DEVS에 기반한 분산 시뮬레이션 환경 D-DEVSim++의 설계 및 구현

Design and Implementation of the DEVS-based Distributed Simulation Environment: D-DEVSim++

김 기 형, 김 탁 곤, 박 규 호
Ki Hyung Kim, Tag Gon Kim, Kyu Ho Park

Abstract

The Discrete Event Systems Specification (DEVS) formalism specifies a discrete event system in a hierarchical, modular form. This paper presents a distributed simulation environment D-DEVSim++ for models specified by the DEVS formalism. D-DEVSim++ employs a new simulation scheme which is a hybrid algorithm of the hierarchical simulation and Time Warp mechanisms. The scheme can utilize both the hierarchical scheduling parallelism and the inherent parallelism of DEVS models. This hierarchical scheduling parallelism is investigated through analysis. Performance of the proposed methodology is evaluated through benchmark simulation on a 5-dimensional hypercube parallel machine.

The performance results indicate that the methodology can achieve significant speedup. Also, it is shown that the analyzed speedup for the hierarchical scheduling time corresponds the experiment.

1. 소 개

분산 (병렬) 시뮬레이션은 시뮬레이션 프로그램을 병렬 컴퓨터에서 실행시키는 것을 의미한다. 최근에 와서 분산 (병렬) 이산 사건 시뮬레이션은 상당한 관심을 끌고 있는데, 공학, 전산, 경제, 군사 등 여러 응용 분야에서 시스템의 복잡도가 커짐에 따라 순차 컴퓨터에서 수행하기에는 너무 많은 시간이 걸리게 되고 분산 (병렬) 환경에서의 시뮬레이션이 그 한가지 해결 방법이 되고 있기 때문이다.

전통적으로 분산 시뮬레이션에서의 시스템은 통신을 통하여 서로 정보를 주고 받는 여러 프로세서들의 집합으로 간주하는 것으로서, 많은 분산 시뮬레이션 알고리즘들이 제안되었다. 분산 시뮬레이션에서 시스템간에 주고 받는 메시지에는 time-stamp가 붙어 있으며, 그 time-stamp에는 그 메시지가 처리되어야 하는 시뮬레이션 시간이 기록된다. 즉, 우리가 시간 t에 프로세스 p에서 메시지 m을 처리하게 하고 싶은 경우에는 메시지 m의 time-stamp에 t를 기록하여 그것을 프로세스 p에게로 전달하면 된다. 분산 시뮬레이션이 다른 분산처리 프로그램과 다른점은 각 프로세스가 자신에게 도착하는 메시지들을 그 메시지의 time-

stamp의 순서대로 배열하여서 시뮬레이션 해야만 한다는 것이다. 이러한 조건을 우리는 인과관계 조건(causality constraint)이라고 한다. 그러나 일반적인 경우에는 메시지들은 이 time-stamp 순서대로 도착하지 않고 위의 조건을 만족시키기란 매우 어려운 일이 된다. 지금까지 제안된 여러 분산 시뮬레이션 알고리즘을 분류해 보면 크게 보수적 방법(conservative methods)과 낙관적 방법(optimistic methods)으로 나뉜다[4]. 보수적 방법에서는 인과관계의 조건은 항상 만족된다. 즉 시뮬레이션을 진행하면서 인과관계가 만족되지 않으면 그 조건이 만족될 때까지 기다렸다가 시뮬레이션을 계속하는 것이다. 그러므로 교착상태(deadlock)에 빠질 가능성이 존재하게 된다. 대표적인 보수적 방법으로서는 Chandy와 Misra에 의해 제안된 알고리즘이 있다[9]. 반면 낙관적 알고리즘에서는 메시지가 도착하는 즉시 처리하고 나중에 인과관계 조건에 위배되는 것이 발견되면 그것을 수정(롤백이라고 부른다)하게 함으로써 시뮬레이션을 진행한다. 대표적인 알고리즘으로서는 Jefferson과 Sowizral에 의해 제안된 Time Warp 알고리즘이 있다[5].

분산 시뮬레이션은 크고 복잡한 시스템을 대상으로 하는 경우가 대부분이므로 모델의 검증(verification), 타당성(validation)등은 중요한 문제가 된다. Zeigler에 의해 제안된 DEVS 형식론(discrete event systems specification formalism)은 다양한 시뮬레이션 세계관(world view)과 모델링 방법들을 통합하고 수학적으로 잘 정의된 모델 구성을 위한 시스템 이론적 방법론이다[12, 13]. DEVS 형식론은 모델의 구조(structure)와 행동(behavior)을 시뮬레이션 수행으로 부터 추상화 시키기 위해 모델을 집합 이론적 방법을 이용하여 기술하며, 이산 사건 시스템을 계층적(hierarchical)이고 모듈화(modular)된 형식으로 기술할 수 있다. 이 계층화와 모듈화 특성은 위에서 언급한 모델의 검증과 타당성 등에 많은 도움을 줄 수 있다.

DEVS 모델의 분산 시뮬레이션에 대한 연구는 지금까지 많이 발표되었지만, 기존의 전통적인 분산 시뮬레이션과는 다른 방법들을 사용해 왔다. 그 이유는 DEVS 형식론이 모델의 외부 사건과 내부 사건을 분리하여 기술하고, 시뮬레이션 방법으로서 DEVS 추상 시뮬레이터(abstract simulator)를 사용해 왔기 때문이다[6]. 따라서 대부분의 병렬 DEVS 시뮬레이션 방법들은 DEVS 모델의 특정한 병렬성만을 이용해 왔다. Concepcion은 DEVS 추상 시뮬레이터에 적합한 병렬 하드웨어 구조를 연구하였다[3]. Wang은 Connection Machine-2에서 추상 시뮬레이터를 구현하였다[11]. Christensen은 Ada 언어 환경에서 DEVS 추상 시뮬레이터에 낙관적 방법을 제한적으로 적용하여 DEVS-Ada/TW이라는 시뮬레이션 환경을 구현하였다[2]. 보수적 방법과 낙관적 방법을 조합한 구조의 P-DEVSim++[10]도 개발되었다. 최근에는 Chow와 Zeigler가 모델의 병렬성을 자연스럽게 나타내기 위해 Parallel DEVS 형식론을 개발하였다[1]. 또한 Parallel DEVS에 기반을 두고 HCCL(Heterogeneous Container Class Library)라는 시뮬레이션 환경이 개발되었다[14]. 그러나 DEVS 형식론이 제공하는 병렬성은 제한적이기 때문에 대규모 병렬 처리 컴퓨터에는 적합하지 않다. 즉 이 병렬성은 동기적(synchronous) 병렬성(같은 time-stamp를 가지는 사건들만을 병렬화 시킴으로써 얻어지는 병렬성)이므로 비록 다른 time-stamp를 가지지만 서로 독립적인 사건들을 병렬화시킬 수는 없다.

본 논문에서는 DEVS 모델의 이러한 비동기적(asynchronous) 병렬성(서로 다른 시뮬레이션 시간을 갖는 사건들 사이의 병렬성)을 이용할 수 있는 분산 시뮬레이션 환경인 D-DEVSim++을 제안한다. D-DEVSim++은 이를 위해 가장 유명한 비동기적 동기 방법인 Time Warp을 기존의 계층적 시뮬레이션 기법에 적용하였다. DOHS (Distributed Optimistic Hierarchical Simulation)이라고 부르는 이 시뮬레이션 기법은 모델에 내재하는 병렬성 뿐만 아니라, 계층적 스케줄링에서 오는 병렬성도 이용할 수 있다. 본 논문에서는 D-DEVSim++의 각 부분에 대해 설명하고, 그 구현과 실험 결과들을 제시한다.

이 논문의 구성은 다음과 같다. 2 장은 DEVS 형식론과 계층적 시뮬레이션 기법, 그리고 Time Warp에 대해 간단히 설명한다. 3 장은 D-DEVSim++의 전체 구조와 각 부분의 역할 및 알고리즘을 설명한다. 4 장은 계층적 스케줄링을 병렬화 함으로써 얻을 수 있는 스케줄 시간에 대한 speedup을 분석한다. 5 장은 제안된 시뮬레이션 기법인 DOHS 방법의 성능을 벤치 마크 시뮬레이션 실험을 통해 제시한다. 6 장은 이 논문의 결론을 맺는다.

2. DEVS 형식론과 Time Warp

DEVS 형식론은 이산 사건 모델을 계층적이고 모듈화

된 형태로 표현할 수 있다[13]. 계층적인 모델 구성을 위해서, DEVS 형식론은 원소(atomic) 모델과 연결(coupled) 모델을 사용한다. 원소 모델은 시스템의 행동(behavior)을 나타내며, 연결 모델은 자신의 서브 모델들이 어떻게 연결되는가를 나타낸다. 연결 모델은 더 큰 연결 모델의 서브 모델이 될 수 있는데 이 특성을 이용하여 복잡한 모델이 계층적으로 구성될 수 있는 것이다. 원소 모델 AM은 다음과 같은 구조로 정의된다.

$AM=\langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$

where,

X : 외부 입력 사건 집합.

S : 순차 상태(sequential state) 집합.

Y : 외부 출력 사건 집합.

$\delta_{int} : S \rightarrow S$: 내부 상태 전이 함수.

$\delta_{ext} : Q \times X \rightarrow S$: 외부 상태 전이 함수.

$\lambda : S \rightarrow Y$: 출력 함수.

$ta : S \rightarrow R^{+}_{0,\infty}$: 시간 진행 함수.

, 여기서 Q는 M의 전체 상태들의 집합으로서 다음과 같이 나타낼 수 있다.

$Q=\{(s, e) \mid s \in S \text{ and } 0 \le e \le ta(s)\}$.

일반적인 모듈화된 명세(specification)와 마찬가지로 원소 DEVS 모델은 외부 세계와 입력 및 출력 포트를 통하여 통신한다. 좀더 상세히 말하면, 다른 모델로부터의 외부 입력 사건이 입력 포트로 들어오면, 모델은 외부 상태 전이 함수를 사용하여 어떻게 반응할 것인가를 결정한다. 만일 외부 사건이 다음 스케줄 시간(이 시간은 시간 진행 함수에 의해 결정된다) 까지 들어오지 않으면 모델은 내부 사건 전이 함수를 사용하여 상태를 전이한다. 이때 외부 출력 메시지도 출력 함수를 사용하여 동시에 출력한다. 이 출력 메시지는 다시 다른 모델에 외부 입력 사건으로서 역할을 하게 되고 이러한 전체 과정이 되풀이 되는 것이다.

여러 원소 모델들은 모여서 하나의 연결 모델이 될 수 있다. 이러한 연결 모델은 DEVS 형식론의 모듈화 특성에 따라 동등한 원소 모델로 표현될 수 있다. 그러므로 연결 모델은 그 자체가 여러 개 모여서 또 다른 연결 모델을 구성할 수 있는 것이다. 연결 모델 CM은 다음과 같이 정의된다.

$CM=\langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, SELECT \rangle$

where,

D : 구성 요소(component) 이름 집합.

For each i in D

M_i : 구성요소 i에 대한 DEVS.

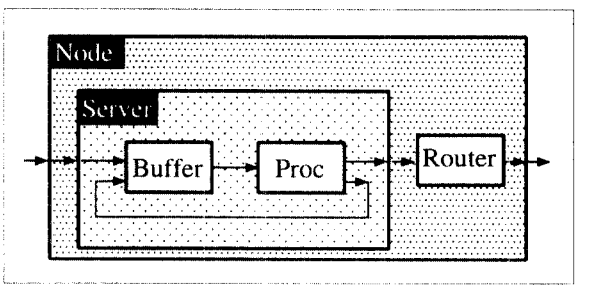
I_i : i의 influencee 모델들의 집합.

For each j in I_i

$Z_{i,j} : Y_i \rightarrow X_j$: i 모델의 출력을 j 모델의 입력으로 연결하는 함수.

SELECT : D의 부분 집합 $\rightarrow D$: 여러 구성요소 모델들이 같은 시간에 스케줄을 원할 때 그것들을 순서화 시키는 함수.

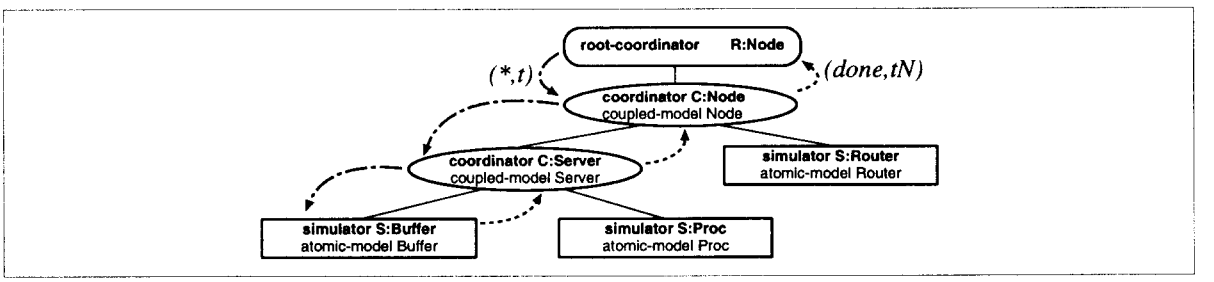
한 예로 〈그림 1〉은 간단한 큐잉 모델을 보여준다. 연결 모델 Node는 연결 모델 Server와 원소 모델 Router로 구성된다. 또한 연결 모델 Server는 다시 원소 모델 Buffer와 Proc으로 나누어 진다.



〈그림 1〉 큐잉 노드 모델 Node

2.1 계층적 시뮬레이션 기법

DEVS 모델을 시뮬레이션 하기 위해 추상 시뮬레이터(abstract simulator)가 개발되었다[13]. 추상 시뮬레이터는 DEVS 모델을 실행시켜주는 일종의 가상 프로세서로서 각 모델마다 하나씩 할당되고 연결 모델의 연결 정보에 따라 서로 연결된다. 두 가지 종류의 추상 시뮬레이터가 있는데, 원소 모델을 위해서 simulator, 연결 모델을 위해서 coordinator라고 부른다. 예를 들어서 〈그림 2〉는 Node 모델의 계층적 추상 시뮬레이터를 보여준다.



〈그림 2〉 Node 모델의 추상 시뮬레이터

Simulator와 coordinator는 (*,*t*), (*x*,*t*), (*y*,*t*), (*done*, t_N)의 네 가지 메시지 타입을 사용하여 보다 상위의 coordinator와 시뮬레이션에 필요한 정보를 주고 받는다. 위에서 *t*는 시뮬레이션 시간이고 t_N은 다음 스케줄 시간이다. 〈표 1〉은 simulator와 coordinator의 알고리즘을 보여준다.

이 스케줄링 과정은 다음과 같이 두 단계로 나뉘어진다. 첫번째 단계는 최소의 스케줄 시간(t_N)을 갖는 simulator를 찾아내는 단계이고, 두 번째 단계는 찾아낸 simulator를 실행시키는 단계이다. 이 두 단계의 스케줄링을 위해 (*,*t*)와 (*done*,,t_N) 메시지를 사용한다. 먼저 스케줄링의 두 번째 단계(스케줄 실행 단계)를 설명한다. (*,*t*) 메시지는 내부 사건(*)이 도착함을 나타낸다. Coordinator는 (*,*t*) 메시지를 받으면 가장 최소의 스케줄 시간을 가지는 구성요소에게 이 메시지를 전달한다. 만일 여러 개의 구성요소가 같은 최소 스케줄 시간을 가진다면, SELECT 함수를 시행하여 그 중 한 구성요소를 정한다. 이러한 과정을 통해 결국 최소의 스케줄 시간을 갖는 simulator는 (*,*t*) 메시지를 받게 되는데, 이때 simulator는 출력 함수를 실행하여 외부 출력 메시지((*y*,*t*))를 발생하고 내부 상태 전이 함수를 실행하여 상태 전이를 한다.

(*x*,*t*)와 (*y*,*t*) 메시지는 각각 외부 입력과 출력 사건 정보를 가지고 있다. Simulator가 출력 메시지 (*y*,*t*)를 출력하면, 그 simulator의 부모 coordinator는 이를 받아서 $Z_{i,j}$ 함수를 검사한다. 만일 그 메시지가 구성요소들에게 전달될 것이라면, (*x*,*t*) 메시지로 변환하여 그 구성요소에게 전달한다. 만일 그렇지 않다면, coordinator는 다시 자신의 부모 coordinator에게 그 출력 메시지를 전달함으로써 메시지에 대한 처리를 마친다. 또한 coordinator가 (*x*,*t*) 메시지를 부모 coordinator로부터 받으면 이 메시지를 구성요소들 중에서 적절한 목적지 구성요소에게 전달하면 된다. 이

〈표 1〉 추상 시뮬레이터 알고리즘

Message (Type)	Simulator	Coordinator
m(*)	−y← λ(s) −send y to the parent −$s \leftarrow \delta_{int}(s)$ −$t_L \leftarrow ts(m)$ −$t_N \leftarrow t_L + ta(s)$ −send a (done) message to the parent	−*select* one child from *star-children*, *i**, and send m to *i** −put *i** into *wait-list*
m(x)	−$e \leftarrow ts(m) - t_L$ −$s \leftarrow \delta_{ext}(s, e, m)$ −$t_L \leftarrow ts(m)$ −$t_N \leftarrow t_L + ta(s)$ −send a (done) message to the parent	−send *m* to the affected children −put the children into *wait-list*
m(y)		−if (m satisfy $Z_{i,j}$) { −translate *type*(*m*) into (*x*) −send *m* to influencess −put influencees into *wait-list* } else send *m* to the parent

m(done)		−delete *src*(*m*) form *wait-list*
		−if (*wait-list* is empty) {
		−$t_L \leftarrow ts(m)$
		−$t_N \leftarrow$ minimum t_N of children
		−put the children with minimum t_N *into star-children*
		−send a (done) message to the parent
		}

〈노트〉

a. 메시지 *m*에 대해 ts(*m*)은 m의 time-stamp, *src*(*m*)은 m의 소스 시뮬레이터, *dst*(*m*)은 *m*의 목적지 시뮬레이터이다.

b. 추상 시뮬레이터는 다음의 5개의 변수를 가진다: (1) *s*, 순차 상태 (sequential state); (2) t_L, 가장 최근에 처리했던 사건의 시뮬레이션 시간; (3) t_N, 다음 스케줄 시간; (4) *e*, 가장 최근에 처리했던 시간으로부터 현재까지의 경과 시간; (5) σ, 다음 스케줄 시간까지 남은 시간.

c. Coordinator는 다음의 두 집합을 관리한다: (1) *wait-list*, 현재 실행 중인 구성요소들의 집합; (2) *star-children*, 최소의 t_N을 가지는 구성요소들의 집합.

러한 전달 과정을 통해 소스 simulator에 의해 출력된 (y, t) 메시지는 목적지 simulator에게 전달되고, 목적지 simulator는 자신과 결합된 원소 모델의 외부 상태 전이 함수를 사용하여 상태를 전이한다.

Simulator가 위에서 설명한 바와 같이 (x,t) 또는 $(*,t)$ 메시지를 처리하고 나면 이제 스케줄링의 두 번째 단계(스케줄 실행 단계)는 끝나게 된다. 이제는 스케줄링의 첫번째 단계인 새로운 스케줄을 만드는 단계로 들어가게 된다. 새로운 스케줄을 만들기 위해 simulator는 부모 coordinator에게 $(done,t_N)$ 메시지를 보낸다. $(done,t_N)$ 메시지를 받은 coordinator는 자신의 다음 스케줄 시간(t_N)을 구성요소들의 t_N들 중에서 최소값으로 정하고 다시 부모 coordinator에게 $(done,t_N)$을 보낸다. 이 과정은 최상위 coordinator (root-coordinator라고 불린다)에게 $(done,t_N)$이 전달될 때까지 계속된다. 〈그림 2〉는 $(*,t)$와 $(done, t_N)$ 메시지의 전달 경로를 보여준다. Root-coordinator는 실제로 전체 시뮬레이션 진행을 맡고 있는 coordinator이다. $(done, t_N)$를 받으면 root-coordinator는 시뮬레이션 시간을 t_N으로 맞추고 새로운 $(*, t_N)$을 발생시킴으로써 새로운 시뮬레이션 사이클을 진행한다. 이렇게 두 단계의 계층적 스케줄 과정을 반복함으로써 시뮬레이션이 진행되는 것이다.

2.2 Time Warp 기법

Time Warp은 가장 유명한 동기화 프로토콜 중의 하나이다[4]. 다른 동기화 프로토콜들에 대한 자세한 설명은 [9]를 참조하라. Time Warp 프로토콜은 다음과 같이 두 가지 기법으로 구성된다: 지역 제어 기법과 전역 제어 기법. 지역 제어 기법은 메시지들을 올바른 순서로 실행시키는 역할을 한다. 전역 제어 기법은 시뮬레이션 전반에 걸친 이슈들 (입출력 핸들링, 종료 탐지, 메모리 관리, 흐름 제어 등)을 다룬다. Time Warp의 지역 제어 기법은 메시지의 처리에 있어서 낙관적인 입장을 취하는데, 일단 메시지가 도착하면 처리하고 본다. 나중에 더 작은 time-stamp를 가지는 메시지(straggler라고 부른다)가 도착하면, 프로세스는 시뮬레이션 시간을 straggler의 time-stamp로 롤백하고 그 시점부터 시뮬레이션을 다시 시작한다. 이러한 롤백을 지원하기 위해서 각 프로세스는 입력 메시지 큐, 출력 메시지 큐, 그리고 상태 큐의 세가지 큐를 관리해야 한다. 입력 메시지는 입력 메시지 큐에서 time-stamp 순서대로 정렬된다. 상태 큐는 각 입력 메시지를 처리하고 난 후의 상태의 복사본을 저장한다. 출력 메시지 큐는 이미 출력된 메시지들의 반대 이미지(anti-메시지라고 부른다)를 저장한다. 즉 정상적인 메시지는 '+'사인을 가지고있고, anti-메시지는 '−' 사인을 가지고 있는 것이다. 롤백이 일어날 때 anti-메시지들은 이미 출력된 출력 메시지들의 영향을 제거하기 위해 사용된다.

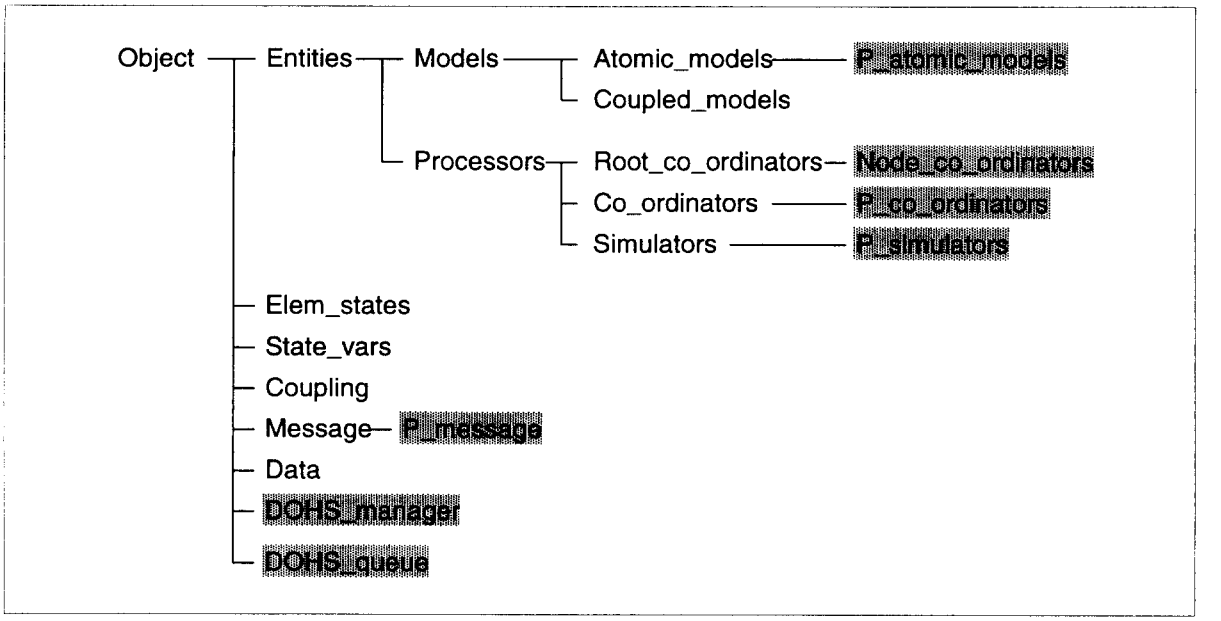
3. 병렬 DEVS 시뮬레이션 환경 D-DEVSim++

본 장에서는 DEVS 모델의 분산 시뮬레이션 환경인 D-DEVSim++을 제안한다. D-DEVSim++은 C++을 이용한 DEVS 모델의 추상 시뮬레이터 환경인 DEVSim++을 기반으로 해서 계층적 시뮬레이션을 병렬화 하였다[8]. 〈그림 3〉은 D-DEVSim++의 클래스 계층구조이다. 빗금 쳐지지 않은 클래스들은 DEVSim++의 클래스 구조를 나타낸다. 빗금 친 클래스들이 병렬화를 구현하기 위해 새롭게 DEVSim++에 더해진 클래스들이다. 본 장에서는 D-DEVSim++의 전체적인 병렬화 기법과 각 빗금 친 클래스의 역할을 설명한다.

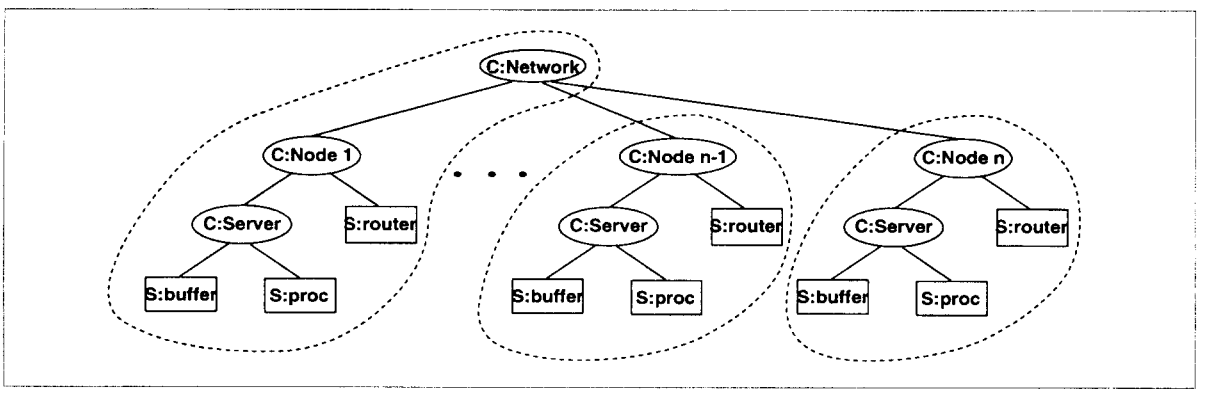
3.1 D-DEVSim++의 분산 시뮬레이션 기법

DEVS 모델의 시뮬레이션을 병렬화 시키기 위하여 우리는 DOHS(Distributed Optimistic Hierarchical Simulation) 방법을 제안한다. 이 방법은 기본적으로 추상 시뮬레이터의 계층적 시뮬레이션 기법을 바탕으로 하고 있는데 먼저 추상 시뮬레이터들을 파티션하고 분산 컴퓨터에 맵핑시킨다. 〈그림 4〉는 한 예로서 파티션된 Closed Queuing Network(CQN) 모델의 추상 시뮬레이터를 보여준다.

각 노드 컴퓨터는 맵핑된 추상 시뮬레이터들을 스케줄링하기 위해 자신만의 스케줄러를 갖게 된다. 각 노드 컴퓨터에서의 사건 처리에 있어서 인과관계의 제약을 만족시키기 위해서는 동기화 프로토콜이 필요하게 되는데, DOHS 방법은 Time Warp 프로토콜을 도입하였다. Time Warp 프로토콜을 계층적 시뮬레이션 기법에 도입하기 위해서 DOHS 방법은 새로운 시뮬레이션 구조를 갖는다. 〈그림 5〉는 DOHS 방법의 전체 구조를 보여준다. DOHS 방법은 각 노드 컴퓨터에서 *node-coordinator*, 분산된 추상



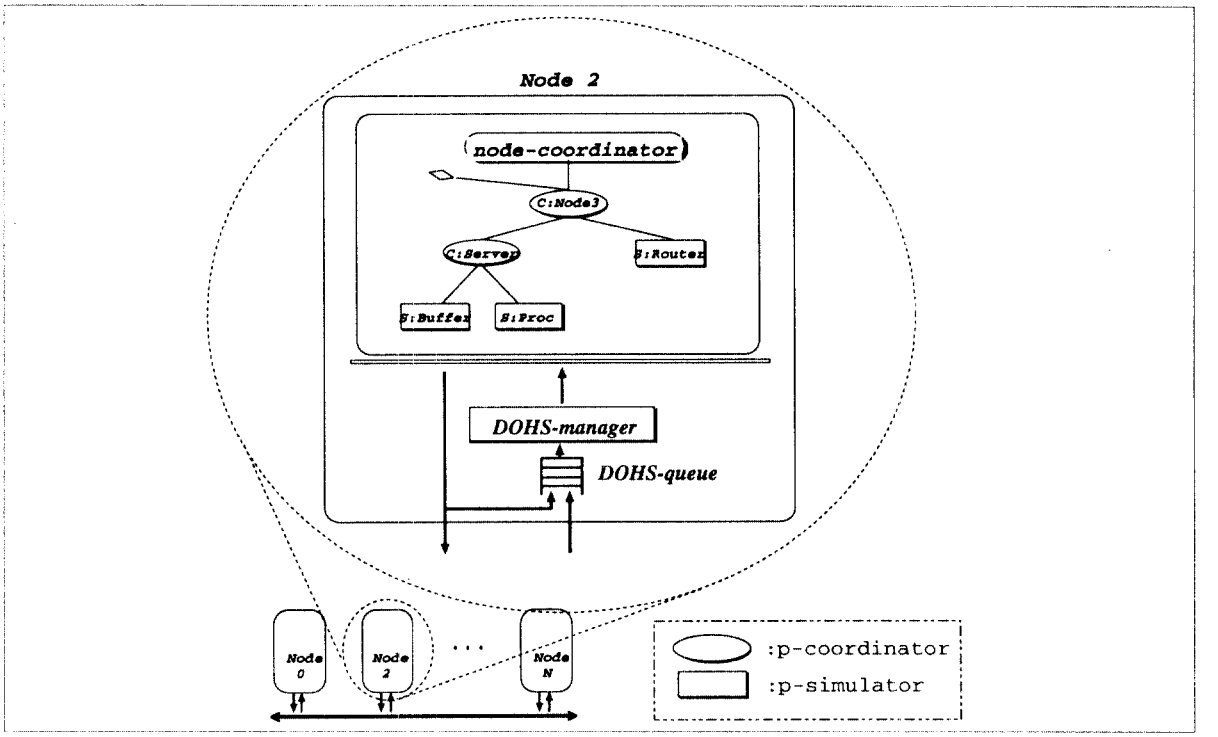
〈그림 3〉 D-DEVSim++의 클래스 계층구조



〈그림 4〉 CQN 모델의 추상 시뮬레이터를 분산 시뮬레이션하기 위한 파티션

시뮬레이터들, *DOHS-manager*, 그리고 *DOHS-queue*의 네 가지 부분으로 구성된다. Node-coordinator는 앞장에서 설명한 root-coordinator의 병렬 버전으로서, 각 노드 컴퓨터에서의 시뮬레이션을 독자적으로 스케줄링하기 위해 고안되었다. DOHS 방법은 비동기식(asynchronous) 분산 시뮬레이션 방법이므로, 각 노드 컴퓨터는 독자적으로 시뮬레이션 시간을 관리할 수 있어야 한다. 이를 위해 계층적 시뮬레이션의 전체 시뮬레이션 관리자인 root-coordinator를 각 노드에 분산시킨 것이다. Node-coordinator는 각 노드 컴퓨터의 지역 스케줄러로서 할당된 추상 시뮬레이터들의 시뮬레이션 시간을 관리하고 시뮬레이션을 스케줄링 한다.

각 노드 컴퓨터는 서로 다른 시뮬레이션 시간을 가지고 있고 이들은 서로 메시지를 주고 받으므로, 서로 다른 time-stamp를 갖는 메시지들이 섞여서 처리되어야 한다. 이들을 순서대로 처리하기 위해서 *DOHS-manager* 와 *DOHS-*



〈그림 5〉 DOHS 방법의 전체 구조

*queue*가 개발되었다. 즉 *DOHS-manager* 와 *DOHS-queue*는 각 노드 컴퓨터들 간의 동기(synchronization)를 관리한다. 각 노드에서 DOHS-manager와 DOHS-queue는 메시지들을 순서대로 처리한다. DOHS-queue는 〈표 2〉과 같은 우선순위로 처리할 메시지들을 정렬하는 우선순위 큐이다. 이 우선순위는 기본적으로는 메시지의 time-stamp 순서대로인데 시뮬레이션의 성능에 직접 영향을 미치는 특별한 메시지들에게는 우선순위를 준다. 또한 같은 time-stamp를 갖는 메시지들에게는 메시지의 타입에 의해 우선순위를 주는데 이는 올바른 시뮬레이션의 진행을 위해서 필요하다.

DOHS-queue에 들어 있는 정렬된 메시지들을 실행시키는 제어기는 DOHS-manager이다. DOHS-manager는 〈그림 6〉과 같이 DOHS-queue의 제일 앞에 있는 메시지를 꺼내서 그 메시지의 목적지 추상 시뮬레이터에게 전달함으로써 메시지의 처리를 한다. 또한 시뮬레이션 진행을 위한 특별한 메시지들에 대해서는 직접 처리를 한다. 또한 만일 다른 컴퓨터에서 온 메시지가 현재 실행되고 있는 메

〈표 2〉 DOHS-queue에서 메시지의 우선순위

	Precedence rule of messages	
1	특별한 타입의 메세지들((flush), (collect), (cgst), (lst), (gst))	
2	Increasing time-stamp 순	
3	같은 time-stamp의 메세지들에 대해	(*)
4		(x) and (y)
5		(done)

시지의 time-stamp보다 더 작은 time-stamp를 가지고 있다면 DOHS-queue는 다음에 설명할 계층적 롤백을 시작한다

추상 시뮬레이터는 DEVS 모델을 실제로 시뮬레이션 한다. DOHS 방법에서는 Time Warp 기법을 구현하기 위해 병렬 추상 시뮬레이터 알고리즘을 개발하였다. 병렬 추상 시뮬레이터에는 두 가지가 있는데, 순차 추상 시뮬레이터인 simulator와 coordinator 각각에 대해서 *p-simulator*와 *p-coordinator*가 개발 되었다[6].

Algorithm 1 DOHS-manager()

```
    ▷ initialization phase
1   노드 컴퓨터에 맵핑된 추상 시뮬레이터들을 초기화시킨다.
    ▷ 메인 시뮬레이션 루틴
2   while (true)
3       if (there is a message v in communication buffer) then do
4           DOHS-queue:Insert(v);
5       end if
6       fetch a message u from the head of the DOHS-queue;
7       if (u.type ∈ {(gst), (lst), (cgst)}) then do
8           calculate GST;
9           if (GST = infinity) then do
10              terminate simulation;
11          else
12              send (collect) message to the node-coordinator;
13          end if
14      else
15          M ← the destination simulator of u.
            ▷ u의 타입을 해석해서 u의 목적지 시뮬레이터에게 전해준다.
16          case u.type
17              (x):M.when_rcv_x();
18              (y):M.when_rcv_y();
19              (*):M.when_rcv_star();
20              (done):M.when_rcv_done();
21          end case
22      end if
23  end while
```

〈그림 6〉 DOHS-manager 알고리즘

DOHS 방법은 Time Warp 프로토콜을 사용하므로 다음과 같이 크게 지역 제어 기법(local control mechanism)과 전역 제어 기법(global control mechanism)의 두 가지 파트로 나눌 수 있다. 지역 제어 기법은 메시지들이 시뮬레이션 시간 순서대로 실행될 수 있도록 해주는 기법이다. 즉 롤백과 재계산 기법을 의미한다. 전역 제어 기법은 I/O 문제나 종료 탐지, 메모리 관리 등과 같은 시뮬레이션 전반에 걸친 문제들을 다룬다.

롤백을 위해 각각의 p-simulator는 입력 메시지 큐, 상태 큐, 출력 메시지 큐의 세가지 자료 구조를 가진다. 입력 메시지 큐는 들어오는 (x) 나 (*) 메시지를 time-stamp 순서대로 저장한다. 상태 큐는 롤백을 위해 모델의 상태를 주기적으로 저장한다. 출력 메시지 큐는 이미 보낸 출력 메시지의 복사본(anti-메시지)을 저장한다.

DOHS 방법의 롤백은 기존의 롤백보다 더 복잡하다. 그 이유는 DOHS 방법이 기본적으로 계층적 시뮬레이션을 통해 p-simulator들을 스케줄링하기 때문이다. 롤백은 다음과 같이 세가지 스텝으로 구성된다. 첫째는 straggler 메세지가 다른 노드 컴퓨터로부터 도착했을 때, 현재 진행 중인 계층적 스케줄을 취소해야 한다. 물론 이 방법만 있는 것은 아니다. 이러한 straggler에 대한 쉬운 대처 방법은 현재의 스케줄이 끝날 때까지 기다렸다가 그 후에 straggler에 대한 처리를 하는 것이다. 그러나 이러한 방법은 현재의 스케줄 실행이 언제 끝날지 예측을 할 수 없고 또한 현재의 스케줄이 롤백의 대상이 될 수도 있으므로 바람직하지 않다. 그러므로 효율적인 롤백을 위하여 DOHS 방법

은 straggler가 도착하자 마자 현재 진행 중이던 스케줄의 실행을 중지시키는 것이다. 스케줄 취소가 끝난 후 straggler는 DOHS-manager에 의해 목적지 simulator에게 전달된다. 그 다음은 두 번째 롤백 단계로서 그 목적지 simulator가 상태와 이미 내보낸 출력 메시지를 롤백한 다음 straggler부터 재 처리한다. 마지막 롤백 단계는 이러한 재계산이 끝난 후 이 simulator들을 coordinator들이 다시 계층적으로 재 스케줄링을 하는 것이다. 그러므로 이러한 세가지 단계의 롤백을 거치면 시뮬레이션은 straggler로부터 새로운 시뮬레이션이 진행될 수 있는 것이다. 다음은 이 계층적 롤백을 각 단계별로 상세히 설명한다.

3.1.1 계층적 스케줄 취소 알고리즘

DOHS 방법의 스케줄링은 기본적으로 계층적으로 이루어지기 때문에 스케줄 정보가 여러 coordinator들에게 분산되어있다. 따라서 이렇게 분산된 스케줄 정보를 제거하기 위해서 계층적인 스케줄 취소 알고리즘이 개발되었다. 먼저 coordinator의 상태를 두 가지로 나누어 볼 수 있는데 coordinator가 현재의 스케줄 과정에 관여하고 있다면 스케줄 상태, 그렇지 않다면 휴지(idle) 상태이다. 스케줄 취소 알고리즘은 스케줄 상태에 있는 모든 coordinator들에게 현재의 스케줄이 취소되었음을 알려야 한다. 스케줄 상태에 있는 coordinator는 (*) 메시지를 자신의 구성요소에게 이미 주었고 그 구성요소로부터 (*done*) 메시지를 기다리고 있는 coordinator이다. 그러므로 스케줄 상태에 있는 coordinator는 다음의 세가지 중에 한가지 경우이다. 즉, DOHS-queue에 있는 (*) 메시지들의 소스이거나, DOHS-queue안에 있는 (*done*) 메시지들의 목적지이거나, 다른 스케줄 상태에 있는 coordinator의 부모인 경우이다. 이러한 정보를 이용해서 스케줄 상태의 coordinator를 발견하는 알고리즘이 〈그림 7〉이다.

일단 그러한 coordinator가 발견되고 난후 스케줄이 취소되었음을 그 coordinator에게 알려주어야 하는데, 이를 위해 새로운 타입의 메시지 (*flush*)를 정의한다. Coordinator가 (*flush*) 메시지를 받으면 자신의 상태를 휴지 상태로 바꾸고 그 부모 coordinator에게 그 메시지를 보낸다.(부모

Algorithm 2 DOHS-queue: Hierarchical Schedule Cancelation

▷ 메시지 w에 대해, *src(w)*와 *dst(w)*는 각각 소스와 목적지 추상 시뮬레이터를 의미한다.

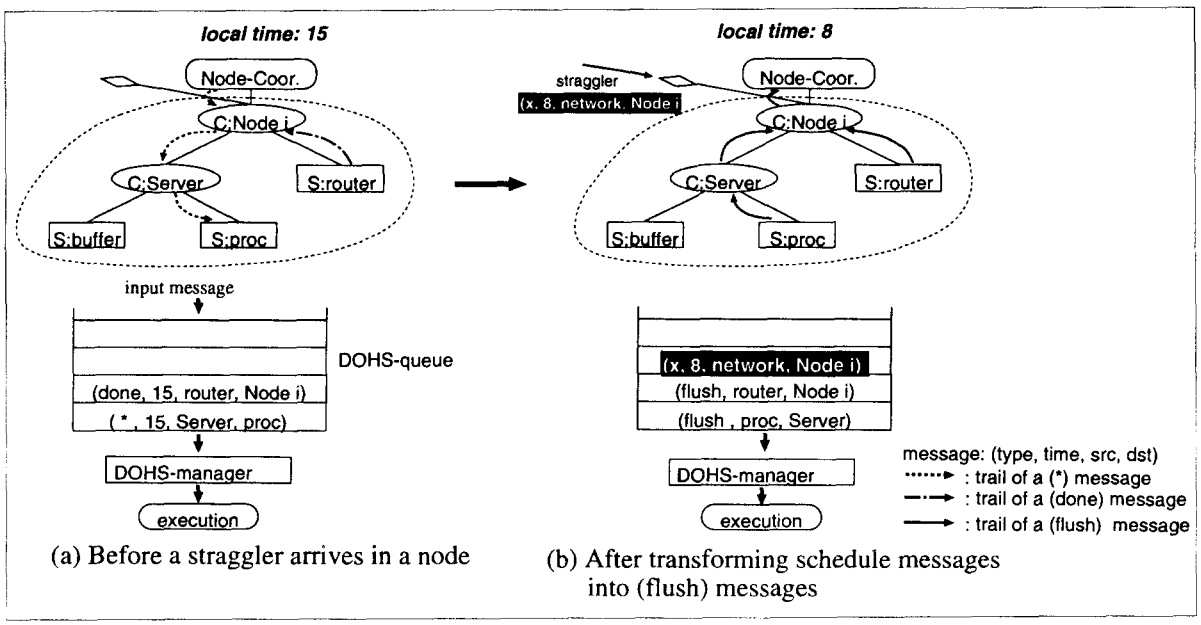
```
1  for each message w with a larger time-stamp than the straggler in the DOHS-queue do
2      if ( type(w) == (*) ) then do
3          delete w;
4          create a (flush) message f;
5          dst(f) ← src(w);
6          insert f before the straggler;
7      else if (type(w) == (done)) then do
8          delete w;
9          create a (flush) message f;
10         dst(f) ← dst(w);
11         insert f before the straggler;
12     else if (type(w) == (x) && src(w) is mapped in the same node ) then do
13         create a (flush) message f;
14         dst(f) ← src(w);
15         insert f before the straggler;
16     end if
17 end for
```

〈그림 7〉 계층적 스케줄 취소를 시작하는 알고리즘

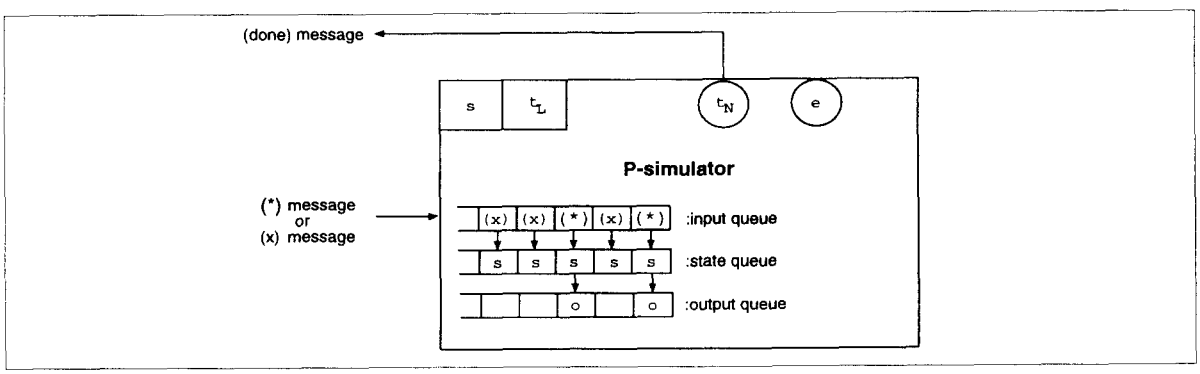
coordinator도 역시 스케줄 상태에 있음을 주의하자.) 〈표 2〉에서 보인 바와 같이, (*flush*) 메시지는 DOHS-queue에서 가장 높은 우선순위를 가지고 있음을 알 수 있다. 이것은 스케줄 제거 알고리즘이 straggler 메시지의 처리보다 먼저 시급히 처리되어야 함을 의미한다. 예를 들어서 〈그림 8〉은 스케줄 취소의 스냅샷을 보여준다. Straggler 메세지가 도착했을 때 DOHS-queue에는 두개의 스케줄 메세지가 있었다〈그림 8(a)〉. 스케줄 취소 알고리즘은 이 두 메시지들을 (*flush*) 메시지로 바꾸고 straggler 메시지 앞에 삽입한다. 이들 (*flush*) 메시지를 다 처리하고 나면 비로서 현재의 스케줄이 취소되는 것이다. 〈그림 8(b)〉에서 실선의 화살표는 이들 (*flush*) 메시지의 전달 경로를 보여준다.

3.1.2. P-simulator의 롤백 알고리즘

계층적 스케줄 취소 과정이 끝나면 straggler 메시지는 DOHS-manager에 의해 목적 p-simulator에게 전달된다. 이



〈그림 8〉 계층적 스케줄 취소 과정



〈그림 9〉 롤백을 위한 P-simulator의 구조

Algorithm 3 P-Simulator:Input-Synchronization()

```
    ▷ m+ 는 + 사인을 가지는 메시지 m을 의미한다.
    ▷ taps order는 time-and-priority-stamp 순서를 의미한다.
1   if (timestamp(m) >= T_L ) then do
        ▷ m은 미래의 time-stamp를 갖는 메시지이다.
2       if (sign(m) == '+') then do
3           if (m- ∈ IQ) then do
4               annihilate m-;
5           else
6               insert m+ into IQ in the chronological order (taps order);
7           end if
8       else if (sign(m) == '-') then do
9           if (m+ ∈ IQ) then do
                ▷ m+은 아직 처리되지 않았다.
10              annihilate m+;
11          else
12              insert m- into IQ in the chronological order;
13          end if
14      end if
15  else if (timestamp(m) < T_L ) then do
        ▷ m은 과거의 시간을 갖는 메시지이다.
16      if (sign(m) == '+') then do
17          if (m- ∈ IQ) then do
18              annihilate m-;
19          else
20              rollback;
21              insert m+ into IQ in the chronological order;
22          end if
23      else if (sign(m) == '-') then do
24          if (m+ ∈ IQ) then do
                ▷ m+은 이미 처리된 메시지이다.
25              rollback;
26              annihilate m+;
27          else
28              insert m- into IQ in the chronological order;
29          end if
30      end if
31  end if
```

〈그림 10〉 P-simulator의 입력 메시지 동기화 알고리즘

때 p-simulator는 상태와 출력 사건들을 롤백하게 된다. 이 두 번째 단계의 롤백을 위한 p-simulator의 구조는 〈그림 9〉와 같다.

P-simulator는 전통적인 Time Warp에서와 같이 세 개의

큐를 가지고 있다. 외부 사건 메세지가 도착하면 p-simulator는 이 메시지를 입력 메시지 큐에 증가하는 time-and-priority-stamp 순서로 삽입한다. 여기서 time-and-priority-stamp는 DEVS의 사건들을 분산 시뮬레이션 환경에서 정렬할 수 있도록 제안된 것이다[7]. 이 삽입 과정에서 롤백 여부를 결정하는 알고리즘은 〈그림 10〉과 같다.

3.1.3 계층적 재 스케줄링

두 번째 롤백 단계를 마치면 여러 개의 p-simulator들이 인과관계에 의해 롤백 되게 된다. 세 번째 롤백 단계는 이들 p-simulator들을 계층적으로 재 스케줄링하는 것이다. 이를 위한 p-coordinator의 알고리즘을 개발하였다[6].

4. 스케줄링 시간의 분석

계층적 시뮬레이션에서 스케줄 메시지들((*) 와 (*done*))은 각 시뮬레이션 스텝마다 추상 시뮬레이터의 계층을 순회해야 한다. 계층적 스케줄링은 여러 가지 장점을 가지고 있으나, 한 simulator를 스케줄링하기 위해서는 계층(level) 수 만큼의 부모 coordinator를 거쳐야 한다. 그러므로 다른 조건이 모두 같을 때, 계층의 수가 많을수록 스케줄링에 걸리는 시간이 많아진다. DOHS 방법은 스케줄링 자체를 병렬화 하였으므로 각 노드 컴퓨터에 할당된 추상 시뮬레이터의 계층수는 전체 계층수 보다 줄어들고 이에 의한 성능향상을 얻을 수 있다. 다음의 정리 1은 계층이 줄어듬으로써 생기는 스케줄 시간 speedup을 보여준다.

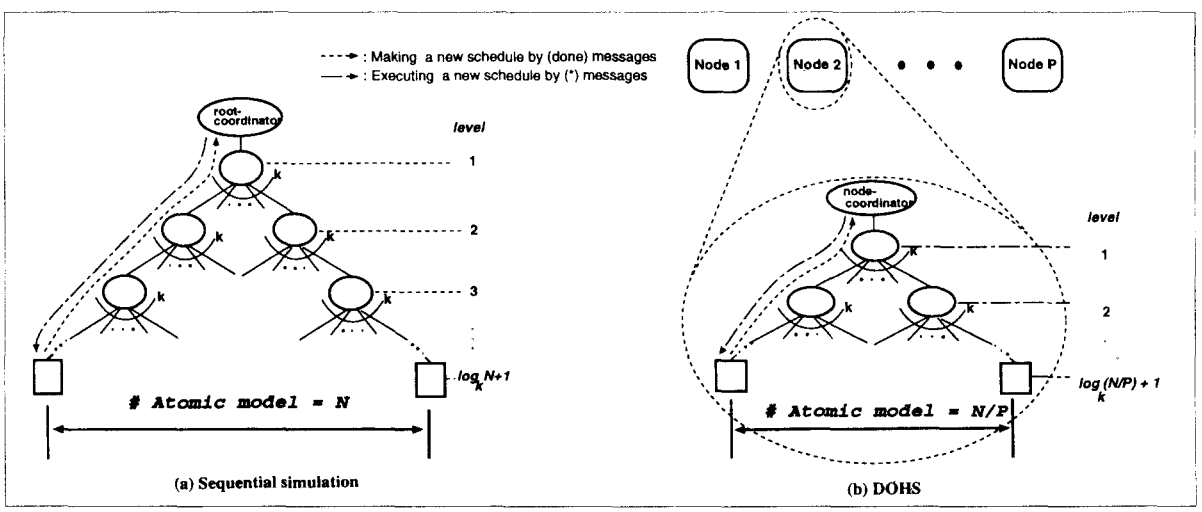
정리 1 〈그림 11〉에서 보는 바와 같이 N 개의 원소 모델을 위한 높이 l, 차원 k의 균형 잡힌 추상 시뮬레이터 구성 트리를 가정할 때, 내부 사건의 병렬화를 통해 얻을 수 있는 스케줄링 시간의 speedup은 다음과 같다.

$$S_{sync} = \rho \cdot P \cdot \frac{\log_k N + 1}{\frac{N}{P} + 1}, \qquad (1)$$

위의 식에서, $\rho = \frac{number\ of\ committed\ (*)\ messages}{number\ of\ (*)\ messages}$, N = 원소 모델의 수, P = 노드 컴퓨터의 수, k = 각 연결 모델에서 서브 모델의 수

증명 〈그림 11〉에서 트리의 높이(l)는 $l = \log_k(N) + 1$이 된다.

앞 장에서 추상 시뮬레이터의 스케줄링 과정은 두 단계, 즉 (1) 최소의 스케줄 시간(t_N) 결정 단계와 (2) 찾아낸 simulator를 실행시키는 단계로 나뉘어진다고 하였다. 우선 최소 스케줄 시간 결정 단계에서 각 레벨의 coordinator는 자신의 모든 구성요소들의 t_N들을 비교하고 최소의 t_N($min\ t_N$)을 찾아서 부모 coordinator에게 (*done*, min t_N) 메시지를 보내게 된다. (*done*, min t_N) 메시지를 받으면 root-coordinator는 이 스케줄을 실행시키기 위하여 (*, min t_N)



〈그림 11〉 계층적 스케줄 시간에 대한 Speedup

메시지를 발생시키고 이제부터 스케줄 실행 단계가 시작된다. 이 메시지를 받은 각 레벨의 coordinator는 최소의 t_N을 가지는 구성요소를 발견하고 그 구성요소에게 이 메시지를 전달하게 된다. 이와 같이 두 단계의 스케줄 프로세스에서 걸리는 시간을 모두 종합해보면, 한 시뮬레이션 스텝에서 스케줄 프로세스에 걸리는 전체 시간은 다음과 같다.

$$t_{seq}=(ck+a)\cdot(\log_k N+1)$$
$$t_{DOHS}=(ck+a)\cdot(\log_k \frac{N}{P}+1).$$

여기서, c 는 k 개의 구성요소의 t_N을 비교하기 위한 상수이고, a는 스케줄 실행 단계에서 최소의 t_N을 발견하는 데 걸리는 시간이다.

시뮬레이션 전체 스텝에서 걸리는 스케줄 시간은 발생된 모든 (*) 메시지의 수만큼 위의 스케줄 시간을 더하면 된다.

$$T_{seq}=m\cdot t_{seq}=m\cdot(ck+a)\cdot(\log_k N+1)$$

$$T_{DOHS}=\left\lceil\frac{m+u}{p}\right\rceil\cdot t_{DOHS}=\frac{1}{\rho}\cdot\frac{m}{p}\cdot t_{DOHS}=\frac{1}{\rho}\cdot\frac{m}{p}\cdot(ck+a)\cdot\left\lfloor\log_k\frac{N}{P}+1\right\rfloor.$$

여기서, m은 전체 노드 컴퓨터에서 올바르게 실행된 (*) 메시지의 전체 갯수이고, u는 전체 노드 컴퓨터에서 올바르게 실행되지 않고 롤백된 (*) 메시지의 전체 갯수이다.

그러므로 T_{seq}를 T_{DOHS}로 나누면 (*) 메시지의 발생을 병렬화 시킴으로 해서 얻어지는 스케줄링 시간의 speedup을 계산할 수 있다.

$$S_{sch}=\frac{T_{seq}}{T_{DOHS}}=\rho\cdot P\cdot\frac{\log_k N+1}{\log_k\frac{N}{P}+1}.\qquad\square$$

만일 위의 정리를 비대칭의 구성 트리로 확대한다면, speedup은 다음과 같이 예측할 수 있다.

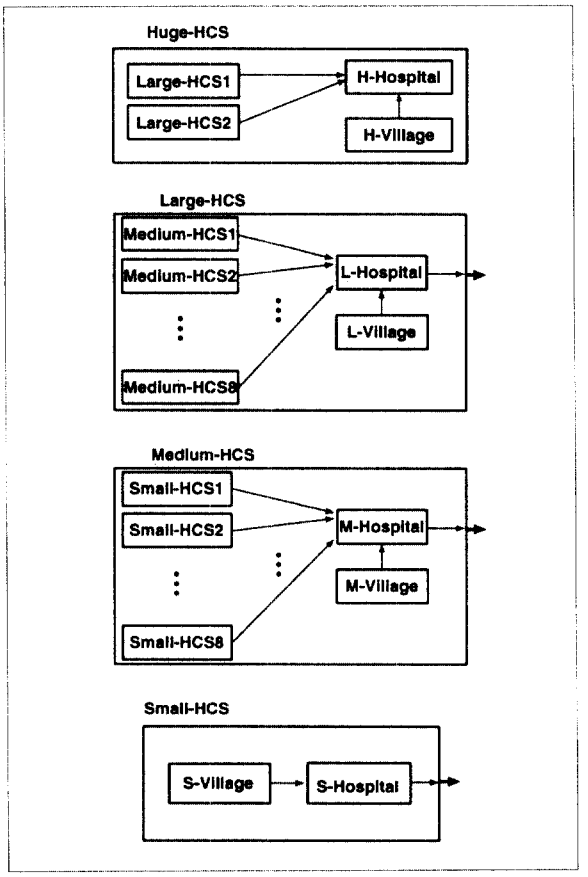
$$S_{sch}=\frac{T_{seq}}{T_{DOHS}}\approx\rho\cdot P\cdot\frac{l}{l'}\qquad(2)$$

, 여기서 ρ는 올바르게 처리된 (*) 메시지의 수와 전체 (*) 메시지의 수의 비율이고, P는 노드 컴퓨터의 수, l는 원래의 구성트리의 레벨의 수, 그리고 마지막으로 l'은 파티션된 서브트리의 레벨의 수이다.

l'은 보통 l보다 아주 작기 때문에 만일 $\rho > l'/l$ 이라면 스케줄 시간에서 만큼은 초 병렬 speedup이 얻어질 수 있다. 사실 ρ는 시뮬레이션하고자 하는 시스템에 내재하는 병렬성을 나타내므로 ρ가 충분히 크지 않다면 이 초 병렬 speedup이 실제의 speedup으로 나타나기는 어렵다.

5. 실험 결과

이장에서는 제안된 DOHS 방법을 실험을 통해서 성능을 평가해 본다. 성능 평가에 사용된 모델은 콜롬비아 보건 관리 시스템(CHCS)이라고 부르는 쿠잉 모델이다. 이 모델은 여러 분산 시뮬레이션 연구들에서 사용되어 왔다. CHCS 모델은 마을과 보건소를 기본단위로 하는 여러 계층의 보건 관리 시스템이라고 할 수 있다. 각 마을에는 한



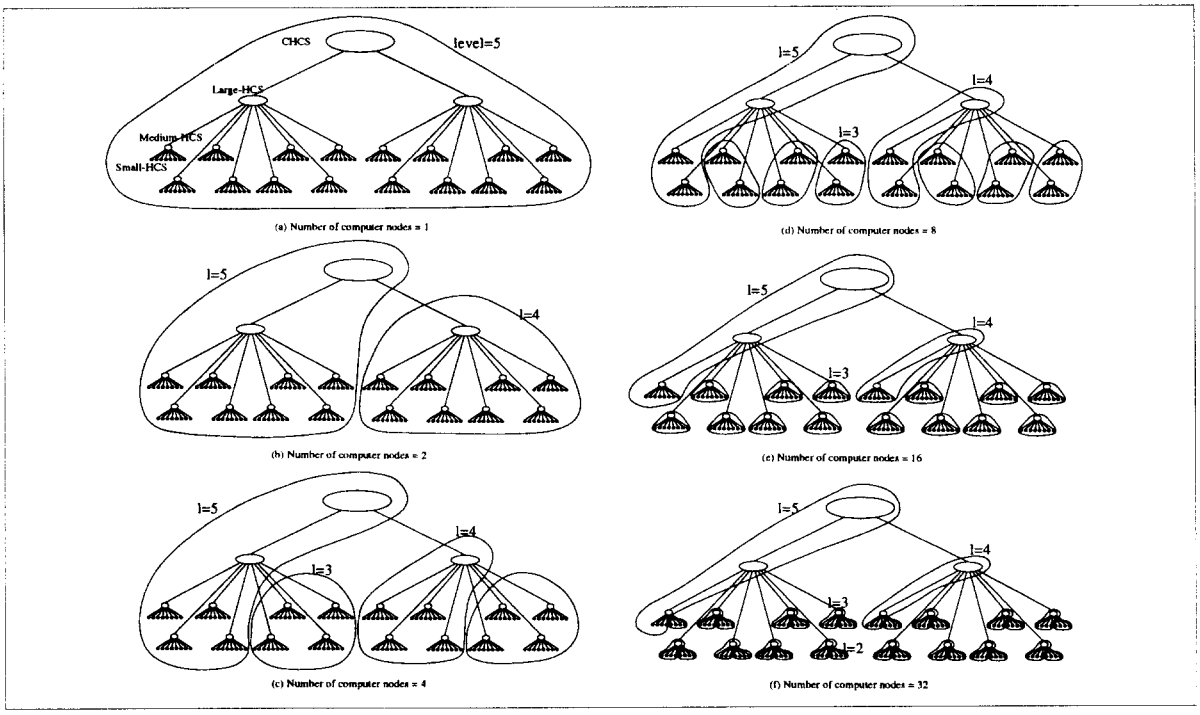
〈그림 12〉 콜롬비아 보건 관리 시스템의 계층적 모델 구성

개의 보건소가 있다. 마을 사람들이 병에 걸리면 그 지역의 보건소로 가서 진찰을 받고 가능하면 치료도 받게 된다. 만일 그 보건소에서 치료할 수 없는 병이라면 좀더 큰 병원으로 가게 되고 다시 진찰과 치료를 받게 된다. 병원에 환자가 도착하면, 환자는 큐에 넣어지고, 순서대로 진찰을 받는다. 마지막으로 가장 높은 레벨의 병원은 모든 병을 다 치료할 수 있다고 가정한다.

〈그림 12〉는 계층적으로 구성된 CHCS 모델을 보여준다. 가장 높은 레벨의 연결 모델은 SLARGE-HC이다. SLARGE-HC는 2개의 LARGE-HC로 구성되고 LARGE-HC는 MIDDLE-HC로, MIDDLE-HC는 8개의 SMALL-HC로 구성된다. 각각의 HC 모델은 HOSPITAL 과 VILLAGE의 두 원소 모델을 가지고 있다. 실험 틀(experimental frame)로서 각 VILLAGE는 환자들을 발생시키는 소스가 된다. 각 HOSPITAL은 큐와 정해진 수의 의사들을 갖는 서버가 된다. 각 1/2/3/4계층의 HOSPITAL은 16/4/4/1명의 의사들을 각각 가지고 있다고 가정한다. 다음은 DEVS 형식론에 의해 기술된 CHCS 모델을 D-DEVSim++으로 표현한다. 이제는 CHCS 모델을 병렬 시뮬레이션을 위해서 분할(partition)을 한다. 〈그림 13〉은 가능한 한가지 분할 예이다.

시뮬레이션은 KAICUBE860 시스템에서 이루어 졌다. KAICUBE860은 KAIST에서 개발된 5 차원의 초 격자형(hypercube) 병렬 컴퓨터이다. 각 컴퓨터 노드는 40MHz i860 마이크로 프로세서와 8Mbyte의 메모리, 그리고 store-and-forward 방식의 라우팅을 하는 5 차원 통신 채널을 가진다. 운영체제로서 각 노드는 커널과 그 위에 Express라는 병렬 OS를 가지고, 호스트 컴퓨터의 운영체제는 UNIX System V R4.2이다.

DOHS 방법의 성능 평가를 위해서, 노드 컴퓨터의 수를 1 에서 32개 까지 변화시켜가면서 시뮬레이션 시간을 측정해 보았다. 주의해야 할 점은, 1개의 노드 컴퓨터에서 실행될 때는 D-DEVSim++은 순차(sequential) 모드로 동작한다. 순차 모드에서는 전체 시뮬레이션 시간(global



〈그림 13〉 실험에 사용된 CHCS 모델의 파티션 /은 각 파티션의 레벨의 수를 나타낸다.

simulation time) 계산, 상태 저장, 롤백과 같은 컴퓨터 노드간 동기가 필요 없으므로 이와 관련된 연산을 하지 않는다.

우선, 정리 1의 분석적인 스케줄 시간 speedup을 검증하기 위하여, 0 (%)의 referral rate의 경우에 대해 스케줄 시간을 측정하였다. 0 referral rate는 ρ의 계산을 쉽게 하기 위하여 정한 값인데, 이때 롤백은 전혀 일어나지 않기 때문에 ρ는 1이 된다. 〈표 3〉은 비교 결과를 보여준다. 두 번째 열 (측정된 speedup)의 값들은 coordinator들이 스케줄 메세지들((*)과 (*done*))을 처리하는데 걸린 시간이다. 세 번째 열 (분석된 speedup)의 값을 얻기 위해서는 식 2가 이용되었다. 식 2를 계산하기 위해서는 ρ,P,l,l'의 패러미터들을 알아야 한다. l'을 계산하기 위해서는 각 파티션된 서브 트리의 레벨의 수를 알아야 한다. 각 서브 트리에 대한 l'값은 〈그림 13〉에 나타내었다. 이렇게 구해진 각 서브트리의 l'을 평균을 내면 평균 l'을 얻을 수 있다. 예를 들어서, 노드 컴퓨터의 수가 8 일 때, 평균 $l'=(5+4+3\times6)/8=3.38$ 이다. 〈표 4〉는 이러한 방법으로 구해진 평균 l'을 보여준다. 〈표 3〉에 보인 바와 같이 측정된 값과 분석 값이 어느 정도 일치함을 알 수 있다. 물론, 이 speedup은 $\rho=1$일 때의 값이므로 가장 높은 값이다. Referral rate의 값이 증가할 수록 실제의 speedup은 감소하게 된다.

DOHS 방법의 전체적인 성능을 보기 위하여서는 스케줄 시간에 대한 speedup 뿐만 아니라 모델 실행 시간에 대한 speedup까지 합한 전체 speedup이 중요한 결과가 된다. 전체 시뮬레이션 시간에 대한 speedup은 순차적인 구현에 의한 시뮬레이션 시간과 병렬적 구현에 의한 시뮬레이션 시간과의 비율이 된다. 이때 주의해야 할 점은 순차적 구현에서는 동기화와 관련된 연산들 (전체 시뮬레이션 시간 계산, fossil collection, 상태 저장 등)은 포함되지 않아야 한다는 것이다. 〈그림 14〉는 노드 컴퓨터의 숫자가 변할 때의 speedup 결과이다. 또한 〈그림 15〉는 referral rate가 변할 때의 speedup 변화를 측정한 결과이다.

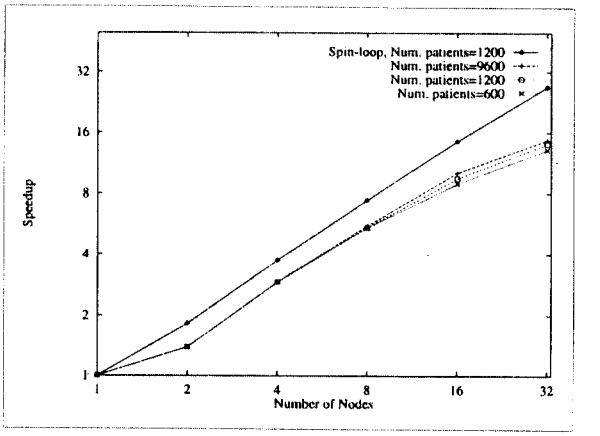
〈표 3〉 스케줄 시간에 대한 speedup

(Referral rate = 10, # patients = 1200)

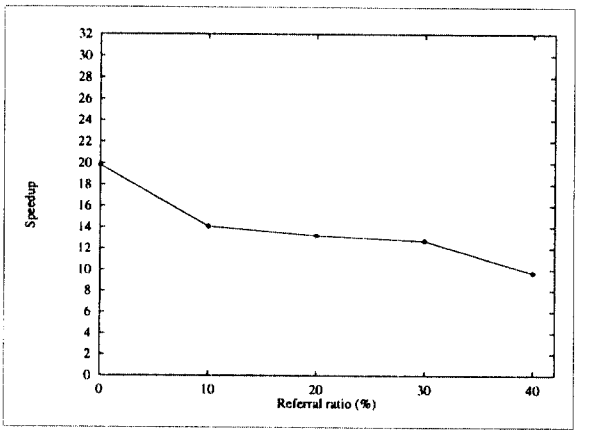
노드의 수	측정된 speedup	분석된 speedup
1	1	1
2	2.2	2.2
4	5.0	5.3
8	10.7	11.83
16	22.5	23.7
32	57.9	61.5

〈표 4〉 분석을 위해 사용된 패러미터들

노드의 수	ρ	평균 l'
1	1	5
2	1	4.5
4	1	3.75
8	1	3.38
16	1	3.38
32	1	2.60

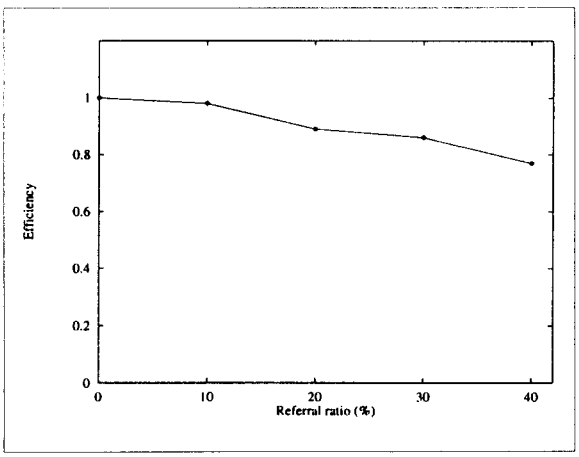


〈그림 14〉 Speedup v.s. 노드 컴퓨터의 수

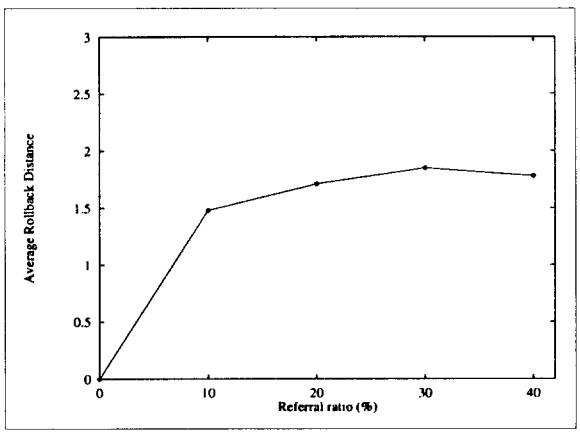


〈그림 15〉 Speedup v.s. referral rate (# nodes=32)

DOHS 방법은 기본적으로 Time Warp기법을 사용하므로 롤백 오버 헤드에 대한 평가를 해보아야 한다. 〈그림 16〉은 효율성(efficiency)을 보여준다. 효율성은 시뮬레이션에 기여한 메시지 수를 전체 메시지 수로 나눈 것으로 낙관적 시뮬레이션이 얼마나 효과를 보았나를 나타내는 성능 지수이다. 또한 〈그림 17〉는 평균적인 롤백 거리를 보여준다. 32 노드에서도 2.28로서 비교적 낙관적 기법이 성공적임을 알 수 있다.



〈그림 16〉 효율성 v.s. referral rate (# nodes = 32)
(# nodes = 32)



〈그림 17〉 평균 롤백 거리 v.s. referral rate (# nodes = 32)
(# nodes = 32)

결 론

이 논문에서는 계층적 DEVS 모델을 위한 분산 시뮬레이션 환경인 D-DEVSim++을 제안하였다.

D-DEVSim++에서는 DEVS 모델의 분산 시뮬레이션을 위해 DOHS 방법을 사용하고 있다. DOHS 방법은 계층적 스케줄링을 하는 추상 시뮬레이터에 Time Warp 프로토콜을 적용하였다. 이을 위해 DOHS-manager 와 DOHS-queue라는 메시지 제어 기법을 제안하였다. 또한 효율적인 롤백을 위해 다음과 같이 세단계로 구분되는 롤백 알고리즘을 제안하였다. 즉 계층적 스케줄 취소, simulator의 상태와 출력 메시지 롤백, 그리고 계층적 재 스케줄링이다. DOHS 방법은 계층적 스케줄링을 병렬화 함으로써 스케줄 시간에 대한 병렬성을 이용할 수 있는데 이로 인한 speedup을 분석하였다. 또한 벤치 마크 시뮬레이션 실험을 통해 이 분석을 검증하였고, D-DEVSim++이 상당한 speedup을 얻을 수 있음도 확인하였다. 앞으로 다양한 시뮬레이션 모델에 대해 실험을 계속해서 시뮬레이션의 성능에 영향을 미치는 여러 이슈들을 연구해보아야 한다.

참고문헌

[1] Alex C. Chow and Bernard P. Zeigler. Parallel DEVS : A parallel, hierarchical, modular modeling formalism. In *Proceedings of the 1994 Winter Simulation Conference*, Orlando, Florida, 1994.

[2] Eric R. Christensen and Bernard P. Zeigler. Distributed discrete event simulation : Combining DEVS and Time Warp. In *Proceedings of the SCS Eastern Multiconference on AI and Simulation Theory and Applications*. Simulation Series, 1990.

[3] A. I. Concepcion. A hierarchical computer architecture for distributed simulation. *IEEE Transactions on Computers*, 38(2) : 311-319, February 1989.

[4] Richard M. Fujimoto. Optimistic approaches to parallel discrete event simulation. *Trans. of the Society for Computer Simulation*, 7(2):153-191, October 1990.

[5] D. Jefferson and H. Sowizral. Fast concurrent simulations using the Time Warp mechanism, *Distributed Simulation*, 15(2) :63-69, 1985.

[6] Ki Hyung Kim. *Distributed Simulation Methodology Based on System Theoretic Formalism : An Asynchronous Approach*. PhD thesis, Korea Advanced Institute of Sicence and Technology, 1996.

[7] Ki Hyung Kim, Tag Gon Kim, and Kyu Ho Park. Events ordering in optimistic distributed simulation of DEVS models. *Journal of the KOREA Society for Simulation*, 5(1):81-90, 1996

[8] Ki Hyung Kim, Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park. Distributed optimistic simulation of hierarchical DEVS models. In *Proceedings of the 1995 Summer Simulation Conference*, page 32-37, Ottawa, Canada, 1995.

[9] Jayadev Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39-65, March 1986.

[10] Yeong Rak Seong, Sung Hoon Jung, Tag Gon Kim, and Kyu Ho Park. Parallel simulation of hierarchical modular DEVS models : A modified Time Warp apporoach. *International J. in Computer Simulation*, 5(3):263-285, 1995

[11] Y.H.Wang. *Discrete-Event Simulation on a Massively Parallel Computer*. PhD thesis, University of Arizona, 1992

[12] B.P.Zeigler. *Theory of Modelling and Simulation*. John Wiley, 1976.

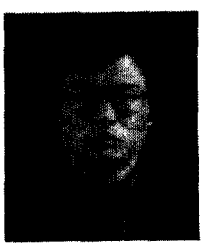
[13] B.P.Zeigler. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press, 1984

[14] B.P.Zeigler and D.Kim. Design of high level modeling/high performance simulation environments. In *10th Workshop on Parallel and Distributed Simulation (PADS'96)*, pages 154-161, 1996.

● 저자소개 ●

김기형

1990년 2월 : 한양대학교 공과대학 전자통신공학과 졸업(공학사)
1992년 2월 : 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1996년 8월 : 한국과학기술원 전기 및 전자공학과 졸업(공학박사)
1996년 9월～현재 : 한국과학기술원 위촉연구원
관심분야 : 분산시뮬레이션, 시스템 분석 및 시뮬레이션, 멀티미디어 시스템, 실시간 고장허용 시스템



김탁곤

1975년 2월 부산 대학교 전자공학과 졸업(공학사)
1980년 2월 경북대학교 전자공학과 졸업(공학석사)
1988년 5월 아리조나대학교 전기 및 전산 공학과 졸업(공학박사)
1975년 2월～1977년 6월 육군 통신장교
1980년 9월～1983년 1월 부산수산대학교 전자통신공학과 전임강사
1987년 8월～1989년 7월 아리조나 환경연구소 연구 엔지니어
1989년 8월～1991년 8월 캔사스대학교 전기 및 전산공학과 조교수
1991년 9월～1993년 8월 한국과학기술원 전기 및 전자공학과 조교수
1993년 9월～현재 한국과학기술원 전기 및 전자공학과 부교수

본학회 논문지 편집위원장
관심분야 : 모델링 이론, 병렬/지능형 시뮬레이션 환경, 컴퓨터시스템

박규호
1973년 서울대학교 전자공학과 졸업(공학사)
1975년 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1983년 프랑스 파리 11대학 졸업(공학박사)
1975년~1978년 동양정밀 개발과장
1983년~1988년 한국과학기술원 전기 및 전자공학과 조교수
1988년~1992년 한국과학기술원 전기 및 전자공학과 부교수
1992년~현재 한국과학기술원 전기 및 전자공학과 교수
관심분야 : 병렬처리, 컴퓨터 구조, 컴퓨터 그래픽스