논문 2011-2-2

온라인 게임에서의 하이브리드기법을 적용한 동기화 기법

A Synchronized Scheme Applying on Hybrid in On-Line Game

김혜영*

Hye-Young Kime

요 약 고속 네트워크의 급속한 발전과 인터넷의 확산 및 컴퓨터 성능의 고급화로 여러 명의 사용자들이 동시에 게임을 진행 할 수 있도록 하는 다중 사용자용 온라인 게임에 대한 요구와 관심이 높아지고 있다. 이러한 다중 사용자용 온라인 게임에서 중요한 요소인 사실적인 게임 진행을 실감하기 위해서는 동기화가 중요한다. 따라서 본 논문에서는 온라인 게임에서의 게임 유저들 간의 효율적인 상태 동기화를 위해 FSM (Finite State Machine) 과 이벤트 잠금 (Event holding) 기법을 결합한 동기화 기법과 코드의 최적화 기법을 제안하고, 이를 적용한 게임서버 및 클라이언트를 구현하여 여러 클라이언트들과의 게임 실행을 통한 테스트를 통해 제안 기법의 효용성과 신뢰성을 보였다.

Abstract Because development of high speed network, spread of internet, and high quality of computer performance, request and internet about massive multiplayer playing the game, is increasing. In order to experience realistic game play which is one of the most importance factor in massive multiplayer on-line, synchronization is importance matter. We propose synchronized and optimized scheme that combined FSM (Finite State Machine) and event holding method for efficient state synchronization for massive multiplayer on-line, and we show the effectiveness and reliability of our proposed scheme through the implementing and testing of the game server applying on our proposed scheme.

Key Words : Game Server, Synchronization, Finite State Machine

Ⅰ. 서 론

컴퓨터 하드웨어는 점차 고속처리가 가능하고 대용량 메모리를 갖추는 방향으로 고급화됨으로써, 고도의 그래픽, 물리엔진, 그리고 인공지능 분야에서도 처리 할 수 있는 범주가 늘어나고 있다.[1] 또한 고속 네트워크의 급속한 발전과 인터넷의 확산으로 여러 명의 사용자들이 동시에 게임을 진행 할 수 있도록 하는 다중 사용자용 온라인 게임에 대한 요구와 관심이 높아지고 있다.[2] 이러한 다중 사용자용 온라인 게임에서 중요한 요소인 사실적인 게임 진행을 실감하기 위해서는 실시간 상호작용이 중요하다.[3] 즉, 한 서버와 통신하는 모든 클라이언트들이 일치되는 게임 상태를 공유하도록 하는 동기화가 중요한 문제가 된다. 그러나 하나의 게임 사이트에서 발생한 게임 진행상의 이벤트들이 인터넷을 통해서 다른 게임 유저 측에 전달되는 과정에서 발생하는 네트워크 지연은 게임 진행시의 동기화를 저해하게 된다.[4]

따라서 본 논문에서는 온라인 게임에서의 실감나는 게임 플레이를 위해 게임 유저들 간의 효율적인 상태 동기화를 위해 FSM (Finite State Machine) 과 이벤트 잠금 (Event holding) 기법을 결합한 하이브리드 기법을 제안하고, 이를 적용한 게임서버 및 클라이언트를 구현하여 여러 클라이언트들과의 게임 실행을 통한 테스트를 통해 제안 기법의 효용성과 신뢰성을 보였다.

본 논문의 구성은 다음과 같다. 뒷장에서는 동기화 기

*정회원, 홍익대학교 게임학부
접수일자: 2010.11.15, 수정일자: 2011.3.24
게재확정일자: 2011.4.8

법에 대한 관련 연구를 보이고, 3장에서는 실시간 게임 엔진을 위해 제안하는 배경 및 기법을 자세히 설명하고, 코드의 최적화 기법을 설명하며, 4장에서 제안 한 기법을 적용한 실시간 게임 서버 및 클라이언트의 설계 및 구현에 대해 나열하고, 5장에서는 제안 기법을 적용하여 구현한 게임과 기존의 온라인 게임서버의 성능 분석을 통해 제안 기법의 효용성을 보였다. 마지막 부분은 향후 연구 방향 및 결론으로 구성하였다.

Ⅱ. 관련연구

FSM (Finite State Machine) 이란, 유한개의 상태를 가진 오브젝트들을 특정 조건에 의해 다음 상태로 보내주는 방법으로, 상태 전이도를 그려 설계하고, 상태 테이블을 직접 구현하여 사용한다. 상태 전이도를 구현하기 위한 방법으로 행동과 조건 두 가지로 분리되어 저장하였다. 그림 1에서 원형안의 내용은 행동을 의미를 하고, 화살표 위의 텍스트는 조건을 의미한다. 그림 2에서와 같이 미리 구현된 FSM을 각각의 몬스터가 가지고 있으며, 이 행동 패턴에 따라 행동하게 된다. 1인 게임 플레이 환경처럼 네트워크가 적용되지 않은 환경에서의 FSM 조건 메시지는 대부분 매 프래임마다 검사되고 조건을 던지게 된다. 예를 들면, 순찰 상태에 놓인 몬스터는 매 프레임 조건을 받게 된다. '순찰 상태'에 있던 이 몬스터가 만일 '데미지를 입는다' 라는 조건을 받는다면, 상태 테이블에 의거하여 현재 상태와 조건 모두 일치하는 것이 있으므로 다음상태인 '공격한다' 상태로 행동을 바꾸게 된다. 그러나 '공격한다' 라는 상태에서 '시야에 적이 있다'라는 조건이 들어 온 경우는 현재상태와 조건의 쌍이 모두 일치하지 않으므로 다음 상태를 정의할 수 없다. 이 경우 몬스터에게 보내진 메시지는 무시되며 그대로 계속 '공격한다' 라는 상태를 가지게 된다.

이 방법은 구현이 간단하고, 작은 수의 상태일 때 메모리도 크게 요구하지 않아서 가장 전형적인 기법 중 하나로 많은 게임들이 이 알고리즘을 따르고 있다. 그러나 상태가 늘어날수록 상태테이블은 기하급수적으로 복잡해지는 단점을 가진다.[5]

이벤트 잠금 (Event Holding)이란 하나의 서버와 여러 대의 클라이언트 상에서 서버가 클라이언트에게 일정 시간이나 혹은 프레임을 진행하라는 명령을 수행하는 형식으로 동기화를 진행하는 방식이다. 즉, 매 주기마다 서버와 클라이언트는 송/수신을 진행해야 하며, 만일 진행하지 못할 경우 다음 프레임을 진행시킬 수 없기 때문에 게임의 진행이 멈춘다. 일정 간격 동안 발생한 메시지를 모으며 호스트는 각 클라이언트에게 메시지를 받는다. 만일, 어느 하나의 클라이언트라도 메시지를 보내지 않으면 그 즉시 게임을 정지 또는 드랍 (drop) 시키고, 모든 클라이언트 메시지를 받았을 때만 게임을 진행시킨다. 여기서 진행이란, 일정 프레임 수를 처리하는 것을 의미한다. 일정 간격 동안 프레임을 처리할 권한은 오로지 호스트만이 부여할 수 있다.[6]

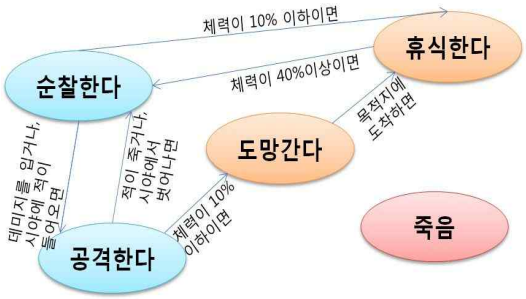

그림 1. FSM 상태전이도의 예
Fig. 1. An Example of the FSM

현재 상태	조건	다음 상태
순찰한다	데미지를 입는다	공격한다
순찰한다	시야에 적이 있다	공격한다
순찰한다	체력이 10%이하이다	휴식한다
공격한다	시야에 적이 없다	순찰한다
공격한다	적이 죽었다	순찰한다
공격한다	체력이 10%이하이다	도망간다
도망간다	목적지에 도착했다	휴식한다
휴식한다	체력이 40%이상이다	순찰한다
죽음	컨디션없음	죽음

그림 2. 상태 테이블의 예
Fig. 2. An Example of the State Table

Ⅲ. 제안기법

게임 내에는 다양하고 수많은 오브젝트들이 존재하며, 이 오브젝트들을 절대좌표(DX Programming에서 사용하는 3차원 (x, y, z)벡터 혹은 2차원 (x, y)벡터)로 동기화 시켜준다는 것은 거의 모든 오브젝트가 움직이고 있

기 때문에, 이에 사용되는 패킷의 양이 방대하다. 예를 들어 패킷의 기본형이 처리의 최소단위 4bytes (size + head)이며 2차원 좌표를 넘긴다면 float형이 2개이므로 8bytes. 즉, 최소 12bytes가 필요하게 된다. 오브젝트가 200개라고 가정하면, 모든 오브젝트들을 한 번에 동기화 시키는데 필요한 패킷양은 12*200 = 2400 bytes이다. 이를 받아 처리하는 데에 200개의 패킷을 DX Rendering 없이 처리해야 하기 때문에 성능상의 심각한 문제를 발생시킬 수 있다. 따라서 본 논문에서는 서로 각자의 클라이언트들 화면에서는 약간의 상이함이 발생할 수 있지만, 서로간의 움직임에서는 부드럽고 자연스럽게 보여 질 수 기법으로 FSM과 이벤트 잠금 기법을 혼용하여 동기화를 수행하는 기법을 제안하고 이를 위한 코드의 최적화를 보였다.

1. FSM과 이벤트 잠금기법을 혼용한 동기화 기법

실제 동작에서 어떤 프로세스에서 한 몬스터가 순찰하는 상태에서 플레이어를 발견하고 공격하는 이벤트가 발생했다고 생각해보자. 이 상황에서 해당 프로세스는 호스트에게 이런 메시지가 발생했다는 것을 알려주고, 호스트는 순찰에서 공격으로 상태가 전이된 몬스터 ID (처음 Initialize 시킬때 부여되는 오브젝트의 고유 번호)와 바뀌어진 다음상태, 그리고 타겟 ID만을 묶어 브로드캐스팅하게 된다. 이렇게 되면, 게임을 진행하는 모든 프로세스는 해당 몬스터가 타겟 ID를 가진 유저를 공격하는 모습을 볼 수 있게 된다. 또, FSM과 그에 해당하는 패킷 메시지를 연결함으로써, 패킷의 크기는 획기적으로 절약된다. 그러나, 위와 같은 구현환경에서의 테스트 결과에서 시간이 흐르면 흐를수록 동기화가 점점 크게 벌어지는 것을 확인할 수 있다. 이는 각 프로세스마다 이벤트의 발생이 완벽히 동시간대에 이루어 질 수 없기 때문인데, 이를 위해서 이벤트 잠금 기법을 사용한다. Event Holding기법으로 진행할 수 있는 프레임(혹은 시간)을 묶어두어 어느 하나의 클라이언트가 과도하게 프레임을 진행하거나 혹은 적은 횟수를 처리함으로써 벌어지는 차이를 줄이기 위함이다

그림 3에서 제안 기법을 적용한 동기화를 보였다. 그림 3에서의 1번과 2번 흐름에서 FSM 메시지는 각각의 프로세스에서 발생한 메시지다. 클라이언트는 FSM 메시지가 발생한 것을 호스트측에 패킷신호로 보낸다. 이를 받은 호스트는 각각의 클라이언트 FSM요청을 한 버퍼에 모은 뒤, FSM_STATE_ACK를 BroadCasting한다. 이러한 순서대로 패킷의 네트워크상 전송시간을 무시한 실제 프레임이 진행되는 시간과 메시지에 따른 처리가 동일하게 된다.

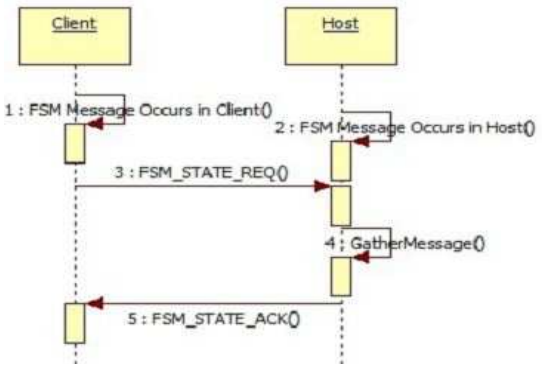

그림 3. 제안기법을 적용한 동기화
Fig. 3. Synchronization applying on proposed scheme

이러한 방법으로 게임 안에서 함께 플레이를 진행하는 플레이어들의 모든 오브젝트의 FSM 상태는 동기화가 이루어지게 된다. 이 방법은 DX에서 float형으로 이루어진 벡터좌표를 동기화 하는 것이 아니라, 같은 메시지를 가진 오브젝트들이 같은 시간 프레임을 진행하면서 벡터좌표의 세밀한 오차발생을 감수하고 개별 클라이언트화면에서 자연스러움을 연출할 수 있다.[7]

2. 코드의 최적화 기법

네트워크 모듈에서의 패킷 기본구조는 cPacket Class로부터 이를 상속받은 각각의 패킷이 생성된다. cPacket은 unsigned short형 변수 Size와 Head를 담고 있으며, 나머지 구체화 된 패킷들은 추가 정보를 담을 변수들을 가진다.[8]

Network Module을 제외한 DX Programming의 Rendering 처리 순서는 [PreRender > Render > PostRender] 식으로 진행된다. Network Model이 WinProc에서 메시지를 감지해주는 상태이기 때문에, 여기에 Network가 추가된다면, [PreRender >Render >PostRender >(메시지가 있으면, PacketParser) >PreRender >Render …] 가 된다.

그림 4는 PacketParser의 처리 함수 호출의 가장 기본적인 헤더를 나타낸다.

cParser::PacketParsing(cPacket* _Packet) 함수안에서 _Packet >Head로 값을 확인하여 그에 맞는 함수를 호출해주는 방식이다. 모든 패킷의 경우의 수를 if else 문을 이용하여 parsing을 했는데, 이때 패킷 Header의 종류가 200개라면, 최상의 경우에는 1번째에, 최악의 경우에는 200번째에 확인된다.

```
class cParser {
public :
 void    PacketParsing(cPacket* _Packet);
 void    OnChattingPacket(cPacket* _Packet);
 void    OnReadyPacket(cPacket* _Packet);
 void    OnStartPacket(cPacket* _Packet);
void     OnConnectPacket(cPacket* _Packet);
};
```

그림 4. PacketParser의 처리 함수의 헤더
Fig. 4. Header of the PacketParser function

따라서 검색 효율은 O(N) 이다. 패, 다양한 패킷 종류를 사용하는 경우엔 효율이 떨어지게 된다. 따라서 이를 개선하기 위해 함수 포인터와 자료구조 Map을 연관시킴으로써, 최적화를 이끌었다. 검색속도는 언제나 ln(N)이지만, 삽입/삭제의 경우 그 시간이 List나 Vector보다 오래 걸리기 때문에 고정된 자료들을 계속해서 검색하는 작업에 알맞은 자료구조인 트리구조를 PacketParser 에 적용하였다. 트리 구조를 가진 Map은 두 개의 Key를 갖으며 함수 포인터와 해더의 Enum값을 묶어서 저장하였다. [Header, FuncPtr]로 묶인 정보는 처음 Network Module이 생성될 때, 모든 종류를 삽입시킨 후, 그 뒤로는 계속 검색을 위해서만 사용한다.

```
if (BasicPacket->Header == MSG_CONNECT_REQ)
  OnConnectREQ() ;
else if (BasicPacket->Header
         == MSG_DISCONNECT_REQ)
     OnDisConnectREQ() ;
          ….
```

그림 5. 코드의 최적화를 적용하지 않은 if문의 모든 경우 수 비교
Fig. 5. A comparison of the "if" command which do not applying optimization of code

그림 5에서는 코드의 최적화를 적용하지 않은 경우이며, 그림 6에서는 코드의 최적화를 적용한 기법을 보인다.

```
std::map <ePacketType, FuncPtr>::iterator Itr ;
Itr = m_MapFuncPtr.find((ePacketType)(BasicPacket
->Header)) ;
if (Itr->second != NULL)
{
 (this->*m_MapFuncPtr[(ePacketType)(BasicPacket
->Header)])(BasicPacket) ;
}
```

그림 6 코드의 최적화를 적용한 경우
Fig. 6. A case applying optimization of code

그림 6에서와 같이 패킷의 경우를 모두 검사해야 하기 때문에 효율성도 낮아질 뿐 아니라, 가독성도 굉장히 떨어지게 된다. 그러나 그림 7에와 같이 제안 기법인 코드의 최적화를 위해 Map을 사용할 경우 효율성 면에서도 우수하며, 단지 패킷이 추가될 때에 클래스의 생성자에 함수 포인터만 추가시키면 되기 때문에 확장 및 재사용성이 증진된다.

Ⅳ. 성능평가

본 논문에서 제안한 동기화 기법을 사용하여 떨어진 LAN 환경에서 Windows XP운영체제가 설치된 클라이언트PC 4대와 서버PC 1대를 이용해서 성능평가를 수행했다.

서버에서 하나의 이벤트 패킷을 받아서 모든 클라이언트에게 넘겨줄 때까지의 시간을 클라이언트의 개수별로 그래프로 나타내 보았다. 그림 7에서와 같이 클라이언트의 개수에 따라, 각 클라이언트 개수 별로 하나의 이벤트를 처리하는 시간의 최대값과 최소값을 비교해보면 50ms가 넘지 않는다. 또한 각 클라이언트 개수별로 평균값은 대체적으로 클라이언트 개수에 관계없이 비슷하다. 15ms에서 30ms의 값을 많이 보여주는데, 이유는 서버 측에서 메시지 잠금 기법을 사용하면 지연시간이 있어서 서버가 각각 클라이언트들과 패킷을 주고받는 시간이 밀리지 않고 충분하기 때문에 서버는 클라이언트 개수의 증가에 따른 네트워크 부하의 증가 없이 원활한 패킷 전송을 할 수 있다.[9]

그림 8의 그래프는 클라이언트를 세 개로 고정을 시켜놓고 실제 게임에서처럼 메시지를 증가시키면서 시간에 따른 메시지 처리시간을 측정한 것이다.

그림 7의 그래프와 비교해 보면 하나의 명령을 처리하는 시간이 그리 큰 차이가 나지 않았다. 메시지가 아무리 증가한다 하더라도 한 번의 고정프레임에서 처리하는 명령은 한 개이고, 명령이 없다 하더라도 고정프레임에서 한 번의 빈 메시지 패킷이 전송되기 때문에 결국 각 고정프레임 당 전송되는 패킷은 한 개로 동일하다.

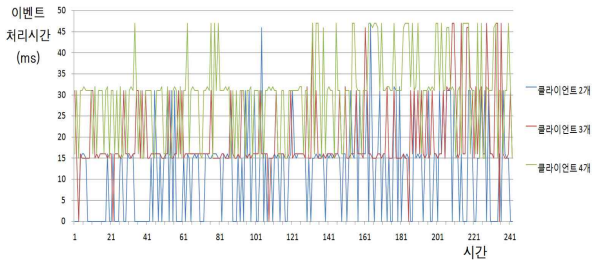

그림 7. 시간에 따른 클라이언트 개수별 메시지 처리 시간
Fig. 7. Message processing time by client number by time

그림 7과 시간이 약간 차이나는 것은 명령이 늘어나서 클라이언트에서 처리하는 로직이 추가되어서 이런 결과가 나온 것이다. 즉 명령이 아무리 늘어나도 서버에 걸리는 부하는 명령이 없을 때와 별 차이가 없다.

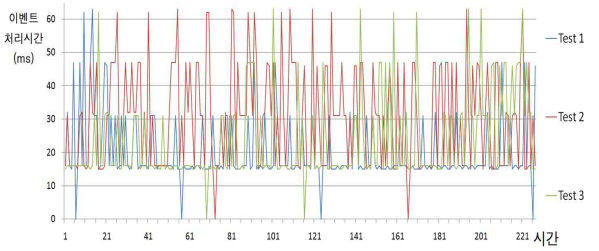

그림 8. 메시지를 증가시켰을 때 시간에 따른 메시지 처리 시간
Fig. 8. Message processing time by time when increased message

V. 결 론

본 논문에서는 온라인 게임에서의 실감나는 게임 플레이를 위해 게임 유저들 간의 효율적인 상태 동기화를 위해 FSM (Finite State Machine)과 이벤트 잠금 (Event holding) 기법을 결합한 하이브리드 기법을 제안하고, 이를 적용한 온라인 게임서버 및 클라이언트를 구현하여 여러 클라이언트들과의 게임 실행을 통한 테스트를 통해 제안 기법의 효용성과 신뢰성을 보였다. 성능분석에서 보인바와 같이 본 논문에서의 제안기법은 클라이언트 개수의 증가와 메시지 증가에 따른 네트워크 부하에도 원활한 동기화를 수행할 수 있음을 보였다.

그러나 본 논문에서 제안하는 지연시간은 게임 유저의 네트워크 상황과 게임 유저의 수 등의 많은 변수에 따라 다른 값을 가질 수 있다. 따라서 향후 연구에서는 온라인상의 게임서버에 접속하는 세션에 대한 정확한 예측을 위한 분석모델을 설계하고 연구에서는 다양한 변수들을 고려하여 수학적 접근에 의한 모델링을 통해 온라인 게임을 위한 최적의 지연시간을 제안함으로써 더욱 효율적이고 신뢰성 있는 온라인 게임 서버 엔진에 대해 연구하고자 한다.

참 고 문 헌

[1] Steve Ravin et. al., 류광 역, “AI Game Programming Wisdom2”, 정보문화사, 3월 15일 2005년

[2] 최설호, “배틀넷 개발을 위한 Network Game Server Programming”, 영진닷컴, 11월 20일, 200년

[3] Nicholas Bonneau, Merouane Debbah, Eitan Altman, and Are Hojrungnes, “Non-Atomic Games for Multi-User System”, IEEE Journal on Selected Areas in Communications, Vol. 26, No.7, September 2008

[4] 김혜영, 함대현, “온라인 게임서버를 위한 객체폴링기법에 관한 연구”, 한국게임학회논문지, 제 9권 제6호, 2009년 12월

[5] 이헌주, “게임 인공지능 기술”, 전자통신동향분석 제20권 제 4호 2005년 8월

[6] 조병헌, 박창준, “게임 인공지능 연구동향”, 전자통신동향분석 제23권 제4호 2008년 8월

[7] 김혜영, 임영종, “실시간 전략 시뮬레이션 게임에서의 효율적인 동기화 기법”, 한국게임학회논문지, 제 10권 제 3호, 2010년 6월

[8] Web Site: http://www.aistudy.co.kr/math/finite_state_machine.htm

[9] Eric, Burton, et. Al, "An Efficient Systronization Mechanism for Mirrored Game Architecture," ACM NetGames 2002, Pp.67-73

※ "이 논문은 2009학년도 홍익대학교 학술진흥비에 의하여 지원되었음" 또한 "이 논문은 2007년 정부의 (교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임" (KRF-2007-531-D00011)

저자 소개

김 혜 영(정회원)

- 2005년 2월 : 고려대학교 컴퓨터학과 이학박사
- 2005년 3월 ~ 2006년 8월 : Wright State Uni. Post-Doc.
- 2007년 3월 ~ 현재 : 홍익대학교 게임학부 조교수

<주관심분야 : 모바일 통신, 이동통신망, 모바일 게임, 온라인 게임서버, 게임엔진>