멀티플레이 실시간 전략 시뮬레이션 게임을 위한 동기화 알고리즘들의 성능 평가

강민석*, 김경식**, 오삼권***
*호서대학교 대학원 컴퓨터공학과
**호서대학교 게임공학과
***호서대학교 컴퓨터공학과
e-mail : jusin26@naver.com

Performance Evaluation of Synchronization Algorithms for Multi-play Real-Time Strategy Simulation Games

Min Seok Kang*, Kyung Sik Kim**, Sam Kweon Oh***
*Dept. of Computer Engineering, Graduate School of Hoseo University
**Dept. of Game Engineering, Hoseo University
***Dept. of Computer Engineering, Hoseo University

요 약

MOG(Multiplayer Online Game)의 네트워크 성능은 네트워크 부하량과 사용자의 입력에 대한 반응속도로 측정 가능하다. 본 논문은 MOG의 일종인 실시간 전략 시뮬레이션 분야에서 게임 동기화에 이용되는 프레임 잠금(frame lock) 알고리즘과 게임 턴(game turn) 알고리즘을 소개하고 그 성능을 평가한 결과를 제시한다. 또한 동기화 알고리즘들을 쉽게 교체하여 효율적인 평가를 진행할 수 있는 MOG 서버 구조도 제시한다.

Abstract

The network performance of MOGs(Multiplayer Online Games) can be measured by the amount of network loads and the response times on user inputs. This paper introduces a frame locking algorithm and a game turn algorithm that have been used for game synchronization in the area of RTS(Real-time Strategy Simulation) Games, a kind of MOG; the results of performance evaluation of these two algorithms are also given. In addition, a server architecture for MOG servers in which replacing synchronization algorithms can be done easily for pursuing efficient performance evaluation, is also introduced.

1. 서론

인터넷의 급속한 보급은 인터넷을 통해 여러 사람이 함께 게임을 진행하는 멀티플레이 온라인 게임(MOG: Multi-play Online Game) 산업에 지대한 영향을 주었다[1]. 이런 유형의 게임으로서, 월드(world)라고 불리는 한 게임 공간에서 수백 명의 참여자가 동시에 게임을 진행하는 리니지나 뮤 같은 MMOG(Massively Multiplay Online Game)들이 있다[2]. 이들은 주로 클라이언트-서버 방식으로 구현되며 동기화를 위해 데드 레코닝(dead reckoning)[3]같은 기법을 사용한다. 완벽한 동기화가 어렵고 또한 절실하게 요구되지도 않으므로 정교한 동기화보다는 더 많은 사람들이 동시에 동일 월드 내에서 게임을 진행하도록 하는데 더 중점을 둔다. 이와는 달리, 10 여명 이하의 참여자가 동일 월드 하에서 게임을 진행하는, MOG로 불리는 유형의 게임들이 있다. 이들은 게임 특성에 따라 피어-대-피어(peer-to-peer)나 클라이언트-서버 방식의 형태로 구현되는데, MMOG들과는 달리, 정교한 동기화 모델을 필요로 하는 경우가 많다.

실시간 전략 시뮬레이션(RTS: Real-time Strategy Simulation) 게임은 MOG의 일종으로서 실제 상황처럼 게임을 꾸며 전략성에 대한 가상 체험을 할 수 있는 게임이다[4]. 전쟁의 전략 및 전술을 시뮬레이션 한 스타크래프트 같은 게임이 그 대표적인 예이다[5].

본 논문은 이런 RTS 게임의 네트워크를 통한 멀티플레이의 동기화를 위한 알고리즘들을 소개하고 평가한다. 이를 위해 알고리즘들의 평가를 위한 서버 구조를 제안한다. 제안된 서버 구조는 차후 RTS 게임들을 포함하는 MOG들로 확대 적용할 수 있다.

본 논문의 구성은 다음과 같다. 2 장에서는 멀티플레이를 위한 동기화 알고리즘인 프레임 잠금(frame lock) 알고리즘과 게임 턴(game turn) 알고리즘을 살펴본다. 3 장에서는 이런 동기화 알고리즘을 교체하여 적용할 수 있는 MOG들에 적용 가능한 한 서버 구조를 제안한다. 4 장에서는 이 서버에 프레임 잠금 알고리즘과 게임 턴 알고리

즘을 각각 구현하여 적용한 후 성능을 평가한다. 마지막으로, 5 장에서 향후 연구 과제를 제시하며 결론을 맺는다.

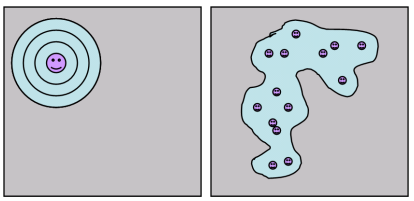
2. 관련 연구

2.1 MMORPG와 RTS 게임의 멀티플레이 비교

MMORPG(Massively Multiplayer Online Role Playing Game)는 MMOG의 대표적인 형태의 게임이다. 이 MMORPG와 MMOG의 일종인 RTS 게임은 다음과 같은 여러 면에서 차이가 있다:

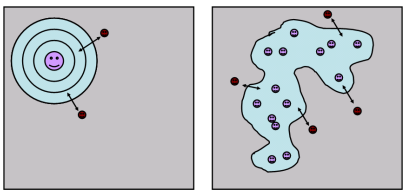
- 관심영역(AOI: area of interest)
- QoS(Quality of Service)와 전송 대상의 수
- 자율성

(그림 1)에서 보는 바와 같이, MMOPRG와 MOG는 한 클라이언트가 주위 환경을 공유하기 위해 가지는 관심영역에 있어 큰 차이를 보인다. 일반적으로 MMORPG의 경우, 클라이언트는 자신을 나타내는 캐릭터 하나로 표현되며 이 캐릭터의 주위를 자신의 AOI로 삼는다. 그러나 RTS의 경우는 클라이언트가 다수의 캐릭터를 가지며 각 캐릭터 주위의 AOI를 모두 합친 영역이 클라이언트의 AOI가 된다. 따라서 MMORPG에 비해 RTS의 AOI가 더 복잡하고 추론하기 어렵다.



(그림 1) MMORPG와 RTS의 AOI

QoS와 전송 대상의 수 또한 많은 차이를 보인다. (그림 2)에서 보는 것처럼, MMORPG의 경우에는 캐릭터를 중심으로 카메라가 위치하기 때문에 캐릭터와의 거리에 따라 QoS를 결정하거나 카메라의 방향에 따라 QoS를 조절할 수 있다. 그러나 RTS 의 경우, 게임 참여자가 자유롭게 카메라의 위치를 옮길 수 있어 QoS를 조절하기 어렵다.



(그림 2) MMORPG 와 RTS 의 유닛(Unit) 움직임

전송 대상의 수 또한 차이가 있다. MMORPG의 경우에는 1 클라이언트가 1 캐릭터를 가지고 이동하고 등장하는 NPC(Non-Playable Character)의 경우도 수가 적어 전체 전송 대상 수가 최대 100 개를 넘지 않는 경우가 대부분이다. 그러나 RTS의 경우에는 1 클라이언트가 약 100 개의 캐릭터를 가지고 움직이고 AOI가 넓기 때문에 10 명의 클라이언트가 함께 게임을 진행할 경우 최대 1000 개의 전송 대상을 가지게 된다.

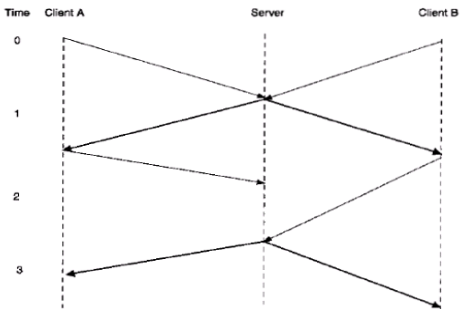
마지막으로, MMORPG의 캐릭터나 단순한 몬스터와는 다르게, RTS의 유닛들은 AI를 가지고 움직인다. 게임 참여자는 단지 유닛들의 큰 움직임을 제어할 뿐이다. 따라서 참여자가 내리는 명령들의 집합만을 제어한다면 게임 내의 수백 개의 캐릭터들이 자율적으로 움직이게 된다. 따라서 기존에 연구된 MMOG의 네트워크 설계를 적용하는 것보다는 근본적으로 다른 구조를 고안하여 적용하는 것이 효과적이다.

2.2 네트워크를 통한 멀티플레이를 위한 동기화

앞서 언급한 AOI 문제로 인해, 대다수의 RTS 멀티플레이는 정확한 동기화에 집중하게 된다. 주로 임의의 턴(또는 프레임)을 생성하고 그 턴을 동시에 진행하도록 하는 알고리즘으로 동기화를 진행한다[6].

2.2.1 프레임 잠금 알고리즘

게임 시에 각 클라이언트들이 모두 이벤트를 보낼 때까지 서버는 대기하고 모든 클라이언트들은 서버로부터 이벤트를 받을 때까지 다시 대기하는, 즉 그 진행이 잠금을 당해 정지하는, 방식을 의미한다. 서버로 보낼 이벤트가 없을 경우에도, 클라이언트는 서버의 대기 상태를 풀어주기 위해 특별한 이벤트가 없음을 알리는 패킷을 보내줘야 한다.



(그림 3) 프레임 잠금

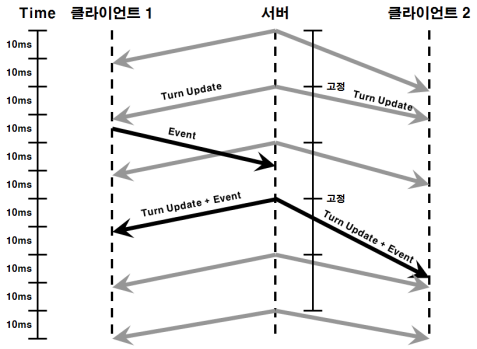
(그림 3)은 프레임 잠금[7]의 한 예이다. time 2 – 3의 과정에서 Client B의 전송이 늦어 Client A 또한 느린 속도로 반응하게 되는 것을 볼 수 있다.

2.2.2 게임 턴 알고리즘

이 알고리즘은 프레임 잠금 알고리즘에서 서버의 대기 부분을 제외한 것으로 볼 수 있다. 서버는 모든 클라이언트가 이벤트를 보낼 때까지 대기하지 않고 일정 시간동안

만 대기하며 그 기간 동안에 모인 이벤트만을 현재 턴에서 실행할 이벤트로 취급한다. 즉 클라이언트가 보내는 “이벤트 없음” 패킷을 받지 않는 대신 일정 시간 동안 대기함으로써 서버의 대기 상태를 효과적으로 푸는 방식이다[8].

(그림 4)는 게임 턴 알고리즘의 예이다. 클라이언트의 이벤트와 관계없이 서버에서 일정시간이 경과하면 이벤트를 전달한다. 클라이언트 2의 경우 네트워크 속도에 따라 이벤트를 받는 시간이 틀린 것을 볼 수 있다.



(그림 4) 게임 턴

3. 동기화 알고리즘의 성능 평가를 위한 서버 구조

정확한 성능 평가를 위해서는 프레임 잠금이나 게임 턴 같은 동기화 알고리즘과는 독립적으로 이들 알고리즘들을 쉽게 교체하여 적용해 볼 수 있는 서버를 구축해야 한다. 이를 위해, 서버에서 게임 내용과 관련된 부분들을 분리 구현하였다.

3.1 서버 구조

3.1.1 IOCP

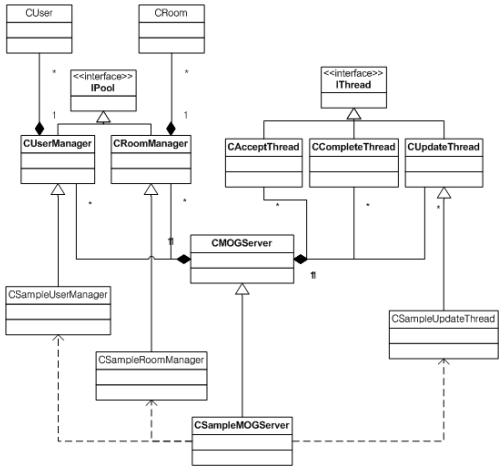
Accept, Send/Recv, Disconnect 등을 담당하는 네트워크 계층이다. (그림 5)의 CMOGServer 클래스 내부에 포함되어 있으며 게임 내용이나 알고리즘과 독립적으로 구현되어 있다.

3.1.2 User Manager, Room Manager

접속 중인 유저들을 관리한다. 같은 세계를 공유하는 유저들의 집합을 룸(room)이라 하고 서버에 존재하는 여러 개의 룸들을 관리한다. (그림 5)와 같이 서버 구현 시에 게임 내용에 맞춰 오버라이딩(overriding)할 수 있다.

3.1.3 Update Thread

동기화 알고리즘 구현을 위해 별도로 분리되어 있는 계층이며 정해진 시간 단위로 특정 알고리즘을 실행한다. (그림 5)에서 보는 바와 같이 게임 내용에 따라 오버라이딩하여 사용할 수 있다. 게임에 따라 사용하지 않을 수도 있다.

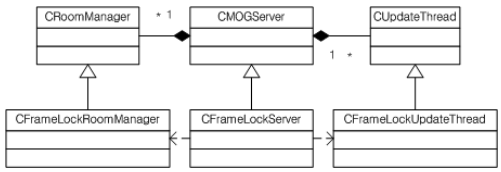


(그림 5) Server Class Diagram

(그림 5)는 알고리즘에 따라 변화 가능한 부분들만 모은 서버의 클래스 다이어그램이다. CMOGServer를 상속받아 서버 클래스를 만든 후 필요에 따라 오버라이딩하여 구현한 클래스들을 적용시키면 된다.

3.2 구현

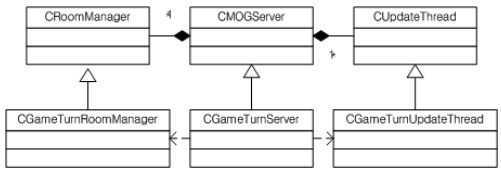
3.2.1 프레임 잠금(Frame Lock) 알고리즘



(그림 6) Server Class Diagram

(그림 6)에서 보는 바와 같이 프레임 잠금 알고리즘을 구현하기 위해서는 RoomManager와 UpdateThread를 오버라이딩해야 한다. RoomManager는 알고리즘에 맞춰 룸의 잠금과 열림을 관리한다. UpdateThread가 모든 클라이언트의 이벤트를 받기 전까지는 턴 갱신 이벤트를 클라이언트로 보내지 않는 방식으로 구현되어 있다. 모든 클라이언트로부터 이벤트를 받은 후에 저장해둔 이벤트들과 턴 갱신 이벤트를 클라이언트로 보낸다.

3.2.2 게임 턴(Game Turn) 알고리즘



(그림 7) Server Class Diagram

게임 턴 알고리즘의 구현은 (그림 7)과 같다. 프레임 잠금과 마찬가지로 RoomManager와 UpdateThread를 구현해야 한다. RoomManager는 프레임 잠금과 같은 방식이고 UpdateThread는 일정시간이 경과하면 클라이언트의 반응과 관계없이 모든 클라이언트로 저장해둔 이벤트들과 턴 갱신 이벤트를 보내는 방식으로 구현되어 있다.

4. 성능 평가 분석

4.1 RTS 게임의 네트워크 효율성 측정을 위한 항목

- 시간당 전송량 (Byte Per Hour, BPH)
- 시간당 잠금 시간(Locking Time Per Hour, LTPH)
- 시간당 잠금 회수(Locking Count Per Hour, LCPH)
- 평균 응답 시간(Average Response Time, ART)

4.2 실험 환경

- 약 1 시간 동안의 테스트 결과로 평가한다. 준비된 이벤트 수를 모두 종료한 시점이 정확하게 실험이 종료된 시점이다.
- 1 시간 동안 클라이언트의 이벤트는 1200 번 (평균 3초당 1번, 20 바이트 발생하는 것으로 가정한다.
- 프레임 잠금과 게임 턴의 경우 1 시간 동안 36000 번의 갱신(갱신 이벤트 : 10 바이트)을 준다. (갱신 주기 100ms, 1초당 10번)
- 같이 게임을 한 클라이언트의 수는 10 개로 정한다.
- 네트워크 전송 시 네트워크 랙을 가정하여 클라이언트의 송신 시 1000ms 씩 10 번의 랙(lag)을 발생시키고 서버의 송신 시 각 클라이언트마다 1000ms 씩 10 번의 랙을 발생시킨다.

4.3 평가

<표 1> 시험 결과

	프레임 잠금	게임 턴
시간당 전송량 (1 서버 -> 1 클라이언트)	372,000 바이트	372,000 바이트
시간당 전송량 (1 클라이언트 -> 1 서버)	372,000 바이트	24,000 바이트
시간당 잠금 시간 (1 클라이언트)	110,000 ms/h	20,000 ms/h
시간당 잠금 회수 (1 클라이언트)	110 회	20 회
평균 응답 시간 (1 클라이언트)	183.33 ms	116.66 ms

시간당 전송량을 통해 필요한 네트워크의 대역폭을 측정할 수 있고 시간당 잠금 시간을 통해 네트워크 지연에 영향 정도를 알 수 있다. 응답 시간 항목으로 구현된 시스템의 반응성을 유추할 수 있다.

일반적인 경우 프레임 잠금 방식보다는 게임 턴 방식이 클라이언트에 미치는 부하가 작음을 알 수 있다. 반응성의 경우에도 비슷하거나 더 나은 것을 알 수 있다.

5. 결론

발전하고 있는 MOG 분야의 대표적인 장르인 RTS 게임에서 사용되는 알고리즘들의 성능을 평가했다. 프레임 잠금 알고리즘과 게임 턴 알고리즘을 구현하여 평가했으며 실험결과를 보면 게임 턴 방식의 알고리즘이 프레임 잠금 방식의 알고리즘에 비해 더 효율적인 것을 알 수 있다.

실험을 위해 동기화 알고리즘을 교체 구현하기 쉬운 서버 구조를 제안했다. 새로운 동기화 알고리즘들이 고안될 경우, 이 서버 구조를 활용하여 손쉽게 비교 평가를 할 수 있다. 이 서버의 구조는 RTS뿐만 아니라 일반적인 MOG들에도 활용 가능할 것으로 판단한다.

5.2 향후 연구

MOG들의 확장을 위해 동시에 같은 세계를 공유할 경우 얼마나 많은 참여자들이 적정 반응성을 유지하며 게임을 진행할 수 있을지에 대한 연구가 필요하다.

또한 네트워크 부하를 줄이기 위해 효율성이 뛰어나다고 알려진 이벤트 잠금(event lock) 기법[7]을 구현하고 이에 대한 성능 평가 후 기존의 결과와 비교해보는 연구를 진행할 예정이다.

참고문헌

[1] 게임백서 2007, 한국게임산업진흥원, 2008
[2] Abdennour El Rhalibi, Agents-based modeling for a peer-to-peer MMOG architecture, Computers in Entertainment (CIE), 2005
[3] Lothar Pantel, On the suitability of dead reckoning schemes for games, Proceedings of the 1st workshop on Network and system support for games, 2002
[4] 염태선 역, 온라인 게임 기획 & 인터랙티브., p62, 제우미디어, 2003
[5] Wikipedia, StarCraft, Wikimedia Foundation, Inc., 2008
[6] Bettner, Paul and Mark Terrano, "GDC 2001 : 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond", Game Developer Conference, 2001
[7] Jim Greer 외 공저, Game Programming Gems 3, pp.575-582, 정보문화사, 2003.
[8] Jan Svarovsky 외 공저, Game Programming Gems 3, pp.583-592, 정보문화사, 2003.