이산사건 워게임 시뮬레이션을 위한 실시간 병렬 엔진의 설계 및 구현

김 진 수[†]·김 대 석[†]·김 정 국[††]·류 근 호[†††]

요 약

군사용 워게임 시뮬레이션 모델들의 상호연동을 위해서는 국제표준연동(HLA : High Level Architecture)구조를 반드시 갖추어야하며 타 모델과 연동시 발생되는 시스템 오버헤드를 줄이기 위해서는 병렬 시뮬레이션 엔진 도입이 효과적이다. 그러나 기존 군사용 워게임 시뮬레이션 모델엔진의 이벤트 처리는 순차적 이벤트-드리븐 방식으로 처리하고 있다. 이는 병렬로 처리 시 글로벌 자료영역에 대한 동시참조등의 문제점들이 발생하기 때문이다. 아울러 기존 시뮬레이션 플랫폼으로 다중 CPU 시스템을 사용하여도 여러 개의 CPU를 다 활용하지 못하는 결과를 초래하고 있다. 따라서 이 논문에서는 군사용 워 게임 모델의 시스템 처리능력 향상과 글로벌 자료 영역에 대한 동시참조, 대외적인 시뮬레이션 시간처리, 장애 회복(Crash Recovery)시 병행 처리된 이벤트들의 순서를 보장 할 수 있는 객체모델에 기반한 병렬 시뮬레이션 엔진으로의 전환을 제안한다 이 전환된 병렬 시뮬레이션 엔진은 다중 CPU 시스템(SMP)상에서도 병렬 실행이 가능하도록 설계하고 구현하였다.

Design and Implementation of Real-Time Parallel Engine for Discrete Event Wargame Simulation

Jin Soo Kim[†]·Dae Seog Kim[†]·Jung Guk Kim[††]·Keun Ho Ryu[†††]

ABSTRACT

Military wargame simulation models must support the HLA in order to facilitate interoperability with other simulations, and using parallel simulation engines offer efficiency in reducing system overhead generated by propelling interoperability. However, legacy military simulation model engines process events using sequential event-driven method. This is due to problems generated by parallel processing such as synchronous reference to global data domains. Additionally, using legacy simulation platforms result in insufficient utilization of multiple CPUs even if a multiple CPU system is under use. Therefore, in this paper, we propose conversing the simulation engine to an object model-based parallel simulation engine to ensure military wargame model's improved system processing capability, synchronous reference to global data domains, external simulation time processing, and the sequence of parallel-processed events during a crash recovery. The converted parallel simulation engine is designed and implemented to enable parallel execution on a multiple CPU system (SMP).

키워드 : 병렬처리(Parallel Processing), 군사용 워게임 시뮬레이션 엔진(Army War game Simulation), 실시간 병렬 시뮬레이션 엔진 Real-Time Parallel Simulation Engine)

1. 서 론

많은 국가들이 급변하는 국가정세와 예측 불가능한 미래 전략환경에 부합되면서, 보다 경제적이고 과학적인 군사훈련 수단으로 시뮬레이션을 선택하여 개발/발전 시켜오고 있다.

그러나 상호운용성측면이 배제된 상태에서 개발된 1990년대 이전의 군사용 시뮬레이션 모델들은 원래 목적한 이외 분야에 활용하기위해 타모델과 연동한다는 것은 거의 불가능하였다.

따라서 다양한 유형의 시뮬레이션 모델들을 상호연동하고 통합하는 방법을 통해, 보다 복잡한 새로운 목적을 달성할 수 있도록 하기위한 방안들이 요구되었으며, 이러한 시도의 하나로 미 국방성에서는 1995년부터 국제표준연동(HLA : High Level Architecture)를 제안하였고[1], 이는 2000년 9월에 IEEE1516 국제표준으로 등록되었다.

한국군 또한 제대별 훈련용모델들을 독자 개발하여 운영중에 있으며 년 수회 실시되는 한미 연합 연습과 기타 연습시 이들 모델을 운용하기 위해서는 미국, 각군 또는 제대별 모델간 연동이 필수적이다.

이미 개발되어 운용중인 비표준연동 모델을 국제표준연동구조인 HLA 기반 시뮬레이션 모델로 전환 개발하기 위해

† 정 회 원 : 육군교육사 체계분석실
†† 정 회 원 : 한국외국어대학교 컴퓨터공학과 교수
††† 종신회원 : 충북대학교 전기전자 및 컴퓨터공학부 교수
논문접수 : 2003년 4월 4일, 심사완료 : 2003년 5월 30일

서는 객체모델화 및 타 모델과의 연동시 발생하는 시스템 오버헤드를 줄이기위한 효율적인 모델 엔진 재 설계가 필수적이다.

현 운용중인, 대다수의 군사용 워 게임 시뮬레이션 모델 엔진에 있어서 이벤트 처리는 순차적 이벤트-드리븐 방식으로 처리하고 있다.

이는 병렬로 처리시 발생하는 문제점 즉, 글로벌 자료 영역에 대한 동시참조, 대외적인 시뮬레이션 시간처리, 장애 회복시 병행 처리된 이벤트들의 순서 보장 등 때문이다. 따라서 시뮬레이션 플랫폼으로 다중 CPU 시스템을 사용하여도 여러 개의 CPU를 다 활용하지 못하는 결과를 초래하고 있다.

따라서 이 논문에서는 군사용 워 게임 모델의 시스템 처리능력 향상과 글로벌 자료 영역에 대한 동시참조, 대외적인 시뮬레이션 시간처리, 장애 회복시 병행 처리된 이벤트들의 순서 보장을 할 수 있는 객체모델에 기반한 병렬 시뮬레이션 엔진으로의 전환을 제안한다 이 전환된 병렬 시뮬레이션 엔진[2, 3]은 다중 CPU 시스템(SMP)상에서도 병렬 실행이 가능하도록 설계하고 구현하였다.

이 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 새로운 패러다임으로 부상하고 있는 분산 실시간 객체 모델인 TMO(Time-triggered Message-triggered Object)[4]와 이의 수행을 위한 분산 미들웨어 엔진[5-7]에 대해서 소개하고 3장에서는 모델의 이벤트 병렬처리 기법 개발 4장에서는 실험 및 성능분석 5장에서는 결론을 기술한다.

2. 관련 연구

실시간 멀티미디어 서비스[8]나 실시간 시뮬레이션 시스템[9], 실시간 제어 시스템등의 다양한 분야에서는 시간적 제한이 따르는 실시간 시스템을 요구한다. 실시간 시스템을 설계할 때, 각각의 환경에 맞는 독자적인 시스템 보다는 일반성을 지닌 시스템을 개발할 필요가 있으며, 정시 보장 서비스를 시스템 설계자가 시스템을 설계할 때 제공 해야한다. 또 매우 크고 복잡한 시스템의 설계를 위해서는 시스템이나 응용 프로그램 환경에 대해 일정하면서 통합적인 표현 방법이 필요하다.

이러한 요구로 인해 실시간 객체인 TMO 모델이 연구되어져 왔으며 다양한 환경에서 객체모델 수행 플랫폼에 대한 연구[10-14]가 이루어지고 있다. TMO는 정시보장, 객체지향, 분산 환경 등의 특징을 가지는 분산 실시간 객체 모델로써 실시간 시스템의 설계를 단순하게 해준다. TMO 객체모델 수행 엔진인 DREAM(Distributed Real-time Ever Available Micro-computing) 커널[15]이 UCI의 DREAM Lab.에서 구현되었으며, 이는 MS-DOS 기반의 경성 실시간 시스템으로 설계되었다. 반면에 RTDCS(in H.U.F.S.)Lab.에서 개발된 TMO 객체모델 수행 엔진인 WTMOS(Windows TMO System)[16]는 Win32 환경에 미들웨어 플랫폼 형태로 구현되었으며 이는 그래픽 기반의 연성 실시간 시스템으로 설계되었다. 따라서 기존 운영체제의 모든 API를 사용할 수 있어, TMO 스타일의 실시간 프로그래밍과 함께 그래픽, 멀티미디어, 윈도우 관리 등의 서비스를 자유롭게 제공한다.

최근 들어 많은 주목을 받고 있는 유닉스 클론(Clone)인 리눅스 시스템 또한 실시간 작업에 대한 스케줄링과 고해상도 타이머, 브로드 스레드(broad thread), 분산 IPC(Inter Process Commun ication), X-윈도우 시스템 등과 같은 실시간 객체를 수행하기 위한 충분한 조건과 개발 환경을 갖추고 있으며 실시간 분야에서 경성 실시간 시스템을 위한 '실시간-리눅스'와 같은 연구가 많이 이루어지고 있다.

LTMOS(Linux TMO System)는 이러한 리눅스 환경에서 X-윈도우 그래픽 기반 미들웨어 형태의 TMO 객체모델 수행 플랫폼이다.

2.1 TMO

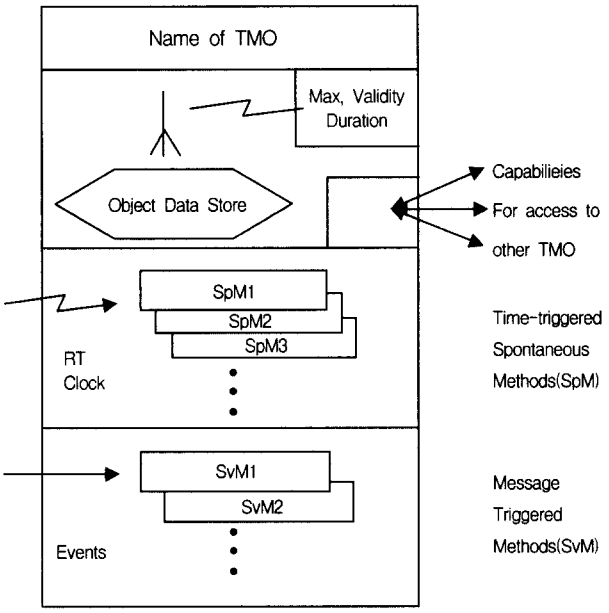
TMO는 정시보장 컴퓨팅(Timeliness Guaran teed Computing)을 목표로 제안된 실시간 객체 모델로서 다음과 같은 특징을 갖는다.

- 경성/연성 실시간 응용뿐만 아니라 일반적인 분산 병행 프로그램 응용에도 사용할 수 있는 객체모델이다.
- 설계시 시간 보장 개념을 제공한다.

TMO의 Method는 특징적인 두 가지 부류로 나누어지는데, 하나는 시간 조건에 의해 동작하는 SpM(Time-triggered Method : Spontaneous Method)이고, 다른 하나는 분산 IPC 메시지에 의해 동작하는 SvM(Message-triggered Method : Service Method)이다. SpM에는 그 주기 및 수행 deadline 등으로 구성되는 시간제약(Timing Constraint)인 AAC가 주어지며, SvM에는 메시지 수신에 의한 구동 이후의 처리를 마칠 때까지의 종료시간(deadline)이 주어진다.

전술한 SpM과 SvM이 객체내의 Object Data Store Segment를 동시에 접근할 때의 Mutual Exclusion을 위해 SpM이 SvM 보다 높은 우선 순위를 가지는데 이것을 BCC(Basic Concurrency Constraints)이라 한다.

TMO 모델의 실시간 데이터는 최대 유효기간(Maximum Validity Duration)이 부과되는데 이 시간이 경과되면 그 데이터는 유효성을 잃게 된다. TMO의 구조는 (그림 1)과 같다.



(그림 1) TMO의 구조

2.2 LTMOS(Linux TMO System)의 기능 및 구성

2.2.1 LTMOS의 기능

LTMOS[6]는 현재 주목 받고 있는 실시간 객체 모델인 TMO의 주요 기능을 C++객체로 구현한 실시간 객체네트워크의 수행 엔진으로서, 리눅스 운영체제 위에 개발된 미들웨어 플랫폼이다. 개발환경은 리눅스 커널 2.2.X버전과 C++ 언어를 위한 g++컴파일러 및 리눅스 스레드를 위한 Linux Pthread 라이브러리, X-윈도우 시스템의 GUI(Graphic User Interface)를 위한 GTK(Gimp Tool Kit)라이브러리를 사용하였다.

LTMOS가 TMO 응용 프로그램을 위해 제공하는 기능과 구현된 LTMOS의 특징을 다음과 같다.

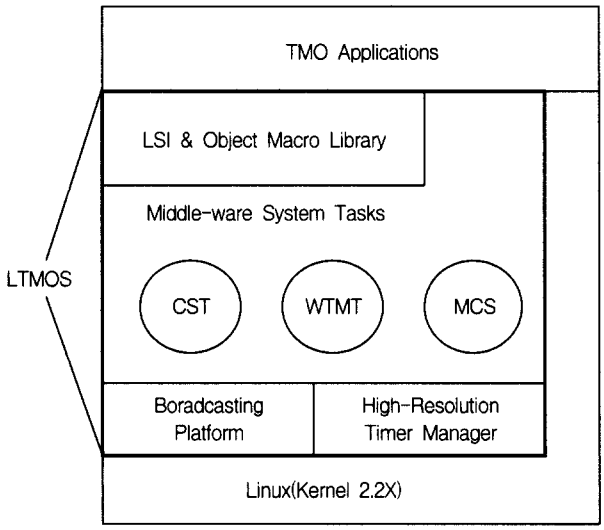
- TMO는 C++객체로 정의된다.
- SpM과 SvM은 TMO의 구성요소로 정의되는 스레드 군으로 class 구성요소로서의 scope rule의 적용을 받는다.
- SpM의 on-time invocation 및 deadline scheduling.
- 1/1000초 시간단위의 정시 수행과 데드 라인 감시 및 예외 처리
- 분산 IPC 메세지의 도착에 의해 구동되는 메소드인 SvM의 수행 및 deadline scheduling
- BCC의 제공대신 TMO 객체내의 공유 데이터의 병행 접근을 위해 CREW(Concurrent-Read Exclusive-Write) 모니터를 객체 라이브러리에서 제공한다.
- Deadline exception handler를 제공한다.
- C++의 모든 객체지향프로그래밍 특징을 제공한다.
- Linux 시스템에서 수행되는 각종 library를 사용할 수 있다.
- SvM thread pool의 생성 및 관리
- 분산 환경에서의 클럭(clock)동기화
- 실시간 객체 TMO 응용에 윈도우 인터 페이스를 제공하기 위한 그래픽 인터페이스 통신을 제공한다.

2.3.2 LTMOS의 계층구조

LTMOS는 리눅스 커널 위에서 동작하면서 실시간 객체 모델 TMO로 설계된 TMO 응용 프로그램에 대한 수행을 맡는다.

- Layer 0 : 하드웨어 제어, 윈도우 관리, 실시간 스케줄링, 멀티 스레딩, 동기화 도구, 분산 IPC 도구
- Layer 1 : 고해상도 클럭 관리기, 브로드 캐스팅 플랫폼
- Layer 2 : TMO 메소드의 시간 조건에 의한 구동과 우선순위 조절, 데드라인 감시 및 예외 처리기 구동, 분산 IPC 메시지 및 윈도우 메시지 처리, 분산노드의 클럭 동기화
- Layer 3 : 실시간 메소드 관리, 분산 IPC 관련 함수들, 기타서비스 관련 함수들, TMO 응용 프로그래머를 위한 매크로(macro) 라이브러리.
- Layer 4 : TMO 응용 프로그램.

이러한 기능을 지원하는 C++ TMO 수행 환경 LTMOS는 (그림 2)와 같이 5단계의 계층 구조로 구성된다.



LSI : LTMOS Service Interface
CST : Clock Synchronization Task
WTMT : Watchdog TMO Management Task
MCS : Message Communication System

(그림 2) LTMOS의 계층 구조

3. 모델의 이벤트 병렬처리 기법 개발

이 절은 모델의 TMO 구조화 및 객체 구조화를 바탕으로 이벤트의 TMO SvM Pool에 의한 병렬처리 기법 개발 내용에 대해 기술한다.

모델의 병렬처리시 해결 해야할 조건들은 다음과 같다

- 이벤트 처리중 모의시간 정지
- 이벤트 처리중 생성되는 이벤트도 시간 순 처리 보장
- 이벤트 병렬화 시의 동시 자료저장 접근방지
- 병렬화시 장애회복의 일관성 유지
- 시각이 다른 이벤트의 병렬 처리 중에도 대외적인 모의시간은 일정한 하나의 것으로 제공되어야 한다.

위와 같은 모든 조건을 만족시키기 위해

- 병행처리가 가능한(즉, 동시 자료 접근에 의한 종속성 문제가 없는 이벤트) 이벤트의 판별기법
- 이벤트 처리중 새로운 이벤트의 처리를 시작하는 patch 기법
- 이벤트 병행 수행 중의 모의시간 관리기법
- 이벤트 수행중, 파생되는 이벤트의 처리기법
- 중도의 이벤트 파생으로 기 처리 중이던 이벤트가 잘못(처리순서) 시작된 것으로 판별되었을 때의 처리기법을 제안하고 이를 바탕으로 병렬처리 시스템을 설계 구현하였다.

다음은 병렬처리와 관련된 용어정의, 병렬처리, 시스템구조, 제안기법들에 대해 순차적으로 기술한다.

3.1 이벤트의 병행/병렬 처리와 관련된 용어 정의

병행처리시 그 결과가 서로 영향을 주지 않는 이벤트들을 "상호간에 독립성이 있다"고 하고 이와 관련된 사항 및 용어들을 다음과 같이 정의 기술한다.

[정의 1] 이벤트

시뮬레이션 운영 도중 발생하는 이벤트를 E_i로 표기한다. 이때 i는 이벤트의 발생순서를 나타내는 일련 번호이다.

[정의 2] 이벤트 시간(이벤트-time)

어떤 이벤트 E_i의 처리 시각을 $t(E_i)$로 표기한다. $t(E_i)$는 이벤트 E_i 처리 기간 중, 이벤트에 대한 모의 시간으로 변화하지 않는다.

[정의 3] 파생 이벤트

워 게임 참가자 등의 외부 입력에 의해 발생하는 이벤트가 아니고, 어떤 이벤트의 처리 도중 내부적으로 발생시키는 이벤트를 파생 이벤트라 정의한다. E_i에 의해 발생한 파생 이벤트 E_k의 처리 시간은 항상 $t(E_k) >= t(E_i)$를 만족한다.

[정의 4] 시뮬레이션 시간 및 입자도(granularity) 시뮬레이션 시스템과 대외적인 사용자들에게 유일하게 제공되는 모의시간을 말하고 Sim-time로 표기한다. 시뮬레이션 시간의 입자도(granularity)는 시뮬레이션의 시간변화의 단위를 말한다.

시간변화의 단위 안에서 처리하는 이벤트는 그 이벤트 시간이 동일한 것으로 간주한다.

컴퓨터 시스템의 시간은 Sys-time으로 표시한다.

[정의 5] 선행 이벤트, 후행 이벤트, 동시 이벤트

이벤트 E_i를 기준으로 E_k의 처리 시각이 앞설 때 ($t(E_i)) > t(E_k))E_k$를 E_i의 선행 이벤트, 그 반대의 경우를 ($t(E_i)) < t(E_k))$ 후행 이벤트라 한다. $t(E_i) = t(E_k)$인 경우 두 이벤트를 동시 이벤트라 정의한다.

[정의 6] 대기 이벤트 큐(WaitQ)

현재 실행을 위해 대기중인 이벤트들로 시간 및 발생 순서에 의해 정렬된 이벤트의 집합을 말한다(동시 이벤트인 경우 발생 순서에 의해 정렬된다). WaitQ = $\{E_i, \cdots, E_k\}$로 표시한다.

[정의 7] 우선대기 이벤트 (E_{front})

대기 이벤트 집합에서 최우선으로 처리되어야 할 이벤트를 우선대기 이벤트라 하고 E_{front}로 표시한다. 우선 대기 이벤트의 시간을 $t_{wait} = t(E_{front})$로 표시한다.

이벤트의 병렬/병행 수행을 SvM thread pool내의 고정 thread의 개수를 P_n, 이중 idle한 스레드의 개수를 P_{avail}로 표시한다.

[정의 8] 실행 이벤트 큐(RunQ)

현재 SvM pool에 의해 병행으로 처리중인 이벤트들의 집합을 실행 이벤트 큐 RunQ = $\{E_i, E_j, \cdots, E_k\}$라 표시한다. 실행 이벤트 집합의 최대 cardinality는 병행 처리 스레드 pool의 스레드 개수이다(degree of parallelism = P_n).

[정의 9] 이벤트의 자료 종속성

이벤트 E_i와 E_k의 수행 시간에 관계없이 E_i와 E_k가 같은 자료 영역을 접근(Read-Write, Write-Write)하여 병행 실행될 경우 상호의 수행 결과에 영향을 미치는 경우 두 이벤트는 자료 종속성이 있다고 한다(이하 종속성으로 칭한다. 전투는 모든 이벤트와 종속성이 있는 것으로 간주한다).

[정의 10] 배타적 이벤트(Exclusive Event)

근접전투 이벤트와 같이 영향을 미치는 전역자료의 범위를 예측하기가 불가능하여 자료 종속성 판별이 어렵거나, 병렬처리는 가능하지만 이를 위해서는 더 큰 오버헤드를 초래하는 이벤트들은 모든 순차성을 위해 SvM pool을 비우고 단독으로 실행시킨다. 이와 같은 이벤트를 배타적 이벤트라 한다.

[정의 11] 병행성 이벤트(Concurrent Event)

병행 실행이 가능한 이벤트의 종류를 말한다. 병행 실행 이벤트는 아래와 같이 엔진의 상황 및 이벤트의 시간에 따라 결정성 이벤트와 비결정성 이벤트로 나누어진다.

[정의 12] 결정성 이벤트(Deterministic Event)

이벤트의 처리 중에 자료에 대한 write나 처리 중간 결과 전송을 실행 도중 그때 그때 결정적으로 할 수 있는 이벤트를 말한다. 다른 이벤트와 병렬 실행된다는 점 이외에 처리 방식은 배타적 이벤트와 동일하다.

[정의 13] 비결정성 이벤트(Non-deterministic Event)

이벤트의 처리 도중이나 처리 종료 후에, 이벤트의 처리 결과가 파기되어 이벤트 실행 이전으로 원상 복구될 가능성이 있는 이벤트이다. 이벤트 중에 발생하는 자료에 대한 갱신나 중간 결과 전송 등과 같이 모의 시스템에 변화를 주는 행위는 그 결과를 즉각 시스템에 반영하지 않고, 시스템 변경 내용을 이벤트별 실행 기록 E_{log}에 보존하여, 이벤트 수행 종료 후에 이의 반영 여부를 결정하여야 하는 이벤트이다.

[정의 14] 이벤트의 완료(commit)

이벤트 실행 종료 후에 실행 중 반영하지 못한 결과를 시스템에 반영하는 행위를 말한다.

- 배타적 이벤트와 결정성 이벤트의 경우는 이벤트 실행 중에 이미 실행 결과를 반영하였으므로 이벤트 종료 후의 완료(commit)에서는 모의 시간의 갱신만이 일어나게 된다.
- 비결정성 이벤트의 경우는 보존된 실행 행위 자료 E_{log}에 의거 그 결과를 시스템에 반영하고 모의 시간을 갱신하게 된다.

이벤트 실행 결과 반영의 종류는 다음과 같이 분류할 수 있다.

- 부대 등 국부 자료의 변경
- 광역 자료의 변경
- 클라이언트로의 변경 결과 송신(중간 서버 경유)
- 이벤트 종료후의 시뮬레이션 시간 변경

[정의 15] 이벤트의 복귀(rollback)

비결정성 이벤트의 실행 결과를 시스템에 반영하지 않고 파기하는 행위를 말한다. 이 경우 이 이벤트는 다시 WaitQ에 삽입된다.

[정의 16] 실행 이벤트의 이벤트 로그 (E_{log})

이벤트의 완료(commit) 또는 복귀(rollback)을 위해 이벤트 처리 중에 유지되는 이벤트의 실행 결과 기록을 말하고 E_{log}로 표시한다. 이벤트의 E_{log}는 완료(commit) 또는 복귀(rollback)후에 삭제된다.

위 [정의 16]에 따라 E_{log}의 구성은 이벤트의 결정성/비결정성 여부 및 이벤트의 종류에 따라 (그림 3)과 같이 다르게 구성된다.

구 분	결정성 이벤트 및 배타적 이벤트	비결정성 이벤트
E_{log} 내용	[1] 이벤트 시간 및 정보 (결정성, 비결정성, 단독 실행여부, 종속성 여부 등) [2] 이벤트 실행 종료 여부	[1] 이벤트 시간 및 정보 (결정성, 비결정성, 단독 실행여부, 종속성 여부 등) [2] 이벤트 실행 종료 여부 [3] 이벤트 종류별 자료 변화 기록 [4] 이벤트 처리 결과 전송 패킷 [5] 처리 중 생성 이벤트 리스트 [6] reschedule 정보 [7] 전투집합 merge 정보

(그림 3) 이벤트 Log의 구성

위에서 이벤트 실행 종료 여부는 이벤트의 실행은 종료되었으나 아직 완료(commit)나 복귀(rollback)이 결정되지 않은 것을 말한다.

E_{log} 엔트리는 이벤트가 SvM Pool에 patch될 때 생성되며, 이벤트 처리 종료시에 E_{log}에 이벤트 처리 종료 사실이 기록되며, 이벤트의 완료(commit) 이후에 삭제된다.

실행 이벤트의 로그 큐(LogQ)란 SvM pool에 의해 실행이 시작되어 완료(commit)나 복귀(rollback)로 완결되기 이전까지의 모든 이벤트들의 E_{log}들을 이벤트의 시간순에 의해 갖는 리스트를 말한다.

LogQ 상의 front에 존재하는(가장 이벤트 시간이 빠른) 이벤트를 $E_{log_{front}}$로 표시한다.

SvM에 의한 이벤트의 처리 연산이 종료되는 것을 이벤트 실행 종료라 하고 이벤트 처리후의 완료(commit) 또는 복귀(rollback)까지 완료되는 것을 이벤트의 완결이라 한다.

또한 [정의 16]에 따라 실행중인 이벤트는 SvM pool의

SvM하나를 차지하고 있으므로 RunQ에 있으면서 LogQ에도 해당 entry를 가진다. 이벤트의 실행이 종료되면 SvM을 반환하므로 RunQ에서 이벤트는 삭제된다. 그러나 LogQ의 entry는 이벤트가 완료(commit) 될 때까지 유지되므로 RunQ는 LogQ의 subset이다.

[정의 17] 이벤트 patch

WaitQ의 우선대기 이벤트 E_{front}를 실행 이벤트 큐로 옮기는 것을 이벤트 patch라 한다.

이벤트 patch의 조건이란 이벤트의 처리 시간이 되었을 때, 이벤트의 patch 조건은 배타적 이벤트와 병행성 이벤트에 따라 그 조건이 달라진다. 배타적 이벤트는 실행중인 이벤트가 없어야 하고, 병행성 이벤트의 경우는 쉬고 있는 SvM pool의 thread가 있고, 모든 처리중인 이벤트와 자료 종속성이 없어야 한다.

patch시의 결정성(deterministic) 이벤트의 판별은 patch된 이벤트의 이벤트 시간 $t(E_k)$가 LogQ 이벤트 집합의 이벤트 시간 중 가장 빠른 시간의 이벤트와 동시 이벤트일 경우 E_k는 결정성 이벤트이다 ($t(E_k) = t(E_{\log_{front}})$).

patch시의 비결정성(non-deterministic) 이벤트의 판별은 patch된 이벤트의 이벤트 시간 $t(E_k)$가 LogQ 이벤트 집합의 이벤트 시간 중 가장 빠른 시간의 이벤트보다 늦은 시간일 경우 E_k는 비결정성 이벤트이다.

3.2 이벤트 병행처리 시스템의 구조설계 및 기법

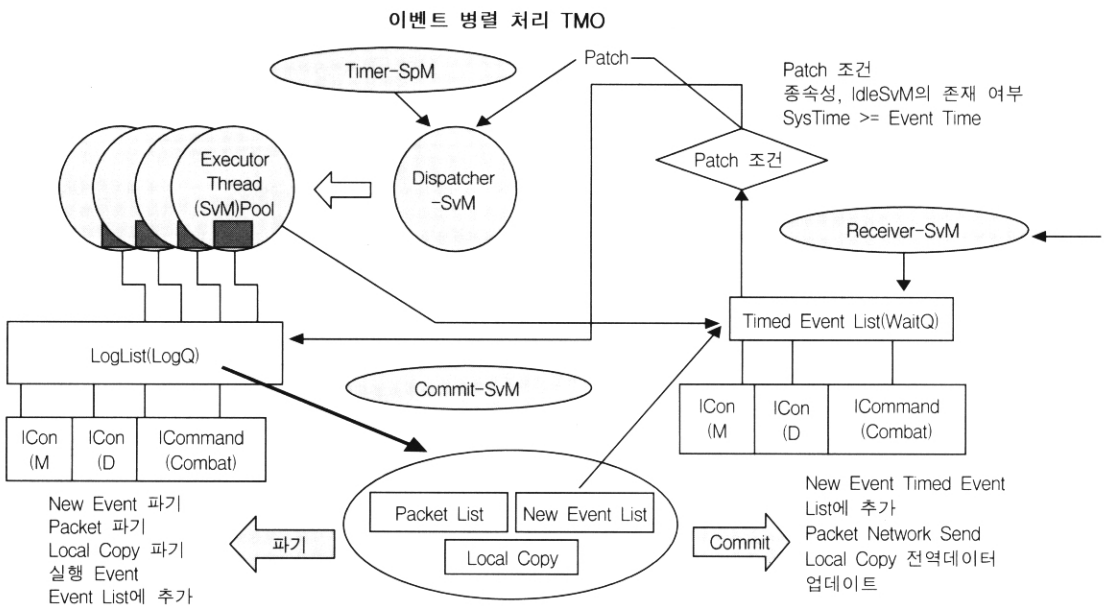
이벤트의 병행 처리를 위한 TMO로 구현된 이벤트-병렬 처리 엔진의 구조는 (그림 4)와 같다.

TMO로 구현된 이벤트 병행 처리 엔진의 각 구성 요소의 기능은 다음과 같다.

- Receiver-SvM : IGI 서버로부터 각종 명령을 접수하여 시간순으로 정렬되는 대기 이벤트 큐(WaitQ)에 삽입하는 역할을 하는 스래드이다.
- Timer-SpM : 주기적으로 Dispatcher-SvM에 메시지를 보낸다.
- Dispatcher-SvM : Executor-SvM 종료시 메시지에 의해 구동되어 병행/병렬이벤트 처리를 진행시킨다. 이벤트 patch를 위한 처리는 별도로 기술한다. 또한 처리 이벤트가 없을 때 Timer-SpM의 주기적 메시지에 의해 구동되어 시뮬레이션 시간을 갱신 한다.
- Executor-SvM : 이벤트 병행/병렬 처리의 주체가 되는 thread pool의 구성 요소 SvM이다. Dispatcher-SvM으로부터 이벤트 객체를 메시지로 넘겨받아 이벤트의 처리를 수행한다.
- Commit-SvM : Executor-SvM으로부터 한 이벤트의 실행 종료 시마다 이를 통보 받아 LogQ에 존재하는 실행 종료 이벤트들의 완료(commit) 및 복귀(rollback) 여부를 결정한다. 현재 실행이 종료된 이벤트라 할지라도 선행 이벤트의 실행이 종료되지 않은 경우는 완료(commit) 여부를 결정할 수 없으므로 매 이벤트의 처리 종료시 마다 이벤트 시간 순으로 완료(commit) 여부를 검사한다.

3.2.1 Dispatcher-SvM의 기능

Dispatcher-SvM은 매 초마다 Scheduling SpM으로부터



(그림 4) TMO로 구현된 이벤트 병행/병렬 처리 엔진의 구조

메시지를 받아 patch를 시도하게 되고 Executor-SvM으로부터 이벤트 처리 종료 메시지를 받으면 이벤트 patch를 시도한다. 우선 대기 이벤트를 patch하여 Executor-SvM pool로 보낼수 있는 patch 조건은 다음과 같다.

① 배타적 이벤트의 patch 조건

전투와 같은 단독 실행 이벤트는 다른 병렬 이벤트의 수행이 없이 단독으로 실행되어야 하므로 다은 조건이 모두 만족될 때 patch될 수 있다.

[조건 1] Sys-time >= t(E_{front})이다.

[조건 2] 현재의 P_{avail}이 $P_{avial=n}$이다(처리 중인 이벤트가 없다).

② 병행 실행 이벤트의 patch 조건

우선 대기 이벤트 E_{front}가 patch되려면 다음의 조건을 모두 만족하여야 한다.

[조건 1] Sys-time >= $t(E_{front})$이다.

[조건 2] 현재의 P_{avail}이 $P_{avail} > 0$이다.

[조건 3] 완료(commit) 이전의 실행중인 모든 이벤트(LogQ 상의 모든 이벤트)와 E_{front}사이에 자료 종속성이 없다.

③ patch시의 결정성(deterministic) 이벤트의 판별

patch된 이벤트의 이벤트 시간(E_k)가 LogQ 이벤트 집합의 이벤트 시간 중 가장 빠른 시간의 이벤트와 동시 이벤트일 경우 E_k는 결정성 이벤트이다 ($t(E_k) = t(E_{\log\_front})$).

④ patch시의 비결정성(non-deterministic)이 벤트의 판별

patch된 이벤트의 이벤트 시간 $t(E_k)$가 LogQ 이벤트 집합의 이벤트 기산 중 가장 빠른 시간의 이벤트보다 늦은 시간일 경우 E_k는 비결정성 이벤트이다.

3.2.2 Executor-SvM의 기능

이벤트 실행을 위한 SvM thread pool에 속한 Executor-SvM은 Dispatcher-SvM으로부터 이벤트 객체를 수신하여 이벤트를 병행 실행하는 메소드이다.

3.2.3 배타적 및 결정성 이벤트의 처리 과정

배타적 및 결정성 이벤트로 결정되는 것은 이벤트가 Dispatcher-SvM에서 결정되어 진다.

Dispatcher-SvM이 E_{Log}의 flag을 배타적(결정성) 이벤트로 설정해 놓게 되고 각 이벤트는 실행되면서 E_{Log}의 flag를 확인하며 E_{Log}에 기록할지 즉석에서 업데이트 할지를 결정하게 된다.

모든 데이터를 즉석에서 업데이트 하며 실행종료시점에서 E_{Log}의 상태를 완료(commit) 할 수 있는 상태로 기록을 하고 Commit-SvM에게 메시지를 보내게 된다(E_{Log}의 구성 참조). 이후 TMO(Engine) 의해 다른 이벤트를 실행하게 된다. E_{Log}의 생성 이유는 Commit-SvM에 의하여 시뮬레이션 시간의 업데이트가 이루어 져야 하기 때문이다. 배타적 이벤트 및 결정성 이벤트에서 새로이 발생하는 이벤트의 시작 시간 또한 E_{Log}에 들어있는 이벤트 시간으로부터 파생되어 진다.

3.2.4 비결정성 이벤트의 처리과정

Log의 flag가 비결정성 이벤트로 마크되어 있다면 이후 모든 업데이트 데이터는 E_{Log}에 저장(기록) 되어진다. 이벤트의 실행종료 시점에서 E_{Log}의 상태를 완료(commit) 할 수 있는 상태로 기록하고 Commit-SvM에게 메시지를 보내고 TMO(Engine)에 의해 다른 이벤트 실행하게 된다. 새로운 이벤트 및 reschedule되는 이벤트의 시간은 모두 현재 이벤트의 기간으로부터 파생 되어진다. 비결정성 이벤트의 경우 중도 포기의 경우도 있으나 로그의 형태는 변화되지 않는다. 다만 E_{Log}의 Commitstate(flag)변수에 이 로그 이벤트는 중도 포기되었다는 것을 표시해주고 이벤트를 WaitQ로 다시 넣어주면 된다. 이후 Commit-SvM에 의하여 Log에 남아있는 것은 폐기된다.

3.2.5 Commit-SvM의 기능

Commit-SvM의 이벤트 실행 종료 후의 처리를 전담하는 SvM으로 매 이벤트의 실행 종료 시마다 깨어나서 LogQ에 존재하는 이벤트들 중 종료된 것을 시간 순으로 완료(commit) 또는 복귀(rollback)한다. LogQ는 이벤트의 시간 순으로 정렬되어 있으므로, 이를 scan하여 처음 entry부터 연속적으로 종료된 것만을 완료(commit) 또는 복귀(rollback)한다. 도중에 종료되지 않은 것이 등장하면 실행을 중지한다. 즉 종료되지 않은 이벤트의 뒤에 있는 이벤트가 종료되었다 하더라도 이는 아직 완료(commit) 시기가 아니므로 이를 연기한다.

① 이벤트의 완료(commit)및 복귀(rollback)의 조건

이벤트 처리 종료시에 해당 이벤트가 다음 조건을 만족하면 완료(commit)한다.

[1] LogQ의 가장 앞의 entry이고 ($E_{\log\_front}$)

[2] t_{wait} >= $t(E_k)$일 때

이벤트 처리 종료시에 해당 이벤트가 다음 조건을 만족하면 완료(commit) 을 연기한다.

[1] LogQ의 가장 앞의 entry이고 ($E_{\log_{front}}$)

[2] $t_{wait} < t(E_k)$이고

[3] E_{front}와 자료 종속성이 없으면

이벤트 처리 종료 시에 해당 이벤트가 다음 조건을 만족하면 이벤트를 복귀(rollback)한다.

[1] LogQ의 가장 앞의 entry이고 ($E_{\log_{front}}$)

[2] $t_{wait} < t(E_k)$이고

[3] E_{front}와 자료 종속성이 있으면

완료(commit)시의 모의 시간 갱신은 다음 절의 모의 시간 관리에 관한 절에서 일괄 기술한다.

3.2.6 모의 시간 관리기법

모의 시간 관리의 원칙은 다음과 같다.

이벤트 시각이 중첩되는 이벤트들을 병행/병렬 처리하더라도 대외적으로는 유일한 모의 시간이 유지되어야 한다. 이와 함께 이벤트별로는 이벤트 처리중 모의 시간(이벤트 시간)이 흐르지 않아야 한다. 이벤트의 과부하로 모의 시간이 시스템 시간보다 늦어졌을 때에는 이벤트가 없어 시스템이 idle 할 때에 이를 보정한다. 이러한 원칙 하에 모의 시간의 관리는 두 부분에서 일어나는데 그 하나는 Dispatcher-SpM이고 다른 하나는 완료(commit)시의 모의 시간 갱신 부분이다. 각 단계에서의 모의 시간 관리 원칙 및 알고리즘은 다음과 같다.

Dispatcher-SpM의 모의시간 갱신의 원칙은 다음과 같다.

- 이벤트가 LogQ가 빈 상태에서 최초 patch되는 것이면 이벤트의 시간으로 모의 시간 설정
- LogQ가 있는 상태에서는 이벤트의 patch가 있건 없건 모의 시간 불변
- 실행중인 이벤트도 없고 patch도 없을 때에는 시스템 시간으로 모의 시간 설정

위와 같은 원칙에 의한 patch시의 모의 시간 갱신은 다음과 같다.

- patch 이벤트가 없을 때의 모의 시간 갱신

patch 조건을 만족하는 이벤트가 없을 경우에, 현재 실행중인 이벤트가 LogQ에 있으면, 실행 중인 이벤트가 시작 될 때 모의 시간의 설정이 있었고 이의 완료(commit)가 일어날 때 모의 시간이 갱신되므로 현재로서는 모의 시간 갱신이 필요 없게 된다. LogQ가 비어있어 실행 중인 이벤트가 없을 경우에는 우선 대기 이벤트가 없거나 처리 시간이 아직 안된 것이므로 단순히 시스템 시간으로 모의 시간을 갱신한다.

if (no LogQ entry) Sim-time = Sys-time ;
else no action ;

- patch 이벤트가 있을 때의 모의 시간 갱신

patch 조건을 만족하는 이벤트가 있을 경우에도, 현재 실행중인 이벤트가 LogQ에 있으면, 실행 중인 이벤트가 시작 될 때 모의 시간의 설정이 있었고 이의 완료(commit) 이 일어 날 때 모의 시간이 갱신되므로 현재로서는 모의 시간 갱신이 필요 없게 된다. LogQ가 비어 있는 경우는 우선 대기 이벤트가 최초로 실행되는 것이므로 patch 이벤트의 시간으로 모의 시간을 설정한다.

if (no LogQ entry) Sim-Time = t_{wait} ;
else no action ;

완료(commit) 시의 모의 시간 갱신처리과정은 아래와 같다.

Case (no more entry in LogQ and no entry in WaitQ) :
Sim_time = Sys_time,
Case (no more entry in LogQ and entry exists in WaitQ) :
Sim_time = min(Sys_time, t_{wait})
Case (entry exists in LogQ and no entry in WaitQ) :
Sim_time = t(E_{log_front})
Case (entry exists in LogQ and entry exits in WaitQ) :
Sim_time = min(t(E_{log_front}), t_{wait})

3.2.7 이벤트 자료 종속성 판별 기법

이벤트의 자료 종속성은 앞서 정의된 바와 같이 한 이벤트의 처리에 의한 전역 자료의 수정이 다른 이벤트에 영향을 미치는가 하는 것을 판별하는 기준이 된다. 즉, 모든 이벤트는 처리시각(commit 시각)이 순차화 되어 있으므로 같은 자료 영역을 수정하는 이벤트는 반드시 종속성이 있게 된다. "이동", "전투", "탐지" 이벤트에 대한 처리만을 수행한 이번 연구 범위 때문에 이와 같은 종속성을 모든 이벤트에 적용되는 것으로 기술하기는 어렵지만 세 가지 이벤트를 중심으로 이벤트의 종속성을 기술하면 다음과 같다.

- "이동", "탐지" 이벤트는 지리적으로 처리 범위가 한정되는 것이 보통이므로 각 이벤트의 지리적 중심 위치에서 일정한 작전 반경(최대 작전 범위, 최대 자료 수정 범위)을 주어 이 반경의 겹침이 있는 지로 종속성을 판별한다.
- "전투"와 같이 작전 반경을 사전에 예측할 수 없는 이벤트는 안정성을 위해 모든 이벤트와 종속성이 있는 것으로 간주한다.
- 이벤트 처리 중 파생되는 이벤트나 "탐지" 프로세스와 같이 계속적인 수행 스케줄이 예약되어 있는 경우, 각 이벤트는 소구간 단위로 구분하여 종속성 여부를 분리 처리

한다.

이러한 이벤트간 자료의 종속성은 각 이벤트 형태의 쌍에 대해서 그 판별 기법의 정리되어야 한다. 다음은 금번 연구에서 사용된 각 이벤트 쌍별의 자료 종속성 판단 기준이다.

● 이동 대 이동

두 부대가 병행 수행 후 상호간에 영향을 주지 않을 거리는 5.6KM이다. 이동 후 탐지를 수행하는데 5.0KM가 탐지의 범위이고 0.3KM가 구간이동 간격이다. 탐지의 경우엔 변화되는 것이 없지만 이동의 경우는 변화되는 것이 있으므로 0.3KM은 상호배제적인 거리이고 5.0KM는 공유할 수 있는 지역이다.

● 이동 대 탐지

이동 대 이동과 동일하게 탐지의 5.0KM는 공유할 수 있는 지역이고(데이터의 변화를 주지 않으므로) 0.3KM의 거리는 이동 부대에 의해 변경될 수 있는 지역이다. 따라서 이동 대 탐지의 경우는 두 부대의 거리가 5.3KM 이상일 경우 병행수행이 가능하다.

● 탐지 대 탐지

탐지 이벤트는 5.0KM의 지역을 탐지하지만 데이터에 대한 변경은 가해지지 않는다. 데이터를 읽어 들여서 타 이벤트를 생성하는데 사용된다. 데이터에 대한 변경이 이루어지지 않으므로 종속성이 없음을 알 수 있고 탐지 대 탐지의 경우는 항상 병렬 수행이 가능하다. 그러나 타 이벤트를 생성하는데 있어서 중복생성이 가능하므로 Lock, Unlock의 적절한 사용을 통해 중복되는 이벤트의 생성하는 경우에 대해 고려해 주어야 한다.

● 전투 대 모든 이벤트

전투의 경우 많은 부대가 참여하게 되고 이 모든 부대의 데이터를 변경 할 수 있는 거리와 다른 이벤트의 거리를 계산해야한다. 이벤트를 실행하며 즉석에서 업데이트 하는 것 보다 종속성을 계산하는 오버헤드가 더 클 수 있다는 판단에서 전투는 모든 이벤트와 종속성이 있다고 판별하고 베타적 이벤트로 실행 단독 수행한다.

3.2.8 이벤트의 roll-back을 위한 Local copy 기법

비결정성 이벤트의 경우는 실행 시작 당시에는 종속성이 없었으나 실행 후 파생 이벤트와의 종속성이 발생할 수 있으므로, 이미 처리된 이벤트를 완료(commit) 하기 이전에 무효화하여 원상 복구할 수 있도록 하여야 한다. 이벤트의 실행 결과를 전역자료에 반영하지 않고 기록하였다가 완료(commit)가 결정되면 전역 자료에 반영하고 복귀(rollback) 시에는 파기하기 위해 이벤트 별 E_{Log}의 Local copy를 사용한다. Local copy의 내용은 이벤트 종류별로 다르므로 본 연구에서 병행 처리 이벤트로 취급된 이동과 탐지를 위한 Local copy에 대해 기술한다. 다만 이벤트의 처리 도중에도 그 흐름이 경우마다 다르므로 사용하는 전역 변수도 달라지게 된다. 다만 이벤트의 모든 흐름에 대한 전역 변수의 Local copy를 유지하려면 그 크기가 방대해져 병행처리의 효율성보다 Local copy의 유지가 더 큰 오버헤드가 되므로 본 연구에서 이동과 탐지의 Local copy로 사용한 전역 변수의 범위를 초과하는 이벤트 실행 처리의 흐름에 대해서는 병행 처리를 중도 포기한다. 다음은 이벤트 별 Local copy의 내용이다.

- E_{Log}의 공통 local copy
  - 네트웍전송 패킷의 저장, 새로운 이벤트의 저장
  - E_{Log}의 실행 이벤트(Event Time이 기록되어 있다.), E_{Log}의 상태
  - E_{Log}가 결정성 혹은 비결정형인지 나타내는 flag
- 이동의 local copy
  - 새로 생성된 detectSet
  - 새로이 이동할 이동좌표
  - 현재 이벤트에서 이동중인 Unit
- 탐지의 local copy
  - 새로이 생성된 CombatSet
  - 삭제될 CombatSet
  - 현 이벤트의 DetectSet
  - 삭제될 AttackSet flag(DetectSet에서 두부대를 Attack에서 제외시키는 경우)
- 이동 및 탐지 이벤트 처리 흐름에서 local copy 사용 기법
  - 데이터의 업데이트가 이루어지는 부분에서 E_{Log}기록하는 경우는 AddLog(변경데이터)를 통하여 기록되어하고 추가와 삭제가 이루어질 경우는 AddLog(변경데이터, Add), AddLog(변경데이터, Del)로써 구현되었다.
  - Local copy에 기록된 사항을 재 사용할 경우는 메소드의 파라미터에 E_{Log}를 넘겨주어서 E_{Log} 기록된 값을 사용하도록 한다.

Event Rollback-log
Event 처리 중 수정하는 객체 자료의 주소와 초기 값 등
Event 처리 중 생성된 Event 목록
IGI 서버로 보내지는 Flush buffer에 저장할 내용

(그림 5) Rollback log의 자료 구조

3.2.9 이벤트 실행중의 Local copy 확장에 따른 이벤트 병렬처리 중도파기

부대의 거의 모든 자료를 갱신하는 특정 이벤트(예를 들면 balance 호출)는 메모리에 대한 overhead가 매우 크다. 이러한 이벤트는 병렬처리를 중지하고 이벤트를 순차적 수행으로 다시 시작하게 된다. 이때의 중도 포기 알고리즘은 (그림 6)과 같다.

1. E_{Log}의 상태가 비결정형 이벤트 이여야 한다.
2. Local copy의 기록 량이 오버헤드가 클 경우 E_{Log}의 상태를 'Delete'로 기록한다.
3. E_{Log}의 이벤트를 단독실행 이벤트로 표시한다.
4. E_{Log}의 이벤트를 waitQ에 넣어준다
5. E_{Log}의 Event를 NULL로 설정한다.
6. Idle SvM의 개수를 증가시킨다.
7. Dispatcher-SvM에게 메시지를 보낸다.

(그림 6) 중도포기 알고리즘

4. 실험 및 성능 분석

워게임 모델의 시스템 성능에 영향을 주는 두 가지 주요 요소는 이벤트처리 시간과 네트웍 트레픽에 소요되는 시간이다.

또한, 시스템 전체 시간 중 80%가 이벤트 처리 시간이며 이중 70%가 "탐지"이벤트 처리 시간이다. 이는 전장의 논리적 특성상, 수천 개의 탐지 이벤트가 동시에 이루어지기 때문이다.

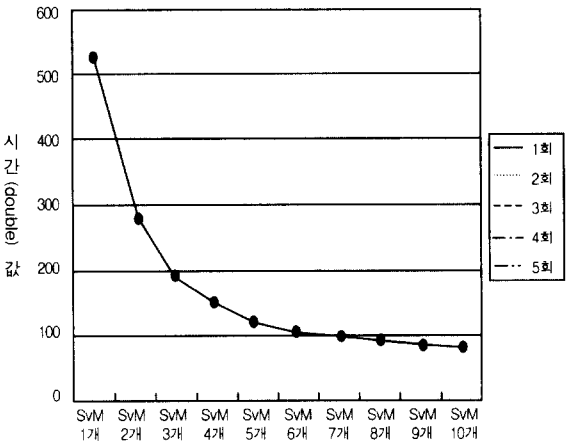
이벤트 처리 관점에서 볼 때, 본 시스템은 4개의 CPU를 사용했을 때 50%의 처리 시간을 감소시켰다. 그러나 이러한 대폭적인 처리 시간의 감소에도 불구하고 게임어가 잘 느끼지 못할 수도 있다. 왜냐하면 군사용 워게임 모델은 이벤트마다 처리시간이 미리 정해지며 정해진 시간에따라 처리되는 이산 사건 시뮬레이션 시스템의 특성을 가지기 때문이다.

4.1 병렬처리 성능 분석

현재 군에서 운용중인 "창조21"모델의 성능향상과 미군, 타군(해군, 공군)의 모델과의 연동을 위해 객체지향/TMO를 적용한 "표준연동 지상모델(가칭)"로 재개발 중에 있으며 이 모델의 지상전투분야 모의엔진을 이용(C++로 작성)하여 병렬처리 성능분석을 하였으며 실험환경은 리눅스 6.2 운영체계하 처리속도 500MHZ의 CPU 4개를 사용한 다중 CPU 시스템(SMP) 상에서 부대 생성 후 특정위치로 이동 시켰을 때 완료(commit)되는 이벤트 수가 1000개가 될 때까지 걸리는 시간을 SvM 개수별로 측정하였다.

그리고 수치적 분석의 편이성을 위해 기본 이벤트 처리 시간에 0.5초를 부가 할당 후 수행하였다. 그 수행 결과는 <표 1>로 정리하였다

성능분석결과 SvM 1개 사용할 경우 기존 시스템(순차적 방식)과 처리시간이 비슷하였으며, (그림 7)에서 보는바와 같이 SvM의 개수가 많아질수록 병렬처리의 효과가 높아진다. 이러한 결과에 따라, 모델 운용시 동시에 처리해야 할 이벤트가 너무 많아 시뮬레이션 시간이 한동안 진행되지 못하는 상황을 제거 할 수 있는 효율적인 시스템 구현이 가능하였다.



(그림 7) 병렬처리(TMO 이용) 성능 분석

5. 결 론

순차적 이벤트-드리븐 방식으로 운영중인 군사용 워게임 시뮬레이션의 성능을 향상시키기 위해이 논문에서는 분산 실

<표 1> 병렬처리 성능분석

	SvM 1개	SvM 2개	SvM 3개	SvM 4개	SvM 5개	SvM 6개	SvM 7개	SvM 8개	SvM 9개	SvM 10개
1회	521.5887	272.7447	193.8007	150.8236	123.2167	107.2237	93.63762	87.35371	80.64973	73.1608
2회	524.1957	274.7512	190.4649	150.7856	123.1766	107.6006	94.60165	87.17081	79.94467	73.05697
3회	521.5889	272.1559	190.6497	149.4873	123.3479	107.458	95.07191	88.55906	80.0467	74.1671
4회	524.6039	274.1556	193.9568	151.2927	123.1507	108.2117	96.84454	88.55786	81.47463	72.04493
5회	520.5523	272.7689	192.2437	150.7626	123.6787	108.4478	94.54158	85.91367	78.39086	72.68368

시간 객체 TMO를 이용하여 다중 CPU 시스템(SMP)상에서 병렬 실행이 가능하게 설계하고 구현하였다.

시스템을 수행한 결과 다중 이벤트 처리 SvM Pool을 두고 다중 CPU 시스템에서 병렬 처리 가능한 이벤트들을 최대한 병렬 처리함으로써 대폭적인 성능향상을 가져오게 되었다. 또한 시뮬레이션 시간의 일관성을 유지시키고 완료/복귀 개념도입에 의한 이벤트 처리 순서의 일관성을 모두 유지하였다 이것은 이벤트 처리 부하의 대부분을 차지하는 탐지 및 이동 이벤트들을 병렬 처리함으로써 가능하였다. 또한 구조적으로 복잡한 병렬처리의 기능 부품들을 각 기능을 전담하는 SpM이나 SvM 및 SvM Pool을 차후 모든 이벤트를 병렬 처리하는 엔진으로 확장이 쉽도록 설계하고 구현하였다.

추후 연구과제로는 구현된 병렬처리 엔진의 전 기능 및 전 이벤트로 확장하기 위해 각 이벤트 종류의 쌍에 대해서 자료 종속성 판별 기법의 정의에 대한 연구가 필요하다.

참 고 문 헌

[1] DoD, USD(A&T), DoD Modeling and Simulation Master Plan, October, 1995.

[2] 박용우, 김문회, 김정국, "TMO 모델 기반 실시간 시뮬레이션", 정보처리학회논문집, 제5권 제4호, pp.67-74, 1998.

[3] 김정국, "실시간 시스템을 위한 미들웨어", 정보처리학회논문집, 제8권 제5호, pp.30-37, 2001.

[4] K. H. Kim and H. Kopetz, "A Real-time Object Model RTO.k and an Experimental Investigation of Its Potentials," Proc. 18th IEEE Computer Software and Applications Conference, pp.392-402, November, 1994.

[5] J. G Kim, M. H. Kim, B. J. Min and D, B. Im, "A Soft Real-Time TMO Platform-WTMOS-and Implenentation Techniques," Proc. 1st IEEE International Symposium on Object-oriented Real-Time Distributed Compution, pp. 256-264, April, 1997.

[6] J. G. Kim and S. Y. Cho, "LTMOS : An Execution Engine for TMO-Based Real-Time Distributed Objects," Proc. Of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Vol. 5, pp.2713-2718, 2000.

[7] Kane Kim, et. al, "A Timeliness-Guaranteed Kernel Model : DREAM Kernel and Implementation Techniques," RTCSA, 1995.

[8] Jung-Guk Kim, J. P. Hong, Byoung-Joon Min, Moonhae Kim, "Modeling of Multimedia Services using the TMO Model," Journal of Computer Systems Science and Engineering, 1998.

[9] K. H. Kim, "Real-Time Simulation Techniques Based on the RTO.k Object Modeling," Proc. 20th IEEE Computer Software & Application Conference, August, 1996.

[10] 김문회, 김정국, 김광희 "Time-triggered Message-triggered Object Modeiling of a Distributed Real-timeControl Application for its Real-time simulation," Proc. Of COMPSAC 2000, Taiwan, Vol.24, pp.549-556, 2000.

[11] 민병준, 김정국, 김문회, "Implementation of a Run-time Monitor for TMO Programs on Windows NT," Proc. Of the International conference on Parallel and Distributed processing Techniques and Application, Las Vegas, Vol.5, pp. 2689-2695, 2000.

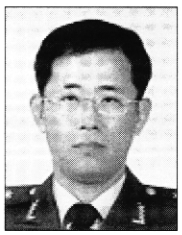
[12] 김정국, "분산 실시간 플랫폼 TMO 엔진과 Adaptive Real-time scheduler의 개발", 정보산업공학논문집, Vol.4, 2000.

[13] 김문회, 김정국, 민병준, 양승민, "실시간 시스템 모형", 정보과학회지, 제14권 제8호, 통권 제87호, pp.15-21, 1996.

[14] 정영준, 김정국, 박용우, 김문회, "실시간 객체를 이용한 원자력 발전소 Safety Injection System의 Modeling 및 실시간 시뮬레이션", 한국정보과학회 가을학술발표논문집, Vol.25, No.2, pp.487-489, 1998[RTS].

[15] M. H. Kim and J. G. Kim, "An Environment for Real-Time Simulation Based on the TMO Model," Proc. Int. Conference on PDPTA, Jun., 1999.

[16] Kim, J. G., Kim, M. H., Min, B. J. and Im, D. B., "A soft Real-Time TMO platform-WTMOS-and Implementation Techniques," Proc ISORC '98, pp.256-264, April, 1997.



김 진 수

e-mail : kjs9990@yahoo.co.kr

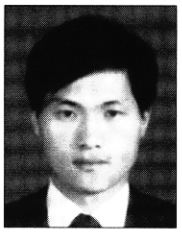
1981년 계명대학교 수학과(학사)

1987년 국방대학교 전산학과(이학석사)

1998년 충북대학교 전산학과 박사과정 이수

2002년~현재 육군교육사 체계분석실 근무

관심분야 : 실시간 객체, 분산컴퓨팅, 시공간 데이터베이스



김 대 석

e-mail : daeseog@prumail.co.kr

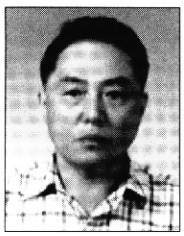
1986년 육군사관학교 졸업(학사)

1996년 국방대학교 전산학과(이학석사)

2003년 충남대학교 전산학과(이학박사)

1996년~현재 육군교육사 체계분석실 근무

관심분야 : 표준연동모델, 실시간 객체, 분산컴퓨팅



김 정 국

e-mail : jgkim@hufs.ac.kr

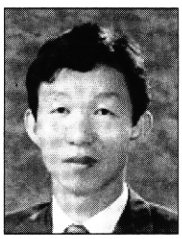
1977년 서울대학교 계산통계학과(학사)

1979년 한국과학기술원 전산학과(이학석사)

1986년 한국과학기술원 전산학과(공학박사)

1983년~현재 한국외국어대학교 컴퓨터 공학과 교수

관심분야 : 실시간객체, 분산컴퓨팅, 실시간운영체제, 내장형 시스템



류 근 호

e-mail : khryu@dblab.chungbuk.ac.kr

1976년 숭실대학교 전산학과(공학사)

1980년 연세대학교 공학대학원 전산전공 (공학석사)

1988년 연세대학교 대학원 전산전공 (공학박사)

1976년~1986년 육군군수지원사전산실(ROTC 장교), 한국전자통신연구소(연구원), 한국방송통신대 전산학과(조교수) 근무

1989년~1991년 Univ. of Arizona Research Staff(TempIS 연구원, Temporal DB)

1986년~현재 충북대학교 전기전자 및 컴퓨터공학부 교수

관심분야 : 시간 데이터베이스, 시공간 데이터베이스, Temporal GIS, 객체 및 지식베이스 시스템, 지식기반 정보검색 시스템, 데이터 마이닝, 데이터베이스 보안 및 Bio-Informatics 등