혼합시뮬레이션에서의 인과관계 오류 해결방안*

A Causality Error Prevention Scheme In The Hybrid Simulation

서 동 욱**, 전 진**, 김 성 식**
Dong Wook Seo, Jin Jun, Sung Shick Kim

Abstract

A hybrid simulation model consists of real physical entities as well as simulated ones. It also contains logical processes for decision making for each operation units, a group of the entities. During the execution of such simulations, the physical and the logical processes consume real clock time while the activity durations of the simulated ones are generated. Due to the inherent chracteristics of the subjects of the hybrid simulation, each of the operation units are distributed over a network of communication channels. Since one can not undo an real event already taken place, the traditional central clock approach is used for the synchronization of the events(Kim[6]). However, there are still chances of causality errors due to the randomness in the communication delays. This error is not found in the distributed pure simulations. This paper explains the error in details and proposes a prevention scheme that is simple to implement.

1. 서 론

컴퓨터 통합환경의 자동화 공장을 구축하는 한 방법으로 컴퓨터 네트워크와 운용 소프트웨어는 공장에서 가동될 실제시스템을 사용하고, 이들보다 값이 비싸고 단지 명령에 따라 움직이는 물리적 기계장비는 시뮬레이션으로 대체하여 실제에 준하는 자동화 공장을 먼저 구축하는 방법(김성식과 배경한[1], 배경한[3])이 있다. 이 방법을 이용하면 장비구축 및 관련 소프트웨어 개발단계, 조정단계 등에서 걸리는 많은 시간과 노력을 줄일 수 있으며, 먼저 구축된 컴퓨터와 운용소프트웨어만의 실제에 준하는 시스템에 실제 기계를 하나씩 결합하여 검토함으로써 실제적인 자동화공장 구축을 용이하게 달성할 수 있다.

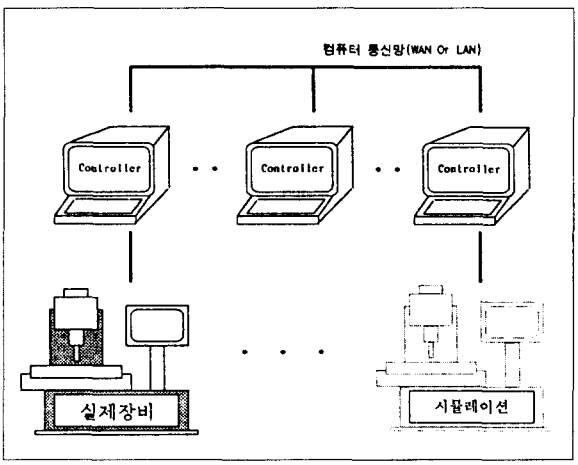
이렇게 물리적 기계장비는 시뮬레이션으로 대체하고 그 외의 컴퓨터, 통신장비와 운용소프트웨어는 실제공장에 설치할 것으로 구성하는 혼합 시뮬레이션은 설치할 시스템의 수행척도를 측정한다는 시뮬레이션의 일반적 용도뿐만 아니라 건설될 시스템이 의도대로 작동하는가를 검토하고 보증하는 용도로도 사용된다. 이러한 실행시스템에서는 구성요소의 일부분은 실제 기계이고 나머지는 시뮬레이션으

* 본 연구는 한국과학재단의 목적 기초연구의 일환으로 수행되었음.

** 고려대학교 산업공학과

로 대체되어 처리되어, 여기서는 물리적인 실제시간을 소요하면서 진행되는 작업(실제작업이라 정의한다.)과 시뮬레이션이 혼재하는 형태가 나타나게 된다(그림 1). 또한 운용 소프트웨어에서 처리되는 의사결정과 장비 상태변수의 갱신등 실시간이 소요되는 작업 또한 실시간을 소모하므로 실제 작업으로 고려되어야 한다.

이와 같은 시뮬레이션은 실행시 분산된 프로세스에서 실제작업과 시뮬레이션 사건이 혼합되어 전개되는 특성을 가지므로 이 둘의 진행을 동기화해야 하는 새로운 문제가 제기된다. 배경한[3]은 이러한 시뮬레이션을 그 특성을 반영하여 혼합분산 시뮬레이션이라 정의하였다.



〈그림 1〉 실제 작업과 시뮬레이션의 혼합

분산시뮬레이션은 각자 자신의 사건을 처리하는 한정된 수의 프로세스와 이들을 연결하는 채널로 구성되며, 각 프로세스는 채널을 통하여 메시지를 교환함으로써 전체 시뮬레이션을 진행한다. 그리고 메시지는 처리되어야 할 사건의 내용과 그 시간정보를 담게 된다(Chandy[3]). 어떤 프로세스가 마지막으로 사건을 처리한 시간을 t라 하자. 이때 다른 프로세스로부터 t보다 빠른 시간정보를 가진 메시지를 전달받게 된다면 현재시점에서 이미 지나간 과거의 사건을 처리해야 하는 오류에 빠지게 된다. 이것은 받아들일 수 없는 오류이며, 분산시뮬레이션에서는 이러한 오류를 인과관계 오류(causality error)라 부른다. 기존의 분산시뮬레이션 연구들에서는 이 인과관계 오류를 해결하는 방안으로 보수적 방법(conservative approach)과 낙관적 방법(optimistic approach)을 제시하고 있다(Chandy & Misra [3], Fujimoto[4], Jefferson[5], Misra[7]).

그러나 혼합분산 시뮬레이션에서는 실제와 시뮬레이션의 혼합이라는 특성으로 인하여 기존의 분산시뮬레이션 기법을 직접 사용할 수 없게 된다. 때문에 김성식과 배경한([1] [2] [6])은 중앙시계를 이용하여 실제작업과 시뮬레이션 사건을 동기화하는 방법을 제시하였다.

그러나 중앙시계를 이용한 동기화 방법을 사용한다 하더라도 중앙시계와 분산된 프로세스간의 통신과정에 의하여 발생할 수 있는 프로세스의 진행상황과 중앙시계의 상황인식간의 차이는 그 결과로서 또 다른 인과관계 오류를 초래하게 된다.(이 문제는 3장에서 자세히 서술한다.)

본 논문의 목적은 혼합분산 시뮬레이션의 성격과 특징을 분석한데 기초하여, 그 특성으로부터 발생하는 동기화 과정에서의 인과관계 오류의 원인과 그 해결방안을 제시하는 것이다.

2장에서는 먼저 혼합분산 시뮬레이션의 특성과 구성, 그리고 중앙시계를 이용한 동기화방법(Kim[6])을 검토하고, 3장에서는 중앙시계를 이용한 혼합분산 시뮬레이션에서의 인과관계 오류의 원인과 그 해결방안을 제시하였다. 그리고 4, 5장에서는 결론과 추후 연구방향, 참고문헌에 대해 서술하였다.

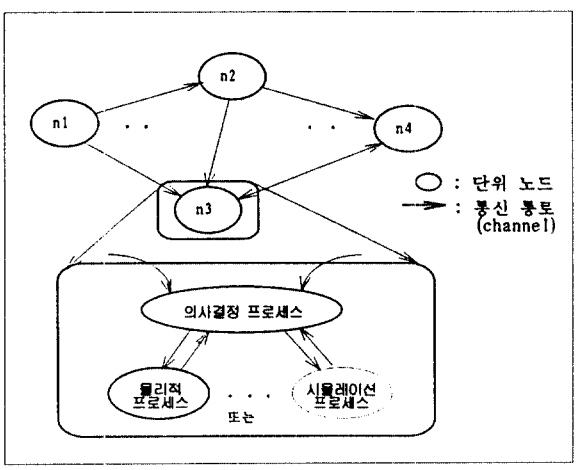
2. 혼합분산 시뮬레이션과 중앙시계를 이용한 동기화

2.1 프로세스의 구성 및 작동

혼합분산 시뮬레이션의 대상은 하나 또는 그 이상의 물리적인 실체들로 이루어진 시스템이다. 여기에서 일부의 물리적인 실체들은 그대로 사용하고 나머지의 행동은 시뮬레이션으로 표현한다. 물리적인 실체의 행동은 의사결정자에 의하여 지시되며, 그 결과는 다시 의사결정자에게 전달되어 의사결정자료로 사용된다. 하나의 의사결정자는 하나 또는 복수의 실체의 행동을 지시·통제하며, 의사결정자간에도 정보를 교환하고, 교환된 정보들은 물리적 또는 시뮬레이션으로 표현된 실체에서 오는 정보와 마찬가지로 의사결정에 사용된다. 한단위에는 전부가 물리적 실체이거나 시뮬레이션으로 표현된 실체일 수도 있고, 섞여 있을 수도 있다. 따라서 혼합분산 시뮬레이션은 하나 또는 여러개의 상호통신이 가능한 분산된 단위들로 구성되

고, 각 단위는 의사결정을 하는 의사결정 프로세스와 물리적으로 움직이는 물리적 프로세스 그리고, 물리적 실체를 대신하는 시뮬레이션 프로세스를 가질 수 있다(Kim [6]). 이러한 시스템의 프로세스 구성은 〈그림 2〉에 표현되어 있다. 그림에서 각 단위는 노드로 표현하였으며, 화살표는 의사교환 채널을 표시한다.

그러면, 〈그림 2〉의 시스템이 어떻게 작동하는지 살펴보자. 각 단위 노드는 의사결정 프로세스를 가지며, 의사결정 프로세스는 다른 단위노드의 의사결정 프로세스나 자기 단위노드의 물리적 프로세스로부터 메시지를 받아서 관련 변수들을 갱신하고 다음 행동을 결정한다. 의사결정 프로세스의 다음 행동은 두가지중 하나인데, 하나는 다른 단위노드의 의사결정 프로세스에게 메시지를 보내는 것이고, 다른 하나는 자기노드의 물리적 프로세스에게 작업명령을 내리는 것이다. 물리적 프로세스는 작업명령을 받게 되면 장비의 가동을 시작하고 작업결과를 의사결정 프로세스에게 전달하는데, 이때 물리적 실체가 존재하지 않는 경우에는 시뮬레이션 프로세스가 이를 대신하여 시뮬레이션으로 그 행태를 표현한다.



〈그림 2〉 시스템의 구성

2.2 혼합분산 시뮬레이션에서의 실제작업

혼합분산 시뮬레이션에서 물리적 실시간이 소요되는 실제작업에는 먼저, 물리적 실체가 결합되어 있는 단위에서 발생하는 실제 장비가 가동되는 작업이 있으며, 의사결정 단위에서 실제시간이 소요되는 작업이 있다(Kim[6]). 일례로, 다음 행동에 대한 의사결정이라든지, 관련 장비상태를 나타내는 변수의 갱신등은 실제의 상황에서도 같은 장비가 같은 일을 수행하므로, 이것은 실제의 시간이 드는 작업이다. 따라서 물리적 프로세스와 의사결정 프로세스는 모두 실제작업을 처리하는 프로세스이다.

이러한 이유로 혼합분산 시뮬레이션에서는 실제 장비의 작업뿐만 아니라 소프트웨어상에서의 실시간 소요작업도 실시간 진행에 반영해야 하며, 이들은 모의된 사건과 동기화시켜 같은 시계에서 표현되어야 한다.

2.3 중앙시계를 이용한 동기화

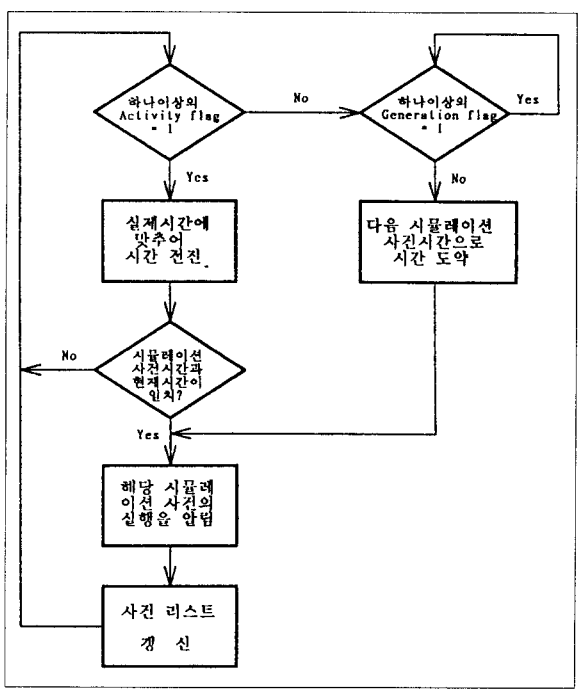
중앙시계는 실제작업과 시뮬레이션 사건을 동기화하는 역할을 한다. 이를 위하여 중앙시계는 각 단위노드에 있는 의사결정 프로세스와 물리적 프로세스의 상태를 표현하는 activity flag를 가진다. 이것은 각각 실행중일때는 1, 그렇지 않을 경우에는 0의 값을 가진다. 중앙시계는 또한 사건리스트를 가진다. 각 시뮬레이션 프로세스는 새로운 사건을 발생시킬때마다 자신의 사건리스트중 가장 빠른 시간을 가진 사건을 고른다. 이 사건시간은 중앙시계에 전달되며, 중앙시계는 이 사건시간을 어디에서 송신되었는가 하는 출원지에 대한 정보와 함께 사건리스트에 등록하여 모의된 사건의 발생시간을 보유하게 된다. 시뮬레이션 프로세스가 사건을 발생시키고 있는 도중임을 표현하기 위하여 generation flag가 사용된다. 이 flag는 해당 시뮬레이션 프로세스가 주어진 시점에 사건을 발생시키고 있을 경우에는 1, 그렇지 않을 때는 0값을 가진다. 새로 발생한 사건의 시간이 등록되어 있는 사건들보다 더 빠른 시간을 가질 수도 있기 때문에 이 flag가 1일 경우에는 모든 실제작업이 진행되지 않고 있더라도 시간도약은 금지된다.

〈그림 3〉에서 보듯이 실제작업과 시뮬레이션 사건을 동기화하기 위하여 중앙시계는 연속적으로 activity flag를 검사한다. 이때 한개라도 activity flag의 값이 1인 것이 있으면 시스템은 실제국면(Real Stage)에 들어서게 된다. 실제국면은 하나이상의 물리적 프로세스 혹은 의사결정 프로세스가 실행중인 국면으로서 이때 중앙시계는 컴퓨터내의 물리적인 실제시간에 맞추어 전진한다. 실제국면이 진행중이라도 사건리스트에 등록되어 있는 시뮬레이션 사건은 자신의 사건발생시간에 처리되어야 한다. 따라서 중앙시계는 자신의 시계가 가리키는 시간과 일치하는 시간을 가

지는 시뮬레이션 사건이 존재하는지에 대해서도 검사한다. 일치하는 사건이 발견되면, 해당 프로세스에게 그 사건의 처리를 지시하고 사건리스트를 갱신한다.

만약 실제작업이 진행중인 프로세스가 없다면 즉, activity flag가 모두 0값을 가지게 되면, 시스템은 순수한 시뮬레이션 국면(Simulation Stage)으로 접어들게 된다. 이때 현재 시뮬레이션 프로세스중에서 사건을 발생시키고 있는 프로세스가 없을 경우에는 즉, 모든 generation flag값이 0인 경우에는 중앙시계는 사건리스트중 가장 빠른 사건의 시간으로 도약하게 된다. 반면에 하나이상의 generation flag값이 1일 경우에는 도약은 금지된다. 이 경우에는 사건발생이 완료되고 사건리스트의 갱신이 끝난이후에(모든 generation flag = 0) 중앙시계의 사건리스트중 장 빠른 사건의 시간으로 도약하게 된다.

이와 같은 과정을 통하여 중앙시계는 실제작업과 시뮬레이션 사건을 동기화한다.



〈그림 3〉 중앙시계를 이용한 동기화

3. 혼합분산 시뮬레이션에서의 인과관계 오류와 해결방안

3.1 혼합분산 시뮬레이션의 특성과 인과관계 오류

혼합분산 시뮬레이션에서는 일반적인 분산시뮬레이션과 마찬가지로 분산된 프로세스에서 각자 자기 사건을 처리하며 시뮬레이션을 진행하는 특성을 가진다. 하지만 혼합분산 시뮬레이션에서는 실제와 시뮬레이션이 혼합되어 있는 특성으로 인하여 기존의 분산시뮬레이션 기법은 사용할 수 없고, 앞서 본 고전적인 중앙시계를 이용한 동기화 방법을 사용하게 된다.

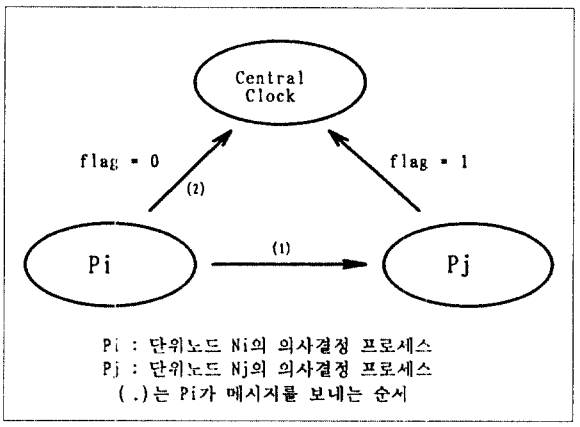
중앙시계를 이용한 동기화방법은 각 지역마다 시계를 따로 두지 않고 중앙시계라는 시뮬레이션 프로세스를 이용하여 모든 사건을 처리하게 된다. 따라서 이 방법에서는 기존의 분산시뮬레이션의 동기화 방법을 쓸 경우에 발생하는 인과관계 오류는 생기지 않는다. 그러나 중앙시계를 이용한 동기화 알고리즘은 다른 종류의 인과관계 오류를 발생시킨다. 분산된 프로세스들에서의 사건처리는 프로세스간의 메시지 교환을 통한 동기화를 필요로 한다. 일반적인 분산시뮬레이션에서 통신 메시지의 교환은 사건의 선후관계를 맞춰주기 위한 수단일 뿐이다. 따라서 통신에 소요되는 시간은 분산 시뮬레이션의 운영에서 아무런 문제를 발생시키지 않는다. 하지만 혼합분산 시뮬레이션에서는 전체 단위노드들의 상황을 총괄하여 시계를 운영하기 때문에, 각각의 단위노드에서 보내는 보고 메시지들의 도착시차로 인해 발생하게 되는 중앙시계의 그릇된 상황인식은 중요한 문제를 야기할 수 있다. 이로부터 초래되는 인과관계 오류는 일반적인 분산 시뮬레이션에서는 발생하지 않는 것으로서 혼합분산 시뮬레이션의 고유한 문제이다.

3.2 통신으로 인한 인과관계 오류

통신으로 인해 단위노드들에 있는 프로세스의 진행상황과 중앙시계의 상황인식간에 차이가 발생하게 되는 과정을 두 프로세스가 통신하는 경우의 예를 들어 구체적으로 살펴보자.

한 단위노드 N_I에서 어떤 작업(실제 혹은 시뮬레이션)중에 다른 단위노드 N_J로 메시지를 보낸후 자신의 작업을

종료하는 경우가 있다(N_i와 N_j이외의 다른 단위노드의 프로세스에서는 아무일도 하지 않고 있다고 가정한다.). 이때 N_i의 의사결정 프로세스 P_i는 N_j의 의사결정 프로세스 P_j에게 메시지를 보내고 곧이어 자신의 작업이 종료됐음을 중앙시계에게 보고(activity or generation flag = 0)한다. 또한 P_j는 P_i로부터 메시지를 전달받고 중앙시계에게 자기 단위노드에서 작업이 시작됨(activity or generation flag = 1)을 알린다(그림 4).



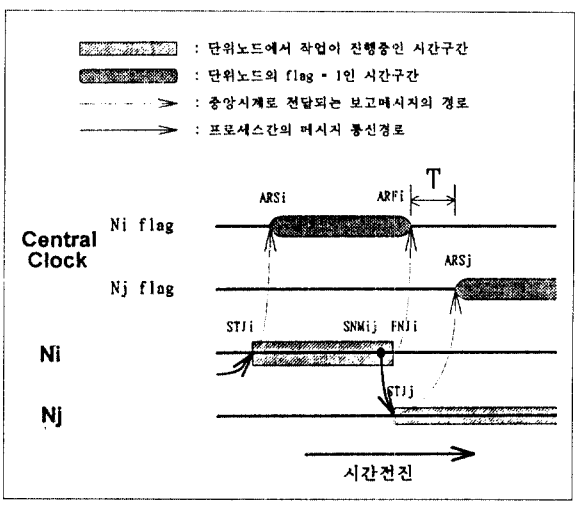
〈그림 4〉 단위노드에서의 작업종료와 작업시작 보고

이때 P_i에서 보낸 메시지를 P_j가 수신하여 시작되는 N_j에서의 작업은 N_i에서의 작업과 독립적으로 진행된다. 중앙시계는 이 상황을 두 단위노드에서 보낸 통신메시지를 통해서만 인식한다. 이 경우 P_i가 보낸 메시지와 P_j가 보낸 메시지가 중앙시계에 도착하는 시간은 실제로 행하는 통신지연시간에 의하여 결정되며, 이때 중앙시계는 두 단위노드의 진행상황에 대해 잘못된 인식을 하는 경우가 생길 수 있다.

〈그림 5〉는 잘못된 인식이 발생하는 과정을 시간의 전개에 따라 표현한 것이다. 그림의 문자가 표현하는 시점들은 다음과 같다.

STJ_i : N_i에서 작업을 시작하는 시점
ARS_i : 중앙시계가 P_i의 메시지를 받아 flag = 1로 인식하는 시점
SNM_{ij} : P_i가 P_j로 메시지를 보내는 시점
FNJ_i : N_i에서 작업을 종료하는 시점
ARF_i : 중앙시계가 P_i의 메시지를 받아 flag = 0으로 인식하는 시점
STJ_j : 메시지를 받아 N_j에서 작업을 시작하는 시점
ARS_j : 중앙시계가 P_j의 메시지를 받아 flag = 1로 인식하는 시점

그림에서 보듯이 실제로는 SNM_{ij}시점에 보낸 메시지로 인하여 N_i에서 작업이 종료된 후에 N_j에서 작업이 진행되는데, 이때 중앙시계로 FNJ_i시점에 보낸 P_i의 작업종료 메시지와 STJ_j시점에 보낸 P_j의 작업시작 메시지가 도착하는 시간차이(ARF_i와 ARS_j의 차이 = T)가 발생하게 되면 구간 T에서는 두 단위노드 모두에서 진행중인 작업이 없는 것으로 간주된다. 이렇게 되면, 중앙시계의 현재시간이 그림에서 T로 표현되어 있는 시간구간에 있고, 사건리스트에 등록된 사건중 가장 빠른 사건의 시간이 t(t〉ARF_i)일 경우, 중앙시계는 모든 flag = 0 이므로 어떤 작업도 진행중이지 않는 것으로 판단한 후에 t로 시간을 도약시키게 된다(〈그림 3〉의 동기화방법 참조). STJ_j시점에서 시작한 작업이 진행중임에도 불구하고 시계는 모의된 시간 t로 도약하기 때문에 이것은 명백한 잘못이며, 이는 결과적으로 인과관계 오류의 발생을 초래하게 된다.



〈그림 5〉 단위노드 진행상황에 대한 중앙시계의 인식과정

물론 일반적인 경우, 두개 이상의 여러 단위노드의 프로세스가 동시에 통신을 하며 진행된다. 그러나 여러개의 프로세스가 복잡하게 얽혀 통신을 한다고 하더라도 구체적으로 쪼개어 보면, 결국 두개의 프로세스가 메시지를 주

고 받는 과정에서 일어나는 작업상황의 변화를, 통신을 통해 중앙시계가 인식하는 과정에서 인과관계 오류가 발생하게 된다.

따라서 이러한 인과관계 오류의 발생을 방지하려면 다음 조건을 만족시켜야만 한다.

P_i에서 P_j로 보내는 메시지가 존재한다면,

$$ARF_i \geq ARS_j$$

즉, 메시지를 수신한 측의 작업시작 메시지는 적어도 메시지를 송신한 측의 작업종료 메시지보다 늦지 않게 중앙시계에 전달되어야 한다는 것이다.

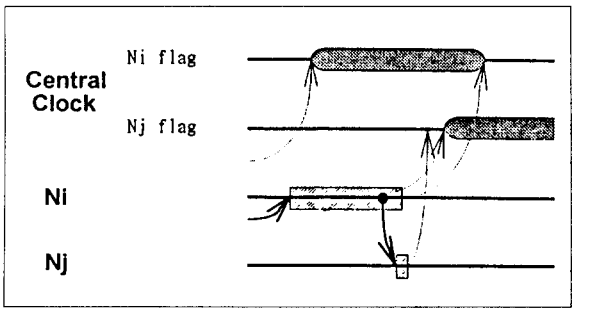
결론적으로, 중앙시계를 이용한 혼합분산 시뮬레이션의 동기화 방법에서는 위 조건을 만족시킴으로써 원치않는 시간도약이 일어나는 시간구간(〈그림 5〉의 T구간. 본 논문에서는 이 구간을 time-jump window라 정의한다.)의 발생을 막아야만 인과관계 오류를 방지할 수 있다.

3.3 통신지연시간의 임의성

해결방안에 앞서 고려해야 할 사항은 통신지연시간의 임의성이다. 단거리 통신망(LAN)이나 운용시스템의 통신채널에서 메시지 교환은 주어진 프로토콜을 따르게 되는데 어떤 방법을 사용하건간에 통신지연시간은 통신순간의 환경에 영향을 받는 확률변수가 되어 임의성을 띄게 된다.

문제는 통신지연시간의 임의성으로 인해 직접 중앙시계로 보내는 통신메시지가 한단계를 거쳐서 중앙시계로 가는 통신메시지보다 먼저 도달하리라는 보장이 주어지지 않는다데 있다.

일례로 time-jump window의 발생을 방지하기 위하여 작업을 종료하는 단위노드의 P_i가 P_j로 메시지를 보내면서 P_j의 작업시작 보고를 대신해 주는 방법을 생각할 수 있다. 이 방법은 앞의 $ARF_i \geq ARS_j$ 조건을 만족하게 되므로 time-jump window의 발생을 방지할 수 있다. 그러나 〈그림 6〉에서 보듯이 P_j가 P_i로부터 메시지를 받고 짧은 작업을 끝낸 후 중앙시계로 보내는 작업종료 메시지가 P_i의 P_j를 대신한 작업시작 메시지보다 빨리 도달하는 경우($ARF_j < ARS_j$)가 생길 수 있다. 즉, 통신지연시간의 임의성으로 인하여 한단계의 메시지가 두단계를 거치는 메시지보다 더 오랜 시간이 소요되는 경우가 발생하게 됨으로서 한 단위노드의 작업종료가 작업시작보다 먼저 인식되는 것이다. 이렇게 되면 N_j에서의 작업이 이미 종료되었음에도 불구하고 중앙시계에서는 계속 진행중인 것으로 오판하여 더이상 시뮬레이션이 진척되지 않음으로써 전체 시스템이 정지되는 결과를 초래할 수 있다.

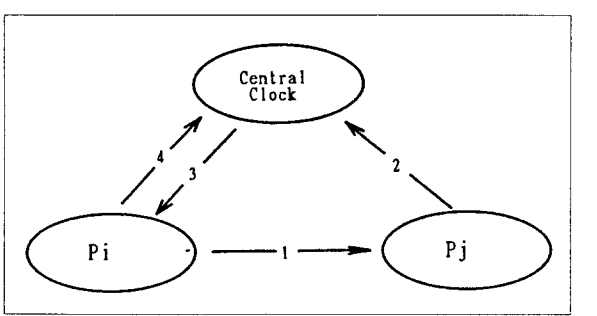


〈그림 6〉 통신지연시간의 임의성을 고려하지 못하는 경우의 문제

따라서 중앙시계를 이용한 혼합분산 시뮬레이션에서 인과관계 오류를 피하기 위해서는 time-jump window의 발생을 막기 위한 $ARF_i \geq ARS_j$ 조건을 만족함과 동시에 통신지연시간으로 인한 문제가 발생하지 않는 동기화 방안이 필수적이다.

3.4 통신으로 인한 인과관계 오류의 해결방안

본 논문에서 제기하는 해결방안은 〈그림 7〉에서처럼 자신이 보낸 메시지를 받게 되는 단위노드로부터 수신신호를 확인한 후에 자신의 작업종료를 보고하는 방법이다.



〈그림 7〉 메시지 통신순서

〈그림 8〉에서의 문자가 표현하는 시점은 기본적으로 〈그

림 5)에서와 같으며, 의미가 바뀌거나 첨가된 부분은 다음과 같다.

FNJ_i : N_i에서의 작업종료 시점
SNF_i : P_i의 작업종료 보고 시점
FNJ_j : N_j에서의 작업종료 시점
SNF_j : P_j의 작업종료 보고 시점

〈그림 8〉에서와 같이 P_i는 자기 단위노드에서 작업이 종료된 FNJ_i시점에 바로 중앙시계로 보고하지 않고 N_j로부터의 수신신호를 확인한 후(SNF_i)에 보고한다. 이때 ARF_i는 ARS_j와 SNF_i를 거친 후의 시점이므로 통신지연시간은 문제될 것이 없으며, $ARF_i \geq ARS_j$는 반드시 성립한다.

여러 단위노드와 통신하는 경우에도, 각 의사결정 프로세스가 자신이 보낸 메시지를 받는 단위노드에 대한 정보를 기억하고 있다가 수신확인 메시지를 모두 받은 이후에 작업종료 보고를 하게 되면, 통신에 참여하는 모든 단위노드에 대하여

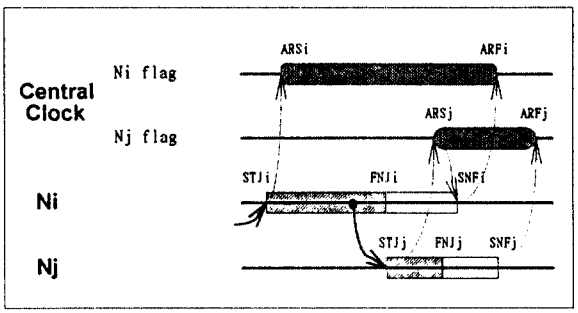
$ARF_i \geq ARS_n$, {n|P_n은 P_i로부터 메시지를 받는 단위노드의 프로세스}

를 만족하게 된다. 또한 이 방안은 자기 단위노드에서의 작업시작과 종료 보고를 자신만이 하기 때문에 통신지연시간등의 다른 조건에 상관없이 $ARS_i < ARF_i$를 만족한다.

이상의 두 조건-어떤 단위노드에서의 작업시작은 작업종료보다 먼저 인식($ARS_i < ARF_i$)되고, 이 단위노드에서 보낸 메시지로 인해 촉발된 다른 단위노드에서의 작업시작은 이 단위노드에서의 작업종료보다 늦지 않게 인식($ARF_i \geq ARS_j$)된다.-이 항상 만족되므로 이 방안은 다른 문제를 발생시키지 않으며, time-jump window를 원천적으로 봉쇄하게 된다.

여기에서 제기될 수 있는 의문은 N_i에서 사실보다 많은 시간 작업하고 있는 것으로 중앙시계에 인식됨으로서 즉, $ARF_i - ARS_j$의 기간이 사실보다 $SNF_i - FNJ_i$(통신지연시간의 편차를 고려하지 않았을 때)만큼 긴 시간으로 인식됨으로서 발생하는 문제는 없는가 하는 점이다.

먼저, $ARF_i < ARF_j$의 경우는 N_j에서 작업이 진행중인 것으로 인식되고 있는 상황에 덧붙여서 N_i에서도 작업중인 것으로 인식(실제로는 작업이 진행중이지 않지만)되는 경우이다. 〈그림 3〉의 동기화 방안의 흐름도에서 보듯이 중앙시계의 운영에서는 주어진 시점에 하나의 flag만 1인 경우와 여러개의 flag가 동시에 1인 경우는 전혀 다르게 취급되지 않는다. 따라서 작업이 진행되지 않음에도 불구하고 작업중인 것으로 인식되는 것은 다른 모든 flag가 0일때라면 문제가 있겠지만, 다른 단위노드의 flag가 1일때 같이 작업중으로 인식되는 것은 전혀 문제될 것이 없는 것이다.



〈그림 8〉 수신신호 확인후의 flag 처리

다음으로, $ARF_i > ARF_j$의 경우에도 중앙시계는 ARF_i 시점에서 모든 flag가 0임을 인식하고 사건리스트의 제일 빠른 사건의 시간으로 도약을 하면 된다. ARF_i- ARF_j사이에서는 실제나 시뮬레이션 작업이 아무것도 실행되지 않고 있기 때문에, 중앙시계의 사건리스트에 있는 다음사건이 처리되기 전까지는(즉, 이 사건의 처리를 통해 어떤 자극이 주어지기 전까지는) 모든 단위노드에서 어떤 작업도 발생하지 않는다. 따라서 이 경우에는 단지 ARF_i - ARF_j만큼 늦게 시간도약을 수행할 뿐 시뮬레이션의 진행에는 지장을 주지 않는다.

3.5 해결방안의 의사코드

각 단위노드의 의사결정 프로세스는 자신이 보낸 메시지의 수신측 프로세스들(rcvpids)과 그 갯수(rcvpidno)에 대한 정보를 가진다. 각 단위노드의 의사결정 프로세스가 중앙시계에게 보고하는 정보는 4-tuple(aflag, gflag, nevent, sendpid)로 구성된다. 여기서 aflag는 activity flag, gflag는 generation flag를 말하며, nevent는 시뮬레이션 사건중 가장 빠른 것의 시간이다. 그리고, sendpid는 activity flag를 1로 보고할때 함께 전달하는, 자신에게 메시지를 송신한

측의 의사결정 프로세스 id이다.

중앙시계는 메시지를 수신(receive)할때 새로 보고된 메시지가 없으면 기다리지 않고 바로 다음단계를 처리한다. 중앙시계는 시뮬레이션 프로세스가 사건처리를 시작하면 현재시간(tnow)을 보내준다. 또한, 수신측의 작업시작 보고메시지가 오면 송신측의 의사결정 프로세스에게 수신확인 메시지(ack)를 전달한다.

message(·)에서 괄호안의 변수는 메시지의 내용을 의미한다. m은 노드의 수이다. 그리고 [·]은 배열을 의미한다.

〈중앙 시계〉

```
VARIABLE
  tnow, aflag[m], gflag[m], nevent[m], sendpid

INITIALIZATION
  do : i = 1 to m
    aflag[i] = 1
    gflag[i] = 0
    nevent[i] = INFINITE
  enddo
  tnow = 0

STEP
  tnow = 1
  do tnow < simtime
    do : i = 1 to m
      receive message(aflag[i],gflag[i],nevent[i],sendpid) from
      i
      if aflag[i] = 1
        send message(i) to process sendpid
      endif
      if gflag[i] = 1
        send message(tnow) to process i
      endif
    enddo

    if flag[i] = 1, ∃i
      Real Stage:
        tnow = tnow + 1
        do (i,tnow = nevent[i])
          send message(tnow) to process i
          nevent[i] = INFINITE
        enddo
```

```
    else
      Simlulation Stage:
      if gflag[i] = 0, ∀i
        ntime := min(nevent[1], .. ,nevent[n])
        tnow = ntime
        do (i,tnow = nevent[i])
          send message(tnow) to process i
          nevent[i] = INFINITE
        enddo
      endif
    end if else
  enddo
```

〈의사결정 프로세스〉

```
VARIABLE
  tnow, aflag, gflag
  evntype[ ], evntime[ ], evntno,
  rcvpids[ ], rcvpidno

INITIALIZATION
  aflag = 0
  gflag = 0
  nevent = INFINITE
  evntno = 0
  (initialize status variables)
  send message(0,0,0,0) to CC(Central Clock)

STEP
  do true
    read message from pid
    if pid = CC and message = ack
      RECV_ACK
    else
      if pid = simulation process
        RECV_RESPONSE
    else
      RECV_MSG
    end if else
    if aflag = 1 and rcvpidno = 0 and working process don'
    t exist
      aflag = 0
      send message(0,-1,-1,-1) to CC
    endif
  enddo
```

```
RECV_MSG::
  receive message from pid
  aflag = 1
  if pid = CC
    send message(1,1,-1,-1) to CC
    event processing
  else
    if pid = decision making process in another node
      send message(1,-1,-1,pid) to CC
  end if else

  if need more processing
    send message to the process
  endif

  if need send message to j process
    send message to j process
    rcvpids[rcvpidno] = j
    rcvpidno = rcvpidno + 1
  endif

RECV_ACK::
  receive message(ack)
  if ack = rcvpids[k]
    delete rcvpids[k]
    rcvpidno = rcvpidno - 1
  endif

RECV_RESPONSE::
  receive message(type, time)
  evntype[evntno] = type
  evntime[evntno] = time
  nevent := min( {evntime[k], ∀k} )
  send message(-1,0,nevent,-1) to CC

  if need more processing
    send message to the process
  endif

  if need send message to j process
    send message to j process
    rcvpids[rcvpidno] = j
    rcvpidno = rcvpidno + 1
  endif
```

4. 결 론

본 논문에서는 혼합분산 시뮬레이션의 특성과 중앙시계를 이용한 동기화 방안을 고찰하였다. 그리고 혼합분산 시뮬레이션의 특성으로 인하여 중앙시계의 운영과정에서 발생할 수 있는 인과관계 오류에 대한 원인을 밝히고 그 해결방안을 제시하였다.

본 논문에서 제시한 해결방안은 혼합분산 시뮬레이션의 안전한 운영을 보장하지만 각 단위노드의 의사결정 프로세스는 자신이 메시지를 보낸 모든 단위노드에 대한 정보를 가지고 있어야 한다. 이것은 단위노드간의 통신이 적은 경우에는 별로 문제가 없지만 단위노드간의 긴밀한 연관성으로 인해 통신횟수가 잦은 경우에는 부담이 될 수 있다. 따라서 time-jump window를 확실하게 방지하면서도 정보부담이 적은 간결한 방안이 나올 수 있다면 더욱 바람직할 것이다. 또한, 시뮬레이션 사건의 처리에 있어서 이제까지의 event-driven방식이외에 time-driven방식을 사용하는 방안을 검토함으로써 좀더 나은 혼합분산 시뮬레이션 운영방안이 비교,검토되어야 한다.

참고문헌

[1] 김 성식과 배 경한, "컴퓨터를 이용한 실제에 준하는 FMS구축", 「산업공학」, Vol.4, No.1, pp.83-91, 1991.

[2] 배 경한, "Shell을 이용한 FMS구축 방법에 관한 연구", 고려대학교 산업공학과 박사 논문, December 1992.

[3] Chandy,K.M. and Misra,J., "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", *Comm.of the ACM*, Vol.24, No.11, pp.198-206, April 1981.

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

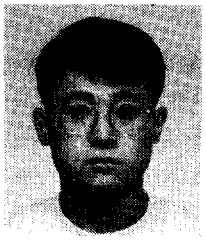
[5] Jefferson,D.R., "Virtual Time", *ACM Trans.on Prog. Lang.and Sys.*, Vol.7, No.3, pp.404-425, July 1985.

[6] Kim.S.S, "A Synchronization Scheme For Real And Simulated Events", *Computers & Industrial Engineering*, Vol.27, No.1-4, pp.181-184, 1994.

[7] Misra,J., "Distributed Discrete-Event Simulation", *Com-*

puting Survey, Vol.18, No.1, pp.39-65, March 1986.

● 저자소개 ●



서동욱
1995년 고려대학교 산업공학과 졸업
1995년~현재 고려대학교 산업공학과(석사과정 재학중)

전 진
1992년 고려대학교 산업공학과 졸업
1994년 고려대학교 산업공학과 석사
1994년~현재 고려대학교 산업공학과(박사과정 재학중)

김성식
1972년 고려대학교 기계공학과 졸업
1974년 고려대학교 산업공학과 석사
1976년 미국 S.M.U. 산업공학과 석사
1979년 미국 S.M.U. 산업공학과 박사
1979년~현재 고려대학교 산업공학과 교수