSSID, SSENT_M)), SToHCB_M[i])
3.4 for ∀k ∉ {h_i, h_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 ≠ ∅
    then
      for ∀ h_k : (k, x) ∈ StoHCB_Suspend_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 MSS와 수신측 MH에 제약조건이 하나가 되도록 메시지 전달의 제약 조건의 합집합을 구하여 리턴하여 준다. *DiffMax* 함수는 현재의 제약 조건으로부터 이미 만족되어 있는 제약 조건을 삭제한다.

4.3.2 핸드오프 모듈(handoff module)

MH h_i가 MSS s_i로부터 s_j로 이동한다고 하자. 먼저 h_i가 s_j가 담당하는 셀로 들어가서 *registerMH*(h_i, s_i) 메시지를 MSS s_j에게 보낸다. 이 메시지는 h_i가 s_j에게 자신의 도착을 알리는 역할을 한다. h_i로부터 *registerMH* 메시지를 받은 s_j는 이전의 s_i에게 *migratedMH*(h_i) 메시지를 보낸다. 이전의 s_i는 *migratedMH* 메시지를 받자마자 h_i와 관련된 정보인 $StoHCB_i$, $StoHDeliver_i$, $Suspend_i$, $WaitAck_i$를 s_j에게 보내준다. 이러한 정보를 받은 s_j는 h_i의 관련 자료들을 갱신하고 새로 만든다. 만약 핸드오프가 끝나기 전에 s_i에게 보내진 메시지는 s_j에게 전파한다.

4.3.3 단절/재접속 모듈(disconnection/reconnection module)

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

5. 정당성 증명

MH h_i, h_j, h_k, h_l이 상주하는 셀의 MSS를 각각 s_i, s_j, s_k, s_l라 하자.

보조정리 1. MSS s_i가 보내는 메시지 M에 $(k, x) \in StoHCB_M[j]$인 정보가 피기백 된다면 $Send(M_x) \rightarrow Send(M)$이고 MSS s_k로부터 MH h_j로 보낸 메시지 M_x가 존재한다.

증명) 두가지 경우가 존재한다. 첫째로 $k=i$일 때, $StoHCB_i[j]$는 메시지 송신 사건의 단계 3에서 갱신된다. M_x를 h_j로 보낸 후에 이전의 $StoHCB_i[j]$는 (i, x)로 치환된다. M을 보낼 때 $StoHCB_i[j]$가 (k, x)를 포함하고 있다면, 즉 $(k, x) \in StoHCB_i[j]$라면 s_i는 M_x를 보낸 후에 M을 보낸다(그림 3(a)).

둘째로 $k \neq j$일 때(그림 3의 (b)), h_i가 M'를 받으면, s_i는 (k, x)를 메시지 수신 사건의 단계 3.4에서 $StoHCB_i[j]$에 추가하므로 $(k, x) \in StoHCB_M[j]$이다. 그러므로 직접 인과적 종속정보 사슬(그림 3(b)의 점선인 DC_i), 즉 $Send_k(M_x) \rightarrow \cdots \rightarrow Deliver_h(M')$에 의해서 s_i는 (k, x)를 $StoHCB_i[j]$에 추가시킨다. 결국 $(k, x) \in StoHCB_i[j]$라면, s_i는 M'가 h_i에게 전달된 후에 M을 보내기 때문에, $Send(M_x) \rightarrow Send(M)$이다. ∎

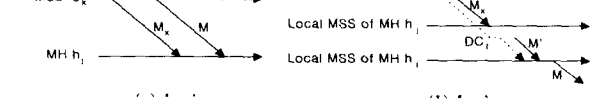


그림 3 보조 정리 1의 예

보조정리 2. MSS s_i가 보낸 메시지 M과 MH h_j에게 보내진 메시지 M_x가 다음의 3가지 조건을 만족하고,

(i) $Send(M_x) \rightarrow Send(M)$

(ii) $Deliver_h(M_x) \rightarrow Send(M)$

(iii) $Send(M)$을 하기 전에 h_j에게 마지막으로 보낸 메시지가 M_x일 경우, h_j에게 보낸 M_x는 존재하지 않는다.

s_i가 보낸 메시지 M과 임의의 MSS s_k가 h_j에게 보낸 메시지 M_x가 존재하면, $(k, x) \in StoHCB_M[j]$이다.

증명) 지면의 제약 때문에 생략한다. ∎

정리 1. 두 개의 메시지 M과 M'에 대하여, $Send(M) \rightarrow Send(M')$이면 $Deliver(M) \rightarrow Deliver(M')$이다.

증명) 메시지 M과 M'의 목적지 MH는 h_j이고, M'는 s_k가 h_j에게 보낸 마지막 메시지이고, $Deliver_h(M') \rightarrow Send(M)$이라 하자. 보조정리 1, 2에 의하여 메시지 M에 피기백되는 $StoHCB_M[j]$는 (k, x)를 원소로 갖는다. 즉, $Send(M) \rightarrow Send(M')$이며, 목적지 MH가 h_j이고 $Send(M) \rightarrow Send(M'') \rightarrow Send(M')$인 M''는 존재하지 않는다. 메시지 수신 사건의 단계 1은 M'가 h_j에게 전달된 후에 M이 h_j에게 전달되도록 한다. 그러므로 직접 인과적 선행자 메시지와 후행자 메시지 쌍간의 인과적 순서화가 이루어진다. 인과적 관계의 이행성(transitivity)은 분산 이동 시스템에서 모든 메시지들에 대한 인과적 순서화가 이루어지도록 한다. ∎

6. 결론 및 향후 연구과제

메시지의 인과적 순서화를 제공하는 이전의 알고리즘들은 메시지 오버헤드가 높은 단점을 가지고 있다. Alagar의 두 번째 알고리즘은 메시지 오버헤드가 작다 하더라도 불필요한 지연이 발생한다. Prakash의 알고리즘은 각 메시지에 즉각 선행자 메시지에 관한 종속 정보만을 피기백하여 상대적으로 다른 알고리즘에 비해서 메시지 오버헤드가 작지만 계산의 대부분이 MH 상에서 실행되기 때문에 MH의 수행 계산량이 많은 단점이 있다. 따라서 이 논문에서는 알고리즘의 대부분을 MSS에서 수행하도록 함으로써 MH의 수행 계산량을 줄이고, 인과적 종속정보를 MSS와 MH 단위로 유지시킴으로써 불필요한 지연을 줄이는, 분산 이동 시스템에서의 인과적 메시지 전달을 제공하는 효과적인 알고리즘을 제안하였다. 제안된 알고리즘은 메시지의 즉각 선행자 메시지에 관한 종속 정보만을 유지함으로써 종속 정보의 크기를 줄여서 낮은 대역폭을 갖는 무선 통신의 통신 오버헤드를 줄였다. 앞으로는 제안된 알고리즘의 성능평가와 하나의 MSS를 여러개의 논리적인 MSS로 나누고, 각 논리적인 MSS 단위로 인과적 장벽 벡터를 유지하는 알고리즘에 대한 제안과 성능평가가 이루어 질 것이다.

7. 참고문헌

[1] A. Acharya, 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. 11, 1996.

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

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

[5] 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.

[6] 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, Feb. 1987.

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

[8] 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.

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

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