머드형 게임의 구조 및 동기화 방법

°안양재, 윤수미, 김상철

한국외국어대학교 정보산업공과대 컴퓨터공학과
전화 : 011-9625-6065

A Method for Reducing Delay in Networked Multi-User Games

°Yang-Jea Ahn, Sumi Yun, Sang-Chul Kim

Dept of Computer Science & Engineering
Han-Kook University of Foreign Studies
E-mail : doggy_dogg_@hotmail.com

Abstract

The multi-user online game is a typical example of networked graphic applications. Increasing the reality of such a game requires the minimization of problems due to the network delay. In this paper, we propose a game architecture that reduces the network delay needed for message transfer, and a method for synchronization of game states in clients. The proposed game architecture is region proxy-based, and it can require a less network delay than a conventional client-server style that is usually used in commercial games. In our synchronization method, messages are processed in a batch-mode style and the number of rollbacks needed for synchronization significantly decreases. Our experiment shows that our method provides better performance than previous TSS (Trailing State Synchronization)..

I. 서론

최근 다수 사용자가 인터넷상에서 즐기는 게임이 보편화되어 가고 있다. 특히, 온라인 다수 사용자 (multi-user)용 게임의 국내 기술 수준은 세계 정상급이라 할 수 있는데, 이들 게임에서의 사용자 사실감을 높이기 위해서는 실시간 상호작용 (real-time interaction)의 지원이 무엇보다 중요하다. 특히 슈팅 게임, 전쟁 시뮬레이션 게임 및 격투 게임등과 같이 빠른 반응을 필요로 게임에 있어서 실시간 상호작용의 지원은 필수적이라고 할 수 있다 [2].

본 논문에서는 실시간 상호작용을 위해서 지원하기 위해서 필요한 두 가지 요소 기술을 제안한다: 영역 프락시 (proxy) 기반의 게임 구조, 배치 처리 방식의 TSS (Trailing State Synchronization).

온라인 다자간 게임의 각 사용자는 게임 속 자신의 현 위치를 중심으로 일정한 공간 영역의 게임 상태만이 게임 진행에 필요하다. 본 논문에서는 일반적인 구조인 전형적인 서버-클라이언트 구조 [1,3]를 개선해서 게임 속의 가상 세계를 여러 영역으로 나누고 각 프락시는 한 영역을 담당하게 함으로써 프락시의 부하와 네트워크 지연을 줄이는 게임 구조를 제안한다.

하나의 서버 (또는 프락시)와 통신하는 모든 클라이언트들이 일치되는 게임 상태를 공유하도록 하는 작업을 동기화라고 한다. 이제까지의 연구 결과[1, 4, 5] 중에서 TSS(Trailing State Synchronization) 를 제외한 기존 방식들은 주로 병렬 시뮬레이션이나 대형 군사 시뮬레이션을 위해서 고안된 것들이다. 반면에 TSS 는 분산형 게임인 Quake [6]과 같은 게임을 위해서 제안된 동기화 방식이다. 이 방식은 다른 방법들의 단점들인 많은 추가적 지연이나 느슨한 동기화 (loosely consistent synchronization) [1]

문제를 해결하지만, 타임 와프 방식과 같은 낙관적 알고리즘이 가지고 있는 공통적인 문제점인 많은 롤백(rollback)이 여전히 존재한다.

본 논문에서는 TSS 의 기본 원리를 바탕으로 하지만, 기존 TSS 와는 달리 수정 메시지를 즉각 처리하지 않고 일정한 시간 동안 버퍼링한 후에 일괄적으로 처리하는 동기화 방식을 제안한다.

II. 영역 프락시 기반의 게임 구조

우리는 그림 1 과 같이 다수 프락시를 가진 클라이언트 서버 방식의 게임 구조를 제안한다. 각 클라이언트는 사용자의 명령어를 받아서 프락시에게 전달하고 자신도 게임 로직을 수행한다. 그 명령어를 전달 받은 프락시도 게임 로직을 수행하고 발생되는 수정 메시지를 관련되는 클라이언트들에게 전달한다. 결국 클라이언트 간은 프락시를 통해서 수정 및 제어 메시지를 서로 주고 받고, 프락시는 클라이언트들간 동기화에 관한 권한을 가진다. 중앙 서버는 사용자 관리, 프락시의 위치 및 상태를 관리하고 전체 게임 상태를 유지한다.

그림 1 의 구조가 기존 프락시 기반 구조와의 근본적인 차이점은, 우리의 프락시는 게임 속 가상 세계의 한 영역을 담당하는 것이다. 프락시는 한 영역의 게임 상태를 유지하고 이들 영역에서 활동하는 사용자들간 서로 수정메시지를 전달하는 역할을 담당한다. 만약 어느 영역에 존재하는 클라이언트 수가 너무 많아서 해당 프락시에 과중한 부하가 발생하면, 그런 영역에 2 개 이상의 미러형 프락시 (mirrored proxy)을 할당하고 클라이언트를 분산하게 된다. 중앙 서버와 각 프락시 사이와 마찬가지로 같은 영역을 담당하는 프락시들은 서로 연결되어 있다. 서로 다른 영역을 담당하는 프락시들 간은 이론적으로 서로 연결될 필요가 없지만, 게임의 로직상 서로 다른 영역에 있는 클라이언트간에 제어나 수정메시지를 전달한 경우가 빈번하다면 서로 연결하여야 할 것이다. 그림 1에서, 프락시1과 프락시2는 영역 A를 담당하고, 프락시3은 영역 B를 담당하고 있다.

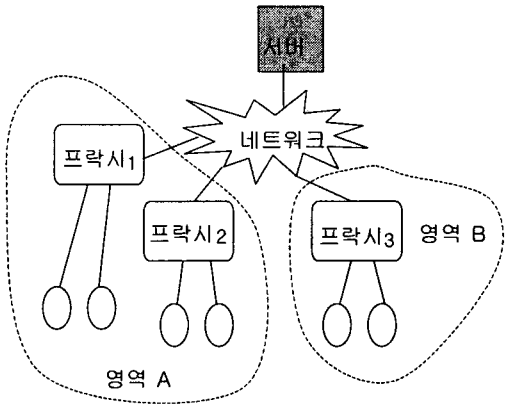

그림 1. 영역 프락시 기반 구조

기존 프락시 기반 게임 구조에서, 각 클라이언트는 어떤 프락시로도 (또는 자신과 가까운 프락시) 연결이 가능하다. 이런 점은 그림 1 의 구조에 비해서 다음과 같은 문제점을 갖는다.

- 기존 구조에서의 각 프락시는 전체 게임 상태를 유지해야 되므로 중앙 서버만큼의 메모리가 필요하지만, 그림 1 의 구조에서는 한 영역의 게임 상태만을 유지하기 위한 메모리면 충분하다.
- 기존 구조에서의 각 프락시는 수정 메시지를 받으면 자신에 연결된 클라이언트들 중에서 관련 있는 클라이언트들을 선택 (또는 필터링)하고 그들에게만 수정 메시지를 전달하기 때문에, 필터링에 소요되는 시간 지연이 필요하지만, 그림 1 의 구조에서는 기본적으로 그런 필터링 작업이 없이 모든 클라이언트들에게 전달하면 된다.
- 기존 구조에서는 수정 메시지를 전달하기 위해서 프락시들간의 메시지 전달이 필요하지만, 그림 1 의 구조에서는 각 프락시(물론, 한 영역을 2 개 이상의 프락시가 담당하는 경우를 제외하고)는 자신에 연결된 클라이언트에만 수정메시지를 전달하면 된다. 이와 같이, 기존 프락시 구조에 나타나는 프락시간의 메시지 전달에 따른 지연을 피할 수 있다.

III. 배치처리 방식의 TSS

3.1 기존 동기화 방식

일치된 게임 상태를 유지하기 위한 동기화는 대단히 중요한데, 기존 동기화 방법들은 크게 보수적인 (conservative) 방법과 낙관적 (optimistic) 방법으로 나누어 진다. 보수적인 방법은 진행 속도가 빠른 게임에는 적합하지 않다 [7]. 낙관적인 방법은 타임 와프 [5]이거나 이를 변형한 것들로서, 기본적으로 전체 상태의 스냅샷(snapshot)을 이용함으로 오버헤드가 크다.

그런 이유에서 [1]는 스냅샵을 이용하지 않고 게임 상태를 여러 카피 (copy)로 유지하는 방법을 제안해지만 롤백이 자주 발생할 수 있는 단점이 있다. 예를 들어, 어떤 메시지가 늦게 도착하면 먼저 어떤 후발 상태로 롤백이 일어 나고 다음에 그 메시지까지가 다시 실행된다. 그런 후, 만약 앞의 메시지보다 조금 앞선 다른 메시지가 도착하면 또 다시 롤백이 일어나게 된다. 버스트 (burst) 형태의 메시지 트래픽이 자주 발생하는 점을 감안하면 연속적인 롤백에 따른 오버헤드는 가벼운 문제가 아니다.

3.2 배치처리 방식의 TSS

시뮬레이션 타임은 일정한 크기의 여러 구간으로 나누어 진다. 각 상태는 한 구간 또는 여러 구간 사이의 메시지들을 배치 방식으로 처리한다. 여러 상태들은 전통적인 TSS 와 같이 서로 일정한 지연을 갖는다. 그림 2 은 세 개의 상태로 구성된 실행 예제이다. 현재 S_0 는 T_2 시점에, S_1 은 T_3 시점에, S_2 는 T_5 시점에 있다. 즉, S_0 는 선발 상태, S_1 은 S_0 보다 한 구간만큼 지연된 후발 상태, S_2 는 S_1 보다 두 구간만큼 지연된 후발 상태이다.

시뮬레이션 사이클이 수행되는 시점을 시뮬레이션 시점이라 부르는데, 시뮬레이션 시점은 구간의 끝이다. 시뮬레이션 시점 T_I에 대해서, $p(T_I, N)$는 T_I 보다 N 개 구간만큼 앞 선 시점을 나타낸다. 예를 들면, 그림 2 에서 $p(T_0,1) = T_1$ 이다. $d(S_I, S_J)$은 상태 S_J 가 상태 S_I보다 몇 구간 만큼 지연된 지를 나타낸다. 예를 들면, $d(S_0,S_1) = 1$ 과 $d(S_1,S_2) = 2$ 이다

.

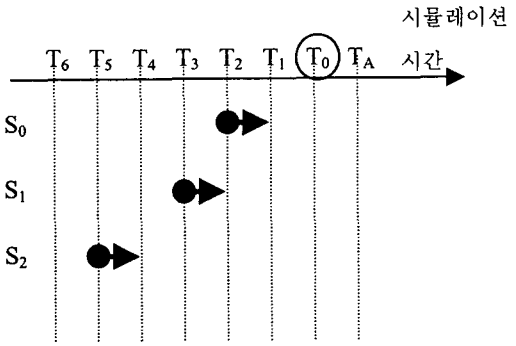

그림 2. 동기화 알고리즘의 실행 예제

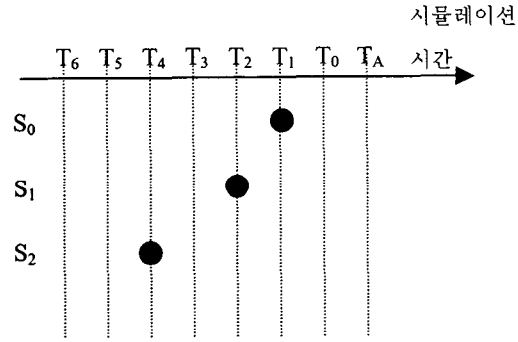

그림 3. 세 상태의 실행이 완료된 상황

현 시뮬레이터 시점은 T_0 이라면 현 시뮬레이션 사이클에서 각 상태는 다음과 같은 일을 한다: 가장 앞선 상태인 S_0 는 T_0 부터 현 시뮬레이션 시점인 T0 바로 앞의 시점인 T_1 까지의 메시지를 실행하고, S_1 은 T_3 부터 T_2 까지의 메시지를 실행하고, S_2 는 T_5 부터 T_3 까지의 메시지를 실행한다. 실행이 성공적으로 완료되면, 그림 3 과 같은 상황이 되며 다음 시뮬레이션 사이클의 시점인 T_A 까지 기다리게 된다. 하지만 T_A 까지 기다리는 동안에도 늦게 도착하는 메시지가 있을 수 있다. 이런 경우에는 그런 메시지의 타임스탬프 바로 뒤의 상태 (편의상 SX 라고 하자)까지는 유효하므로, T_A 에서 실행되는 시뮬레이션 사이클은 다음과 같이 실행된다.

1 단계: S_X 가 S_{X-1}, S_{X-2}, …, S_0 로 각각 복사된다. 즉,

S_X보다 앞선 상태들은 S_X로 롤백된다.

2 단계: S_0는 $p(T_A,1)$ 시점까지 실행되고, S_1은 $p(T_A, 1+d(S_0, S_1))$ 시점까지 실행되고, …, S_X 는 $p(T_A, 1+d(S_0, S_1)+d(S_1,S_2)+ \cdots +d(S_{X-1}, S_X)$ 시점까지 실행된다.

예를 들면, 그림 3 의 상황에서 만약 T_3와 T_2 사이의 타임 스탬프를 갖는 메시지가 도착했다면, S_1 과 S_0 는 더 이상 유효한 상태가 아니게 된다. 다음 시뮬레이션 시점인 T_A에서 다음이 수행된다.

1 단계: S_1과 S_0는 S_2로 롤백된다.

2 단계: S_0 는 T_0 시점까지 실행되고, S_1 은 T_1 까지 실행되고, S_2는 T_3까지 실행된다. 여기서, S_2는 이미 T_3 시점에 있기 때문에 실제적으로 아무런 일도 일어나지 않는다.

3.3 분석

S_0 의 관점에서 보면, 최대 사용자 반응 시간인 T_R은 다음 식으로 구해진다: $T_R = 2T_I + \square$

여기서, TI 는 한 구간의 길이이고, 는 한 구간내의 메시지를 실행하는데 걸리는 최대 시간이다. 그러므로,

$T_R < 150$

$T_I < (150 - \square)/2$

$T_I < 75 - \square/2$

한 시뮬레이션 사이클에서 각 상태는 한 구간내의 메시지를 실행하므로, 그 실행은 다음 시뮬레이션 사이클이 시작하기 전에 끝나야 한다. 그러므로, $T_I > \square$ 이여야 한다.

실험을 통해서, 본 논문에서 제안한 동기화 방법의 성능을 기존 연구와 비교해 본 결과, 10%이상의 롤백 수의 감소를 보였다.

V. 결론

다수 사용자용 인터넷 게임은 주요 멀티미디어 콘텐츠로서 시장의 규모면과 기술적 면에서 빠르게 발전하는 분야이다. 일정하지 않는 네트워크 지연을 갖는 인터넷의 특성상 빠른 사용자 반응을 요구하는 인터넷 게임의 사용자 실감을 높이려면 네트워크 지연에 따른 문제점을 최소화하여야 한다.

본 논문에서는 일반적인 클라이언트 서버 방식을 확장한 영역 프락시 기반 방식을 제안하고, 이 방식이 기존 방식에 비해서 메시지 전달에 소요되는 네트워크 지연을 줄임을 보였다. 우리가 제안한 동기화 방식은 메시지를 배치방식으로 처리하면서 여러 지연을 가지고 진행되는 다수의 게임 상태를 이용해서 클라이언트들간의 동기화를 지원한다. 실험에 따르면, 우리의 동기화 방법은 기존 TSS 방법에 비해서 롤백 수를 줄이는 효과를 보였다.

References

[1]Eric, Burton, et. Al, " An Efficient Synchronization Mechanism for Mirrored Game Architecture," ACM NetGames 2002, p67-73.

[2] Johnathan Blow, " A Look at Latency in Networked Games," Game Developer, July 1998.

[3] Matin Mauve, Stefan Fisher, Jorg Widmer, A Generic Proxy System for Networked Computer Games, ACM NetGames 2002.

[4] L. Gautier, C. Diot, J. Kurose, " End-to-end transmission control mechanisms for multiparty interactive applications on the Internet," IEEE Infocom 1999, Volumne 3, March 1999.

[5] J. S. Steinman, et. Al, " Scalable distributed military simulations using the SPPEDES object-oriented simulation frame," Object-Oriented Simulation Conference, 1998.

[6] id Software Quake, http://www.idsoftware.com

[7] T.A Funkhouser, " RING: A client-server system for multi-user virtual environments," Symposium on Interactive 3D Graphics," ACM SIGGRAPH, April 1995.