분산 시뮬레이션을 위한 효율적인 보수적 알고리즘 개발

Development of Efficient Conservative Algorithm for Distributed Simulation

이영해*, 배강용*
Young-Hae Lee and Gang-Yong Bae

Abstract

There are two approaches to handle the Causality Error in parallel and distributed simulation. One approach is based on the conservative time synchronization and the other is the optimistic time synchronization. In this paper an efficient null message reduction method for the conservative time synchronization approach is suggested with the experimental results, which could improve performance of simulation and avoid deadlock situations.

* 한양대학교 산업공학과

1. 서 론

시스템을 수학적인 방법으로 모델링하거나 분석하기 어려울 때 시뮬레이션이 유용한 방법으로 알려져 있다. 그러나 시스템이 대형화, 복잡화 되어가고, 시뮬레이션 결과를 실시간으로 얻어내야 하는 상황에서는 기존의 순차적 시뮬레이션(sequential simulation) 방법으로는 해결할 수 없게 된다. 이와 같은 문제점들을 해결하기 위해 시뮬레이션 프로세스를 여러 대의 컴퓨터에서 나누어 수행하는 분산 시뮬레이션(distributed simulation) 방법이 개발되어졌고, 이 분야에 많은 연구가 진행되어 왔다[4, 5, 7, 8, 9].

병렬·분산 시뮬레이션에서는 실제 시스템의 프로세스들을 Logical Process(LP)로 표현하는 데, 이러한 LP들간의 시간 동기화(time synchronization) 문제가 가장 큰 이슈가 되고 있다[2]. 각 LP가 독립적으로 시뮬레이션을 진행하기 때문에 이벤트의 순서가 뒤바뀌는 경우가 있을 수 있다. 이러한 오류를 분산 시뮬레이션에서 인과 오류(causality error)라고 한다. 이러한 오류가 발생하면 올바른 시뮬레이션 결과를 도출해 내기 어렵다.

시간 동기화를 위해 크게 두 가지의 방법이 제안되었는데, 한가지는 보수적 알고리즘(conservative algorithm)이고 다른 한가지는 낙관적 알고리즘(optimistic algorithm)이다. 보수적 알고리즘은 LP에서 이벤트를 진행시킬 때 안전한 이벤트(safe event)라고 보증되는 이벤트만 진행시키는 방법이다. 다시 말하면 나중에 이벤트의 순서 변화(event order change)가 발생하지 않는다고 보증할 수 있는 이벤트만 진행시킨다. 이런 방법으로 이벤트의 처리 순서가 바뀌는 상황을 미리 방지할 수 있다[2]. 그리고 낙관적 알고리즘은 이벤트 순서 변화를 염두에 두지 않고 각 LP에서 독립적으로 이벤트를 진행시키는 데, 시뮬레이션 도중 이벤트의 순서 변화가 발생하면 모든 시뮬레이션 상태를 그 시간의 상황으로 되돌린 후 다시 시뮬레이션을 하는 방법이다[6].

보수적 알고리즘에서는 교착상태(deadlock)의 해결이 가장 큰 문제가 되고 있다. 교착상태를 해결하기 위해 교착상태를 미리 방지하는 방법과 교착상태를 찾아내고 이를 해결하는 방법이 제안되었다[2,8]. 교착상태를 미리 방지하는 방법에서 가장 많이 쓰이는 방법은 null message방법이다. 이 방법은 과다한 null message를 발생함으로써 LP간의 통신에 과다한 오버 헤드가 발생하게 된다.

본 논문에서는 보수적 알고리즘에서 null message의 수를 줄임으로써 시뮬레이션 수행도를 향상시키고, 교착상태를 해결하는 효율적인 방법을 제안한다.

2. 보수적 알고리즘

병렬·분산 시뮬레이션에서는 실제 시각과 시뮬레이션 시각을 나타내는 두 가지 시계가 존재한다. 병렬·분산 시뮬레이션이 어려운 이유는 각 프로세스에 사건 메시지들은 실제 시각의 순서대로 도착하는 데 비해, 그 사건들은 스케쥴되어 발생하는 시뮬레이션 시각의 순서대로 처리되어야 하기 때문이다. 이때 사건들이 처리되어야 할 시간적 순서를 지정하거나 규정하는 것이 바로 전역 인과성 조건 (global causality constraint)이다.

보수적 알고리즘과 낙관적 알고리즘은 현재까지도 많이 쓰이고 있는 시간 동기화 알고리즘이다. 보수적 알고리듬에서는 항상 전역 인과성 조건이 엄격하게 지켜진다. 그러므로 보수적 알고리듬에서 각 프로세스는 시뮬레이션 어플리케이션에 대한 사전 지식을 이용하여 안전하게 처리될 수 있는 사건들의 집합을 결정하고 처리하는 과정을 반복한다[14].

분산 시뮬레이션을 수행할 때 이 두 가지 알고리즘 중에서 어떤 것을 선택해야 할지 결정하는 것이 중요하다. 형태가 자주 바뀌는 시스템은 낙관적 알고리즘이 유용한 방법이고 형태가 고정된 시스템은 보수적 알고리즘을 적용해야 한다[7]. 다음에는 본 연구의 대상이 되는 보수적 알고리즘 개념을 자세히 설명한다.

보수적 시간 동기화 알고리즘은 시간 동기화 알고리즘에서 가장 먼저 소개된 알고리즘으로서, Chandy-Misra 알고리듬은 가장 널리 알려진 보수적 알고리듬이다. 이 알고리즘의 기본 개념은 어떤 이벤트를 진행시켰을 때 그 후 이 이벤트보다 작은 time stamp를 가진 이벤트를 다른 LP에서 받지 않는다고 확신할 수 있는 이벤트만 진행시킨다는 것

이다.

보수적 알고리즘에는 input waiting rule과 output waiting rule이 존재하는 데, 이 두 가지 규칙을 이용해서 인과 오류를 방지할 수 있는 것이다[7]. 아래에서 두 가지 규칙을 자세히 설명한다.

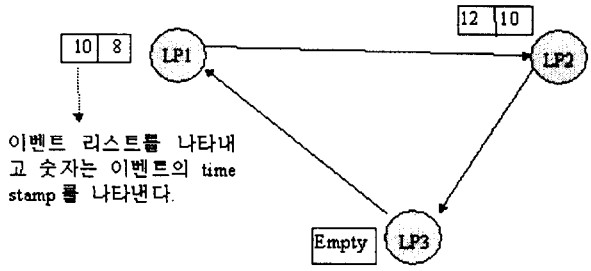
1) input waiting rule

어떤 LP에서 메시지를 진행하려 하면 이 LP로 메시지를 보내는 모든 LP로부터 메시지를 받아야만 된다. 이들에서 오는 메시지의 time stamp가 얼마인지 메시지를 받기 전까지 예측하지 못하므로 기다리는 것이다. 모든 LP에서 받은 메시지를 이벤트 리스트에 삽입하고 리스트에 있는 메시지들 중 time stamp가 가장 작은 메시지를 진행시키는 것이다.

2) output waiting rule

input waiting rule에 의해 진행된 메시지는 곧바로 다른 LP로 보내지 않는다. 이후에 어떤 메시지도 현재 보낼 메시지의 time stamp보다 크지 않다는 보장이 있을 때만 메시지를 다른 LP에게 보낼 수 있다. 이렇게 함으로써 시간 순서로 메시지를 보낸다는 것을 보장할 수 있다.

보수적 알고리즘을 수행하는 데 가장 큰 문제점은 교착상태가 발생한다는 것이다. 보수적 알고리즘에서 교착상태는 input waiting rule과 output waiting rule때문에 야기되는 것이다. 만약 어떤 LP가 하나 이상의 LP에서 메시지를 받지 못한다면 메시지를 받을 때까지 계속 기다리고 이벤트 리스트에 있는 메시지를 진행시키지 못한다. 이러한 상태를 process blocking이라고 한다. process blocking 상태인 LP로부터의 메시지를 다른 LP가 기다리는 경우가 있다. 이러한 상황이 cycle를 이룬다면 cycle상에 있는 모든 LP들은 이벤트를 진행시키지 못하게 된다. 이러한 상태를 교착상태라고 부른다. 교착상태를 그림으로 표현하면 그림 2.1과 같다.



<그림 1> 교착상태의 예

<그림 1>에서 보면 LP1은 LP3로부터 메시지를 받아야만 이벤트를 진행시킬 수 있다. 그러나 LP3의 이벤트 리스트가 비어 있으므로 LP1은 process blocking 상태가 된다. 그리고 LP2는 LP1에서 메시지를 기다리지만 LP1은 blocking상태이므로 LP2도 blocking 상태가 된다. LP3는 LP2에서 메시지를 받아들여야 LP1으로 메시지를 보낼 수 있지만 LP2가 blocking상태이므로 메시지를 받지 못한다. 이런 상태가 보수적 알고리즘에서의 교착상태이다. 이러한 교착상태를 해결하기 위해서 크게 교착상태를 미연에 방지하는 방법(deadlock avoidance)과 교착상태를 발견한 후 해결하는 방법(deadlock detect and recovery)이 제안되었다. 이 두 가지 방법을 아래서 자세히 설명한다.

2.1 교착상태 해결을 위한 방법

그 동안의 연구에서 교착상태를 회피하거나 복구하기 위한 많은 알고리듬들이 제안되었다. 대표적인 알고리듬으로 Misra의 null 메시지를 이용한 알고리듬이 있다[8].

1) Deadlock Avoidance

이 방법은 교착상태가 아예 일어나지 못하도록 하는 방법이다. 이를 위해서 각 LP에서 이벤트를 진행시킬 때마다 null message를 발생시킨다. null message란 실제 이벤트는 아니고 단지 시간에 관계되는 정보만 포함하고 있다. 즉, 어떤 LP에서 null message를 발생시킨다는 것은 그 LP에서 이 시간

이후 null message의 time stamp보다 작은 time stamp를 가진 이벤트는 발생하지 않는 다는 것을 다른 LP에게 알리는 것이다. 예를 들어 LP2에서 값 8의 time stamp를 가진 null message를 다른 LP에 보낸다면 LP2에서 이 시간 이후 time stamp가 8보다 작은 이벤트는 절대 보내지 않는 다는 것을 보증하는 것이다. 하지만 모든 이벤트를 진행시킬 때 마다 null message를 보내야 하므로 수많은 null message를 발생시킴으로써 시뮬레이션 오버헤드를 증가시키게 된다. 그러므로 이 방법에서는 null message의 개수를 줄이는 것이 주요 관심사가 된다.

보수적 알고리즘에서 또 한가지 해결해야 할 문제점이 있다. 그것은 null message의 time stamp를 어떻게 결정하느냐 하는 것이다. 어떤 이벤트를 진행시키고 그 이벤트의 time stamp를 기준으로 다음 null message의 time stamp를 결정하는데 이벤트의 time stamp에 얼마의 시간을 더함으로써 결정하게 된다. 이때 더하는 시간을 lookahead라고 한다. null message는 안전한 이벤트를 결정하는 데 중요한 역할을 하므로 효과적인 lookahead를 계산하는 것이 필요하다. 일반적인 FCFS 대기행렬 네트워크 시뮬레이션에서는 이벤트의 time stamp에 서비스시간을 더해서 null message의 time stamp를 계산한다. 여기서 서비스시간이 lookahead가 되는 것이다. 이러한 방법은 가장 간단한 방법으로 lookahead를 계산하는 것이고 시뮬레이션 시스템의 다른 정보들을 이용해서 더 효과적인 lookahead를 계산하는 방법의 연구가 필요하다[2].

2) Deadlock Detection and Recovery

이 알고리즘은 우선 두 단계를 정의하는 데, 하나는 parallel phase이고 다른 하나는 phase interface이다. Parallel phase는 교착상태가 발생할 때까지 시뮬레이션을 수행하는 단계이고, phase interface는 교착상태를 해결하기 위해 LP에서 계산을 수행하는 단계이다. 이 알고리즘의 특징은 central controller가 존재한다는 것이다. 즉, 분산 환경에 계층구조를 도입한 것이다. Central controller는 각 단계를 찾아내고 다음 단계를 시작하는 것만 담당한다. 그리고 어떠한 계산도 포함되지 않는다. 이러한 계산은 모두 LP에서 계산하고 단지 각 단계에 대한 결과만 받을 뿐이다. 발생 가능한 병목현상을 제거하기 위해 distributed deadlock detection algorithm을 제안하였다[3].

또 한가지 제안된 방법은 marker라는 특별한 메시지를 도입하는 것이다. marker는 LP 네트워크를 순회하면서 교착상태 발견에 필요한 정보를 수집한다. 이 방법에서는 LP에 마지막으로 marker가 거쳐간 후 메시지를 받거나 보냈는지 여부를 표시하는 flag을 가지고 있다. 만약 LP에 marker가 방문한 후 아무런 메시지도 받거나 보내지 않았다면 flag을 white로 한다. 그 이외의 경우라면 flag을 black으로 한다. 시뮬레이션이 시작할 때는 모든 LP가 white로 초기화된다. 만약 marker가 방문한 N개의 LP가 white면 교착상태라고 판정할 수 있다. 여기서 N은 LP 네트워크에서 모든 채널의 개수를 표시한다. 그리고 marker가 방문한 LP의 가장 작은 다음 이벤트 (next event) 시간을 가지고 있음으로써 교착상태를 해결할 수 있다. 교착상태가 발생하면 가장 작은 time stamp를 가진 이벤트를 진행시킴으로써 이를 해결할 수 있다[8].

2.2 Null messages를 줄이기 위한 기존 방법

앞에서 설명했듯이 보수적 알고리즘에서 교착상태를 해결하는 것이 중요한 문제인데 이를 위해 null message방법이 제안되었다. 그런데 이러한 null message를 이벤트를 보낼 때 마다 보낸다면 null message를 처리하는데 상당한 시간이 소요되고 많은 메시지를 처리하기 위하여 통신 병목현상을 초래하게 된다. 그래서 그동안 Chandy-Misra 알고리듬의 성능을 향상시키기 위해서 여러 알고리듬들이 제안되었다. Chandy-Misra 알고리듬의 성능에 가장 큰 영향을 주는 것은 null 메시지의 수이다.

가장 먼저 개발된 방법은 null message를 메시지를 내보낼 때마다 발생시키는 것이 아니고 필요한 경우에만 발생하는 것이다[8]. 그리고 새로운 형식의 null message를 도입하여 null message를 줄이는 방법을 제안하였다. 이 방법은 궁극적으로 좋은 lookahead를 계산해서 null message발생을 줄이는

방법을 택하고 있다[1]. 이 방법은 carrier null message (CNM) method라고 한다. Carrier null 메시지에는 기존의 null 메시지의 정보 외에 메시지가 전달된 프로세스의 이름과 사건 리스트의 최초 사건의 시간표의 시각들이 기록된다. 그래서 각 프로세스들은 추가된 정보를 이용하여 시뮬레이션 시계를 빠르게 진행시킬 수 있다.

우선 carrier null message method에서는 carrier null message라는 새로운 메시지를 도입한다. 메시지의 형태는 다음과 같다.

(t, route.inf, la.inf, carrier.null)

여기서 t는 time stamp, route.inf는 현재까지 carrier null message가 어떤 경로를 거쳐 왔는가를 저장한다. 그리고 la.inf는 지금까지 여러 LP를 거쳐 오면서 여러 정보를 수집하면서 갱신된 lookahead 정보를 나타낸다. 이 방법에서는 위에서 설명한 새로운 메시지를 필요한 경우 새로 발생시키고 여러 LP를 거치는 동안 메시지의 정보들을 갱신한다. 그리고 메시지의 가치가 없다고 판단되면 carrier null message를 삭제 하기도 한다. 이 방법에서 carrier null message를 다루기 위해 크게 두가지 단계를 거친다. 한가지는 carrier null message를 새로 만들거나 기존 메시지의 la.inf를 갱신하는 단계이고 다른 한가지는 route.inf을 갱신하거나 메시지의 가치를 판단하여 유지할 가치가 없다고 판단될 때 메시지를 삭제하는 단계이다.

또 De Vries[15]는 시뮬레이션 시스템을 나타내는 directed 그래프를 feedforward 네트웍과 feedback 네트웍들로 분리하였다. 그리고 각각의 경우에 대해서 Chandy-Misra 알고리듬에서의 null 메시지의 수를 줄이기 위한 알고리듬을 제시하였다.

3. 개선 알고리즘

분산 시뮬레이션은 대상으로 하는 시스템의 성격에 따라 시간 동기화를 위해 보수적 알고리즘을 적용할 것인지 낙관적 알고리즘을 적용할 것인지 결정해야 한다. 예를 들면 교착상태가 빈번하게 발생할 때 보수적 알고리즘을 적용하면 그 해결을 위해 과도한 null message가 사용됨에 따라 시스템 부하 및 시뮬레이션 수행 시간이 매우 많이 소요된다. 그러므로 이러한 시스템에는 낙관적 알고리즘을 적용하는 것이 효과적이다[12].

3.1 기호 정의

본 논문에서 사용하게 될 기호에 대한 정의는 다음과 같다.

STQ*i* : LP*i*의 이벤트 리스트에 있는 이벤트의 time stamp중에서 가장 작은 값

LT*i* : LP*i*에서 가장 최근에 내보낸 이벤트의 time stamp

IL*i* : LP*i*에 메시지를 보내는 모든 LP들의 인덱스 집합

NOM*i* : LP*i*로부터 받은 메시지의 개수를 저장하는 변수

BN : process blocking 여부를 판단할 때 사용하는 변수

SM*i* : LP*i*에서 진행시킨 메시지의 개수를 저장하는 변수

3.2 Null message를 줄이기 위한 개선된 방법

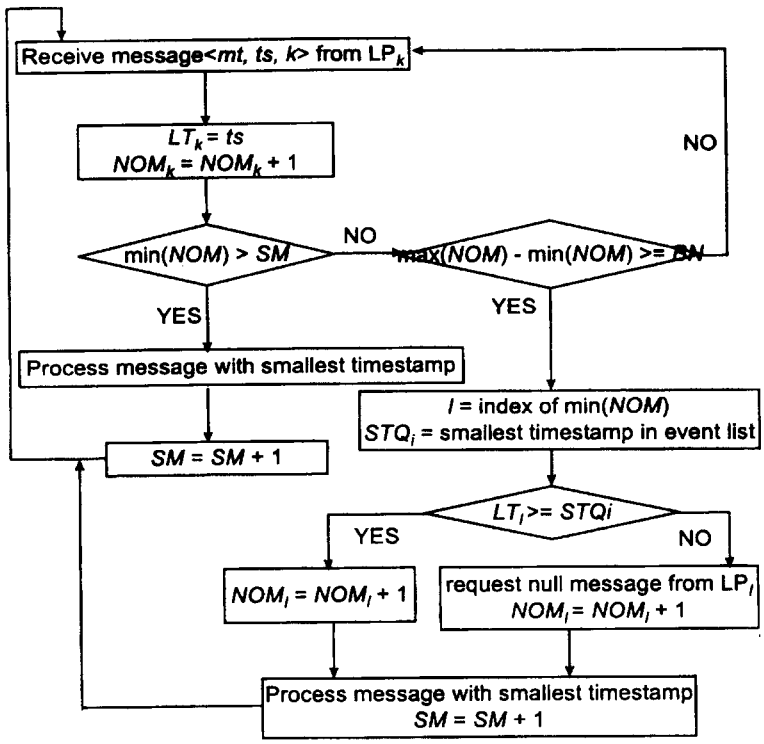
보수적 알고리즘에서는 input waiting rule에 의해 불필요한 blocking이 많이 발생한다. 기존 알고리듬에서 문제점으로 지적된 부분에 대하여, 본 연구에서는 LP에 다른 정보들을 포함시킴으로써 null message를 줄일 수 있는 방안을 제시한다.

우선 본 논문에서 제안하는 알고리즘에 대한 단계들을 기술하고, <그림 2>의 LP network으로 예를 들어 알고리즘을 자세히 설명한다.

단계 1 : 메시지를 받아들임.
(message <mt, ts, k> from LPk)

단계 2 : 특정 LP에서 받아들인 메시지의 개수와 time stamp에 대한 정보를 갱신.
(LTk = ts, NOMk = NOMk + 1)

단계 3 : 받아들인 메시지의 개수에 대한 값 중 최소값과 프로세스한 메시지 개수를 비교함. 그



<그림 2> BN 알고리듬의 Flow chart

래서 min(NOM) > SM 이면 단계 4로 가고, min(NOM) ≤ SM이면 단계 5로 감.

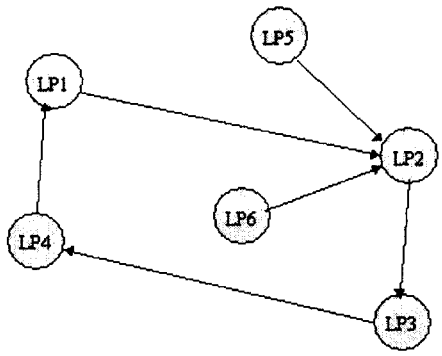
단계 4 : time stamp가 가장 작은 메시지를 진행시키고 프로세스한 메시지 개수를 갱신하고, 단계 1로 감.
(Process message with smallest timestamp, SM = SM + 1)

단계 5 : 받아들인 메시지 개수의 최대값과 최소값의 차이를 구해서 blocking notifier보다 크면 단계 6으로 가고, 그렇지 않으면 단계 1로 감.
(max(NOM) - min(NOM) >= BN)

단계 6 : blocking 된 LP를 찾아내고 그 LP에서 최근에 보낸 메시지의 time stamp (LTi) 와 이벤트 리스트에 있는 이벤트들의 time stamp 중 최소값(STQi)을 비교함.
(I = index of min(NOM), STQi = smallest timestamp in the event list)
LTi >= STQi 이면 단계 7로 가고, LTi < STQi 이면 단계 8 로 감.

단계 7 : 받아들인 메시지의 개수를 갱신하고 프로세스한 메시지 개수를 갱신하고, 단계 1로 감.
(NOMi = NOMi + 1)

단계 8 : 해당 LP로 부터 null message를 요청함. 받아들인 메시지 개수를 갱신하고 프로세스한 메시지 개수를 갱신함. 단계 1로 감.
(request null message from LPi, NOMi = NOMi + 1)
(Process message with smallest timestamp, SM = SM + 1)

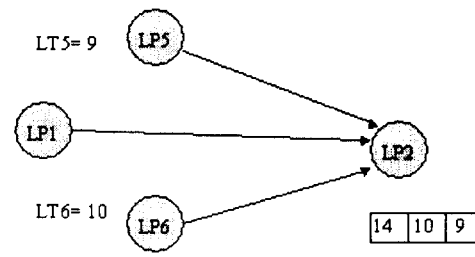


<그림 3> LP network

<그림3>으로 표현되는 시스템에 현재까지 제시된 보수적 알고리즘을 적용한다면 LP2는 LP1, LP5, LP6에서 모두 메시지를 받아야 안전한 이벤트를 결정할 수 있고 이를 진행시킬 수 있다. 이런 상태는 LP1, LP5, LP6의 몇 가지 정보를 LP2에서 알 수 있다면 이벤트를 기다리는 시간을 줄이고 blocking빈도를 줄여 줄 수 있다. 예를 들어 모든 incoming LP에 대해 최근에 보낸 메시지의 time stamp를 저장하고 있고 이 정보를 참조한다면 많은 경우는 불필요하게 모든 LP에서 메시지를 기다릴 필요가 없어질 것이다.

이렇게 하기 위해서 각 LP에는 이벤트를 보내는 모든 LP로부터 받은 메시지 개수를 저장하는 변수가 필요하고 어떤 LP로부터 최근에 받은 메시지의 time stamp를 저장하는 변수가 필요하다. 여기서는 앞에서 언급했듯이 메시지의 개수를 저장하는 변수를 NOM라고 표시하고 최근의 time stamp를 저장하는 변수는 LT라고 표시한다. LP로부터 이벤트를 받을 때 마다 NOM값을 하나씩 증가 시켜 나가고 LT의 값을 갱신한다. 보수적 알고리즘의 input waiting rule에 의하면 모든 incoming LP로부터 메시지를 받아야 안전한 이벤트를 찾아낼 수 있다. 이것은 NOM의 값이 모든 incoming LP에 대해 NOM이 모두 같을 때 안전한 이벤트를 찾아낼 수 있다는 의미로 해석할 수 있다. 그러므로 NOM이 가장 작은 LP에서 메시지를 기다리고 있다라고 말할 수 있다. 이렇게 NOM을 증가 시키는 것은 기존의 메시지 형태로는 어렵다. 들어온 메시지가 어떤 LP에서 왔는지 모르기 때문이다. 그러므로 여기서는 기존의 메시지에 어떤 LP에서 왔는지 표시하는 하나의 정보를 더 포함하게 한다. 즉 *(mt, ts, from)*이라는 형식의 메시지를 보낸다. 여기서 mt는 메시지 종류를, ts는 메시지의 time stamp를, 그리고 from은 어떤 LP에서 보낸 메시지인지 나타내는 것이다. 이 정보를 이용해서 NOM을 증가시키면 어떤 LP로부터 blocking되었는지를 알 수 있다.

다음에는 <그림 4>를 예로 들어 앞에서 언급한 방법을 구체적으로 설명한다. 그림 3.2는 LP5, LP6에서 메시지를 받은 상태이고 LP1에서는 아직 메시지를 받지 못한 상황을 나타내고 있다.



<그림 4> 메시지를 기다리는 LP

<그림 4>에서 LP2에서 NOM1은 0, NOM5은 2, NOM6이 1이라고 하면 LP2는 현재 LP1에서 메시지를 기다리고 있다고 판단할 수 있다. 이때 NOM을 비교하는 시점을 결정하는 것이 중요하다. 메시지를 받을 때 마다 NOM을 비교해서 null message를 요구한다면 기존의 방법과 큰 차이가 없을 것이다. 그러므로 여기서는 NOM의 최대치와 최소치를 계산해서 둘의 차이가 BN이상이면 blocking상태로 판단하고 LT와 SQT를 비교한다.

BN = 3으로 설정된 경우, 위의 예에서 LT1 = 15라면 보수적 알고리즘의 output waiting rule에 의해 LP1에서는 이 시간 이후 15보다 작은 time stamp를 가진 메시지가 절대 올 수 없으므로 LP1의 메시지는 기다릴 필요가 없이 time stamp 9를 가진 메시지를 진행시킬 수 있게 된다. 이때 NOM1은 1을 증가 시킨다. 이렇게 되면 3개의 LP로부터 모두 하나씩은 메시지를 받은 상황이 된다. 그러므로 time

stamp를 비교하여 안전한 이벤트를 결정할 수 있다. 하지만 LT1 = 7이라면 이후 LP1에서 time stamp 9보다 작은 메시지를 보낼 수도 있으므로 LP1에서 메시지가 도착하기를 계속 기다려야 한다. 이것을 위에서 정의한 기호를 이용해서 표시하면 다음과 같다.

{
If (STQ*i* ≥ LT*j*) then
    이벤트를 진행시키지 못함
else
    LP*j* 에서는 메시지를 기다릴 필요 없음
}
(※ 여기서 j 는 *ILi*중에서 메시지를 받지 못한 *LPj*의 인덱스)

위에서 제시한 알고리즘은 기존에 개발된 알고리즘에 비하면 평균 waiting time이 줄어들고 LP가 blocking될 빈도가 낮아지므로 그만큼 deadlock발생 빈도도 줄어들게 된다. 하지만 시뮬레이션 수행 도중에 LP는 몇 가지 정보를 갱신하므로 그만큼 시뮬레이션 오버헤드가 증가하게 된다. 예를 들어 시뮬레이션을 시작하기 전 LP는 incoming LP들을 리스트로 가지고 있어야 한다. 그리고 시뮬레이션을 수행하는 중에 time stamp에 대한 정보를 주고 받을 때 걸리는 시간이 추가된다.

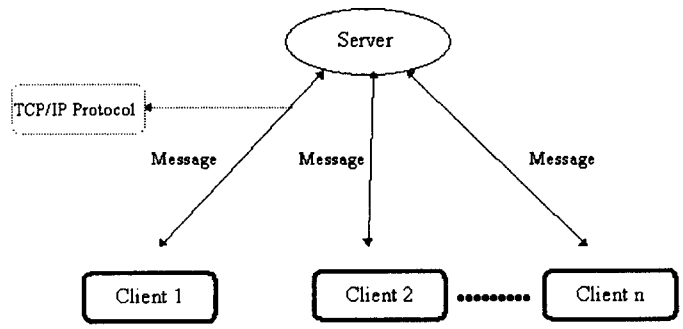
STQi ≥ LTj 인 경우에는 안전한 이벤트를 결정하기 위해 LTj로부터 null message를 요구해야 한다. null message를 받으면 NOM의 값을 1 증가 시킨다. 그렇게 해서 안전한 이벤트를 결정할 수 있다.

위에서 제시한 알고리즘은 기존의 메시지에 다른 정보를 포함하고 메시지 개수를 파악하여 blocking여부를 파악했다. 그리고 LT와 STQ를 비교해서 메시지를 계속 기다릴지 여부를 결정한다.

4. 실험 및 결과 분석

4.1 실험 환경

분산 시뮬레이션은 여러 대의 컴퓨터에서 병렬로 시뮬레이션을 진행하므로 컴퓨터들 간에 메시지를 주고받는 것이 필요하다. 즉, 각 프로세스의 출력을 다른 프로세스들로 전달하거나 전달받는 시스템이 필요하다. 본 연구에서는 이를 위해 LAN환경에서 네트워크 환경을 구축하였다. 여기서 구현한 네트워크 topology는 서버/클라이언트 형태를 채택하였다. 클라이언트에서 클라이언트로 보내지는 메시지는 모두 서버를 통해서 전달 되게 된다. 이렇게 함으로써 네트워크 내에서 발생하는 모든 메시지를 알 수 있고 메시지에 대한 정보를 수집할 수 있다. 즉, 어떤 한LP에서 다른 LP로 얼마만큼의 메시지가 전달되었는가, 또는 보내진 null message의 개수 등에 대한 정보를 수집할 수 있다. 네트워크 환경의 프로그램은 Visual C++의 클래스 라이브러리인 MFC를 이용하였다. MFC에는 네트워크 프로그램을 하기 위한 클래스와 이와 관련된 많은 함수들을 포함하고 있다. 그러므로 네트워크 프로그램을 구축하는데 매우 유용한 라이브러리이다. <그림 5>에서는 구축한 네트워크 환경을 도식적으로 보여 준다.



<그림 5> 네트워크 환경

본 실험에서는 위의 실험환경을 구축하고 여러 가지 LP네트워크를 구성하여 기존의 null message 방법과 여기서 제안한 알고리즘을 비교하였다.

4.2 수행도 평가

분산 시뮬레이션에서 수행도에 대한 평가 척도가 여러 가지가 있을 수 있다.

우선 보수적 알고리즘과 낙관적 알고리즘을 비교하기 위해 시뮬레이션 수행 속도를 들 수 있다. Lipton과 Mizell[16]은 롤백(rollback)에 소요되는 시간이 상수라고 가정할 때 Time Warp 알고리듬과 Chandy-Misra 알고리듬의 성능을 비교하였다. 그 결과 Time Warp알고리듬의 성능은 Chandy -Misra 알고리듬의 성능보다 얼마든지 좋은 경우가 있지만, Chandy-Misra 알고리듬의 성능은 Time Warp 알고리듬의 성능보다 최상의 경우에도 상수배 만큼 좋다는 것을 보였다. 여기서 만약 교착상태가 비교적 적게 일어난다면 보수적 알고리듬이 속도면에서 보다 나은 수행도를 보인다. 그리고 다른 척도로는 메모리 관리 문제를 들 수 있다. 특히 낙관적 알고리즘은 rollback을 대비해서 진행한 이벤트에 대한 정보를 저장하므로 메모리 관리 문제가 가장 중요한 수행도 척도가 된다. 보수적 알고리즘에서는 교착상태가 발생하게 되는데 이를 위해 null message를 발생하는 방법이 있다. 앞에서 언급했듯이 과다한 null message를 발생시킴으로써 통신사의 과부하를 증가시킨다. 그러므로 null message의 개수를 어느 정도로 줄일 수 있느냐가 보수적 알고리듬의 수행도 평가의 중요한 척도가 된다.

본 논문에서는 LT와 STQ를 이용하여 LP가 blocking되는 빈도를 줄이면서도 그와 동시에 null message를 필요할 때만 발생시키는 알고리즘을 제시하였으므로 시뮬레이션 수행 시간과 발생되는 null message의 개수를 기준으로 다른 알고리즘과 비교하였다.

4.3 실험 결과

다음은 8가지의 다른 LP 네트워크에 대해 기존의 알고리즘과 이 논문에서 제안하는 알고리즘으로 같은 시뮬레이션을 수행했을 때 수행도를 평가한 것이다. 표 4.1은 시뮬레이션을 수행하는데 걸리는 시간을 비교한 것이다. 본 논문에서 제안하는 방법(BN)을 실험할 때는 BN 변수, 즉 blocking여부를 판단하는데 참조하는 변수 값의 4가지 경우에 대해 각각 실험해 보았다.

<표 1> 시뮬레이션 수행 시간

CNM	BN algorithm			
	BN=2	BN=3	BN=4	BN=5
1156.1	1458.2	1358.3	1341.2	1547.5
1482.7	1756.4	1563.2	1357.3	1459.4
2980.1	3562.2	2843.5	3254.2	2958.2
4232.8	1954.1	2114.5	2423.6	2236.4
5154.2	6352.3	4824.6	5246.7	7452.6
5871.9	7249.5	5962.6	7125.9	6984.2
8564.7	10025.3	9658.2	9965.2	8821.4
9562.6	11236.4	10236.5	9423.3	9845.7

<표 1>에서 보면 BN이 어떤 값을 가질 때 수행시간이 가장 적게 소요 된다라고 말할 수 없다. 즉, 시뮬레이션 하는 시스템의 구성 및 상황에 따라 다르다. 그러나 현재는 어떤 시스템에서 어떤 BN이 가장 좋다라고 판단할 수 있는 방법이 없으므로 여러 가지 BN값에 대해 실험을 해보아야 한다.

<표 2>는 두 가지 방법에 대해 발생되는 null message개수를 나타낸다. 여기서도 마찬가지로 4가지의 BN변수에 대해 각각 실험해 보았다.

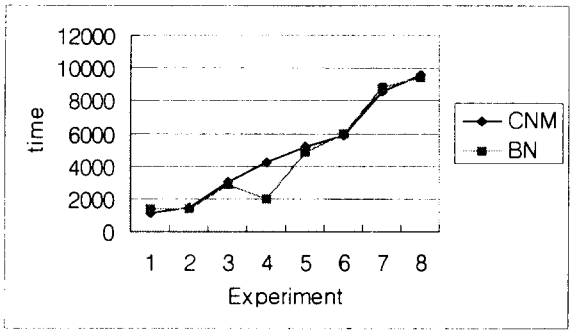
<표 2> 발생된 null message 개수

Exp. No	CNM	BN algorithm			
		BN=2	BN=3	BN=4	BN=5
1	72	95	81	102	145
2	115	123	84	94	89
3	165	154	162	147	123
4	195	213	163	182	194
5	559	584	521	482	536
6	652	635	645	583	712
7	542	598	754	652	537
8	725	951	784	852	767

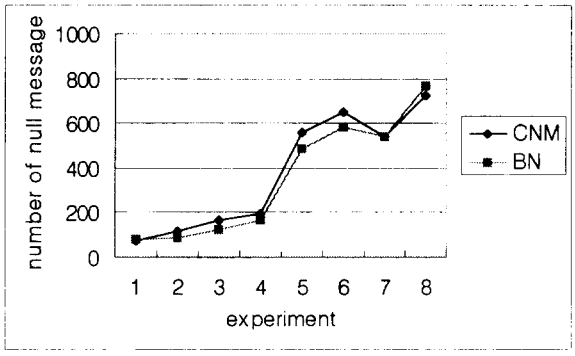
<표 2> 에서 보면 각 실험에서 BN값에 따라 발생된 메시지 개수가 다르다는 것을 알 수 있다. 그리고 서로 다른 시스템에서 좋은 BN값을 쉽게 결정할 수 없다. 그러므로 효율적인 BN값을 선택하는 방법을 연구하는 것이 필요할 것이다.

<그림 6>에서는 CNM방법으로 실험했을 때 소

요되는 시간과 BN 방법으로 실험했을 때 가장 좋은 결과로 두 가지 방법을 비교한 그래프이다. 여기서 BN 방법의 가장 좋은 결과를 택한 이유는 해당 실험에서 최적의 BN값을 미리 결정할 수 없기 때문이다. 그리고 <그림 7>은 발생된 null message개수에 대해 비교한 그래프이다.



<그림 6> 시뮬레이션 수행 시간의 비교



<그림 7> 발생된 null message 개수의 비교

위의 실험 결과로 보면 이 논문에서 제시한 알고리즘이 시뮬레이션 수행 시간은 기존의 알고리즘과 비슷하다. 그러나 LT와 STQ를 비교해서 null message를 요구하기 때문에 기존 알고리즘에 비해 null message의 개수가 적은 것을 알 수 있다. 그러나 이 논문에서 제안하는 알고리즘에서 두 가지 수행도, 즉 시뮬레이션 수행 시간과 null message개수에 대해 모두 좋은 결과를 나타내는 BN을 결정하는 것에 난점이 있다. 그러므로 이 알고리즘을 적용하기 위해서는 시스템에서 중요한 요소가 무엇인지 잘 파악한 후 사용해야 할 것이다.

5. 결 론

분산 시뮬레이션에서는 여러 컴퓨터에서 동시에 시뮬레이션을 수행하기 때문에 시간 동기화가 중요한 문제가 된다. 시간 동기화를 위해 보수적 알고리즘과 낙관적 알고리즘이 개발되었는데 보수적 알고리즘에서는 두 가지 규칙에 의해 교착상태가 발생한다. 교착상태를 해결하기 위한 방법으로 null message방법이 제안되었는데 이 방법에서는 null message 개수를 줄이는 것이 매우 중요하다. 이 논문에서는 각 LP가 여러 가지 정보를 저장함으로써 null message를 줄이는 방법을 제안하였다. 실제 네트워크 환경에서 실험을 수행했을 때 기존의 알고리즘보다 적은 수의 null message를 발생시킨다는 것을 확인할 수 있었다.

제시한 알고리즘에서 BN이라는 변수를 정의하였다. 본 논문에서는 임의의 수를 입력하여 실험을 수행하였다. 그러나 BN을 어떻게 설정하느냐에 따라 수행도에 영향을 미칠 것이다. 예를 들어 BN을 너무 작게 설정하면 null message를 매우 많이 발생시킬 것이고 너무 크게 설정하면 잠깐동안 교착상태에 빠지므로 시뮬레이션 수행시간이 증가할 것이다. 그러므로 시스템의 특성에 맞는 효율적인 BN을 결정하는 문제가 추후 연구 과제로 남는다.

참고 문헌

[1] Cai, W. and S.J.Turner, "An Algorithm for Distributed Discrete-Event Simulation - The Carrier Null Message Approach", *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, Vol. 22, No. 1, 1990, pp. 3-8.

[2] Chandy, K. M. and J. Misra, "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations", *Communications of the ACM*, Vol. 24, No. 4, 1981, pp. 198-205.

[3] Chandy, K. M. and J. Misra, "Distributed Deadlock Detection", *ACM Transactions on Computer Systems*, Vol. 1, No. 2, 1983, pp. 144-156.

[4] Fujimoto, R. M., "Parallel Discrete Event Simulation", *Communications of the ACM*, Vol. 33, No. 10, 1990, pp. 31-53.

[5] Fujimoto, R. M. and D. M. Nicol, "State of the Art in Parallel Simulation", *Proceedings of 1992 Winter Simulation Conference*, 1992, pp. 246-254.

[6] Jefferson, D. R., "Virtual Time", *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, 1985, pp. 404-425.

[7] Lin, Y. and P. A. Fishwick, "Asynchronous Parallel Discrete Event Simulation", *IEEE Transactions on Systems, Man and Cybernetics, Part A*, Vol. 26, No. 4, 1996, pp. 397-412.

[8] Misra, J., "Distributed Discrete-Event Simulation", *ACM Computing Surveys*, Vol. 18, No. 1, 1986, pp. 39-65.

[9] Nicol, D. M. and R. M., Fujimoto, "Parallel Simulation Today", *Annals of Operations Research*, Vol. 53, 1994, pp. 249-286.

[10] 서동욱, 전진, 김성식, "혼합시뮬레이션에서의 인과관계 오류 해결방안", 「한국시뮬레이션학회 논문지」, 제4권, 제2호, 1995년 4월, pp 31-40

[11] 옥시건, 정창성,"Recursive Feedforward Network상에서의 효율적인 병렬 시뮬레이션 알고리즘", 「한국시뮬레이션학회 논문지」, 제4권, 제2호, 1995년 4월, pp 79-92

[12] Richard M. Fujimoto, "Parallel and Distributed Simulation", *Handbooke of Simulation* edited by Jerry Bank, John Wiley & Sons, Inc., U.S.A., 1998, pp 429-464.

[13] Edwin Naroska and Uwe Schwiegelshohn, "Conservative Parallel Simulation of a Large Number of Processes", Simulation, Special Issue: Parallel and Distributed Simulation, Vol. 72, No. 3, March 1999, pp 150-162.

[14] 성영락, 김탁곤, 박규호, "병렬분산 이산사건 시뮬레이션", 「정보과학회지」 특집: 이산사건 시뮬레이션, 제13권 제4호, 1995년 4월, pp 20-32.

[15] Ronald C. De Vries, "Reducing null messages in misra's distributed discrete event simulation method", *IEEE Transactions on Software Engineering*, Vol. 16, No. 1, Jan., 1990, pp 82-91.

[16] R.Lipton and D. Mizell, "Time warp vs. Chandy-misra: A worst-case comparison", *Distributed simulation*, Vol. 22, 1990.

저자소개

이영해

1977년 고려대학교 산업공학, 학사
1983년 Univ. of Illinois, 산업시스템공학, 석사
1986년 Univ. of Illinois, 산업공학 및 경영과학, 박사
경 력 한국시뮬레이션학회, 대한산업공학회, 한국경영과학회, 대한설비관리학회 이사, Purdue Univ., 오사카대학 방문교수, 대우중공업(주)
1986년~현재 한양대학교 산업공학과 교수
1995년~현재 한국시뮬레이션학회 부회장
관심 분야 : Web-Based Simulation, Simulation Output Analysis, Simulation Optimization

배강용

1996년 한양대학교 산업공학과 학사
1998년 한양대학교 산업공학과 석사
관심 분야 : 분산 및 병렬 시뮬레이션