Journal of Korea Multimedia Society Vol. 16, No. 4, April 2013(pp. 493-501)
http://dx.doi.org/10.9717/kmms.2013.16.4.493

대규모 다중사용자용 온라인 게임 시스템의 실시간 태스크를 위한우선순위 스케쥴링 기법

김 진 환†

요 약

대규모 다중사용자용 온라인 게임 시스템의 당면 과제는 동시에 많은 사용자들에게 실시간적 응답 시간을 제공하는 것이다. 다수의 사용자들이 요청한 작업은 제한 시간 내에 응답이 제공되어야 하기 때문에 대규모 다중사용자용 온라인 게임 시스템은 일종의 연성 실시간 시스템이다. 클라이언트 이벤트들은 게임 세계의 본질에 따라 상이한 시간 요건과 일관성 요건을 가지고 있다. 이러한 요건들은 CPU 처리시 상이한 우선순위를 갖는 태스크들을 유발하게 된다. 이러한 태스크들의 시간적 요건을 충족하기 위하여 본 논문에서는 일시적 과부하가 발생한 경우 우선순위가 더 높은 태스크들에게 더 많은 CPU 대역폭을 우선적으로 할당하려는 우선순위 스케쥴링 기법들이 제시된다. 제시된 스케쥴링 기법은 우선순위가 높은 태스크일수록 종료시한 내에 성공적으로 종료되는 수를 최대화하며 종료시한이 경과된 이후 종료된 태스크들의 평균 지연시간을 최소화함으로써 대규모 다중사용자 온라인 게임 시스템의 실시간적 성능을 향상시킬 수 있다. 제시된 스케쥴링 기법의 성능은 다양한 시뮬레이션 실험을 통하여 측정되었다.

Priority-based Scheduling Methods for Real-time Tasks of Massively Multiplayer On-line Game Systems

Jinhwan Kim†

ABSTRACT

A key challenge in massively multiplayer on-line game(MMOG) systems is providing real-time response latencies to the large number of concurrent game players. MMOG systems are a kind of soft real-time systems because requests from many players should be responded within specified time constraints. Client events have different timeliness and consistency requirements according to their nature in the game world. These requirements lead to tasks with different priorities on CPU processing. In order to meet their timing constraints, we propose priority scheduling methods that attempt to allocate preferentially more CPU bandwidth to serve an task with the higher priority level in the presence of transient overloading. The proposed scheduling methods are capable of enhancing real-time performance of MMOG system by maximizing the number of tasks with higher priority completed successfully within their deadlines while minimizing total average latency of tasks finished after given deadlines. The performance of these scheduling methods is evaluated through extensive simulation experiments.

Key words: MMOG(대규모 다중사용자 온라인 게임), real-time(실시간). priority(우선순위), latency(지연시간), scheduling(스케쥴링)

※ 교신저자(Corresponding Author) : 김진환, 주소 : 서울 성북구 삼선동2가 389 한성대학교 멀티미디어공학과(136-792), 전화 : 02) 760-4340, FAX : 02) 760-4488, E-mail : kimjh@hansung.ac.kr

접수일 : 2012년 9월 13일, 수정일 : 2012년 11월 15일
완료일 : 2013년 1월 27일
† 정회원, 한성대학교 멀티미디어공학과
※ 본 연구는 한성대학교 교내연구비에 의하여 지원되었음.

1. 서 론

동시에 수백명 이상의 클라이언트들이 참여하는 대규모 다중사용자 온라인 게임(MMOG; Massively Multiplayer On-line Game) 시스템에서 클라이언트와 클라이언트가 제어하는 게임 객체들은 이동성이 존재하므로 일관된 뷰(view)를 제공하는 것이 매우 중요한 요건이다[1]. 즉 임의의 클라이언트의 이벤트가 특정 게임 객체의 상태를 변경시킬 경우 다른 클라이언트들도 이 변경된 사실을 즉시 파악하여야만 게임에 참여하는 모든 클라이언트들이 게임 객체에 대한 정확하며 일관된 뷰를 가질 수 있는 것이다. 그러나 게임 시스템의 구성상 통신망의 지연시간 또는 게임 서버의 CPU 처리 지연시간 등으로 인하여 클라이언트는 게임 객체의 갱신된 상태 정보를 상당한 시간이 경과된 이후에 파악할 수도 있다. 이럴 경우 게임에 참여하는 모든 클라이언트들에게 일관된 뷰를 동시에 보장하는 것은 매우 어려운 문제가 발생하게 된다.

MMOG 시스템은 게임 수행에 필요한 시간적 제약 사항 내에 클라이언트의 이벤트가 전달되어야 하는 실시간적 요건이 필요하며 약속한 서비스 품질을 보장하기 위하여 기본 통신망에 대한 엄격한 요건과 적은 지연 시간을 요구하게 된다[2]. 게임 서버는 모든 클라이언트들의 이벤트들을 수집하고 각각 처리하여 게임 환경을 다시 설정한 후 가상 세계의 갱신된 정보를 모든 클라이언트들에게 전송하게 된다. 따라서 게임 서버 개발 시 CPU 성능을 고려하여 동시에 서비스 가능한 최대 클라이언트들의 수와 게임 수행에 필요한 통신망의 최소 대역폭에 대한 결정이 필요하다[3,4].

온라인 게임 시스템의 클라이언트는 키보드나 마우스를 조작하는 임의의 입력 행위로 이벤트를 발생시킬 수 있으며 이러한 이벤트는 게임 서버에서 대부분 종료시한이 설정된 비주기적인 태스크로 간주될 수 있다[2]. NPC(Non Player Character)를 생성하거나 게임 세계를 갱신하는 일부 이벤트들은 주기적인 태스크로 간주될 수 있다. 게임 서버는 다수의 비주기적 태스크와 주기적인 태스크들을 관리하고 실시간으로 스케쥴링할 수 있는 능력을 구비해야 한다. 본 논문에서는 MMOG 서버의 CPU 처리 지연시간으로 인한 게임 객체의 갱신 상태 정보가 지연되는 현상을 최소화할 수 있도록 서버가 태스크들의 우선순위를 고려하는 실시간 스케쥴링 기법들을 제시한다.

MMOG 시스템에서 클라이언트들의 수가 증가하거나 클라이언트가 발생시키는 이벤트들의 수가 증가할 경우 게임 서버에 일시적인 과부하가 발생할 수 있다. 기존의 EDF(Earliest Deadline First) 실시간 스케쥴링 정책[5]을 사용할 경우 과부하 상태에서는 시스템의 실시간적 성능을 향상시키기 매우 어렵다. 본 논문에서는 MMOG 서버가 과부하 상태에서 종료시한 내에 처리되는 태스크들의 수가 최대화될 수 있도록 우선순위를 고려하는 태스크 스케쥴링 정책이 제시된다. 제시된 스케쥴링 기법은 우선순위가 높은 태스크일 수록 종료시한 내에 성공적으로 종료되는 비율을 증가시키며 또한 종료시한이 경과된 이후 종료된 태스크들의 평균 지연시간을 최소화하여 전체 시스템의 실시간적 성능을 향상시키고자 하였다.

본 논문은 2장에서 대규모 다중사용자용 온라인 게임 시스템에 적용되는 클라이언트 이벤트의 특성이 기술되며 3장에서 우선순위가 고려된 태스크 스케쥴링 알고리즘들이 기술된다. 그리고 4장에서 시뮬레이션을 통하여 각 알고리즘의 성능이 비교 분석되며 5장에서 결론이 기술된다.

2. MMOG 시스템의 클라이언트 이벤트

본 논문에서 MMOG 시스템은 동영상 스트림을 제공하는 VOD(Video On Demand) 시스템 구조처럼 타임스탬프를 사용하는 UDP[6] 통신망 기반의 클라이언트/서버 구조[7, 8]로 구성되는 것을 가정한다. 게임 세계에서 클라이언트가 발생시키는 이벤트의 본질에 따라 이벤트들은 상이한 일관성 요건을 가지게 된다[9]. 실제 클라이언트가 수십명 이상으로 유지되는 다중사용자 온라인 게임인 Quake[10]의 이벤트들은 다섯 가지 유형으로 분류된다(표 1 참조)

사람이 이벤트에 대한 결과를 감지할 수 있는 시간인 100ms보다 큰 이벤트들은 일관성 이벤트로 분류되며 이보다 작은 시간을 갖으면 실시간 이벤트 또는 일관성 실시간 이벤트로 분류된다[9]. 실시간 이벤트는 엄격한 시간 요건을 갖는 반면 다소 느슨한 일관성 요건을 갖게 되며 반대로 일관성 이벤트는 느슨한 시간 요건과 엄격한 일관성 요건을 갖게 된

표 1. 다중사용자 온라인 게임 Quake의 이벤트 유형

이벤트 유형	이벤트 기술
Fire	아바타가 로켓을 발사
Impact	로켓이 충돌후 폭발
Move	새로운 위치로 아바타 이동
Damage/die	아바타가 손상되거나 사망
Spawn	아바타가 다시 생성

다. 엄격한 시간 요건과 일관성 요건을 모두 가지는 일관성 실시간 이벤트는 가장 중요한 이벤트로 간주되며 Quake의 Fire와 Impact 이벤트들이 이에 해당된다. Move 이벤트는 실시간 이벤트이며 Damage/die와 Spawn 이벤트는 일관성 이벤트로 분류된다. 특성이 상이한 이벤트들의 요건들은 게임 서버의 CPU 처리를 위한 스케쥴링 과정에서 상이한 우선순위를 갖는 태스크로 설정될 수 있다. 본 논문에서는 Fire, Impact 등 일관성 실시간 이벤트의 우선순위가 Move 실시간 이벤트보다 더 높은 우선순위가 설정되며 Damage/die, Spawn 이벤트의 우선순위는 Move 이벤트보다 더 낮은 우선순위가 설정된다.

3. 우선순위 스케쥴링 기법

3.1 스케쥴링 가능성

동시에 수백명 이상의 클라이언트들이 참여하는 MMOG 시스템[11]에서 게임 서버는 클라이언트들의 이벤트를 수집하여 처리한 후 일정 주기마다 게임 세계를 갱신하며 이 주기를 T_s라 정의한다. T_s는 실제 클라이언트 화면의 재생율(1 초 동안 화면이 갱신되는 수)과 직접적인 관련이 있다[10]. 예를 들어 재생율이 25인 경우 T_s는 40ms(=1000ms/25)가 된다. 게임 서버는 T_s 동안 처리할 수 있는 평균 태스크의 수 N과 각 태스크의 평균 실행 시간 m을 고려하여 스케쥴링 가능성을 검사한다. 예를 들어 N이 40이고 m이 1 ms인 경우 N×m이 T_s 이하이므로 40개의 태스크들이 주기 내에 처리될 수 있고 이 경우 CPU 활용도는 이론상 최대값인 100%가 된다. 본 논문에서는 태스크 간 CPU switching context에 대한 시간적인 오버헤드는 없는 것으로 가정한다. 그리고 클라이언트가 서버에 요청한 이벤트는 서버에서 독립적인 태스크로 간주되며 각 태스크마다 종료시한이 설정된다.

CPU 활용율이 100% 이하인 상태에서 EDF (Earliest Deadline First) 스케쥴링 정책을[5] 사용할 경우 실시간 태스크들의 종료시한이 거의 대부분 충족될 수 있다. 그러나 게임 서버의 주기 T_s 동안 클라이언트들이 요청한 이벤트 즉 태스크들의 수가 평균 태스크의 수 N을 초과하거나 평균 실행시간 m을 초과하여 게임 서버에 일시적으로 과부하가 발생할 경우 EDF 스케쥴링 정책은 실시간적 성능을 향상시킬 수 없다. 따라서 본 논문에서는 게임 서버의 CPU 활용율이 100%를 초과하는 과부하가 발생할 경우 이를 효과적으로 처리하여 실시간적 성능을 향상시킬 수 있는 우선순위 스케쥴링 기법들을 제시한다.

표 1에서 기술된 일관성 실시간 이벤트들은 이벤트들 중 우선순위가 가장 높지만 게임 서버가 주기 T_s마다 게임 세계를 갱신해야 하는 태스크를 수행해야 하기 때문에 본 논문에서는 Quake 게임 서버가 직접 수행하는 이 태스크에 가장 높은 우선순위를 설정한다. 즉 게임 세계를 주기마다 갱신하도록 서버가 발생시키는 태스크는 우선순위가 가장 높은 그룹 1이며 Fire와 Impact 이벤트들은 두번째로 우선순위가 높은 그룹 2가 된다. 실시간 이벤트인 Move 이벤트는 다음 우선순위를 갖는 그룹 3으로 설정하고 일관성 이벤트인 Damage/die와 Spawn 이벤트는 우선순위가 가장 낮은 그룹 4로 설정한다. 본 논문에서 게임 서버는 한 주기 동안 클라이언트가 유발한 이벤트들을 태스크 단위로 처리한 후 다음 주기에서 게임 세계를 갱신하는 태스크를 가장 높은 우선순위로 실행하는 것을 가정한다.

3.2 우선순위 우선 스케쥴링

게임 서버의 주기 T_s 동안 CPU 대역폭을 우선순위가 높은 그룹의 태스크부터 CPU를 먼저 사용할 수 있게 하는 우선순위 우선(PF; Priority First) 스케쥴링 알고리즘을 제시하며 이는 그림 1에서 기술된다. 그림 1에서 태스크들이 게임 서버의 주기 내에 사용할 수 있는 CPU 대역폭 B_{sum}은 서버의 주기 T_s와 같기 때문에 1행에서 B_{sum}은 T_s로 설정된다. B_{sum}이 0보다 크면(2행) 항상 우선순위가 가장 높은 그룹 1부터(4행) CPU를 사용할 태스크를 선택하게 된다. 이때 해당 그룹 i($0 \le i \le 4$)의 태스크 수는 Length(i)로 표기된다(6행). Length(i)가 1 이상일 경우 EDF (Earliest Deadline First) 원칙에 따라 그룹 i에서 종

```
1: Bsum=Ts;
2: while ( Bsum > 0)
3:  begin
4:    for i=1 to 4 do
5:      begin
6:       if (Length(i) ≥ 1)
7:          Select(Taski);
8:       else
9:          continue;
10:       if ( Exec_time(Taski) ≤ Bsum )
11:        begin   // Exec_time(Taski)동안 CPU 실행
12:            Bsum=Bsum-Exec_time(Taski);
13:            Exec_time(Taski)=0;
14:         end
15:       else
16:         begin    // Bsum동안 CPU 실행
17:          Exec_time(Taski)=Exec_time(Taski)-Bsum;
18:            Bsum=0;
19:         end
20:     end
21: end
```

그림 1. 우선순위 우선 스케쥴링 알고리즘

료시한이 가장 빠른 태스크 $Task_i$가 선택되며 이 과정은 7행에서 Select($Task_i$)로 기술된다. Length(i)가 1 미만일 경우 해당 그룹의 태스크가 없는 상태를 의미하므로 다음 그룹을 선택하는 4행의 과정이 계속 적용되도록 9행의 continue 문이 적용된다. 선택된 $Task_i$의 실제 실행 시간 Exec_time($Task_i$)은 평균 실행 시간 m보다 크거나 작을 수 있다. Exec_time($Task_i$)가 B_{sum}보다 작거나 같을 경우(10행) $Task_i$는 Exec_time($Task_i$)만큼 CPU 대역폭을 이용하여 실행된다(11행의 주석). $Task_i$의 실행이 종료되면 B_{sum}은 Exec_time($Task_i$)만큼 차감되며(12행) Exec_time($Task_i$)은 0이 된다(13행). $Task_i$의 실행이 종료되는 시점에서 종료시한 준수 여부가 확인되며 해당 그룹의 큐에서 $Task_i$가 삭제되어 Length(i)는 1만큼 감소된다. 한편 Exec_time($Task_i$)가 B_{sum}보다 클 경우(15행) $Task_i$는 B_{sum} 시간만큼 CPU를 사용할 수 있으며 이 과정은 16행의 주석에서 CPU_Execute(B_{sum})로 표현된다. $Task_i$는 B_{sum} 동안 CPU를 사용한 이후 Exec_time($Task_i$)는 B_{sum}을 차감한 상태로 다시 설정되며(17행) 다음 주기 이후에 새로 설정된 Exec_time($Task_i$)만큼 CPU를 사용할 수 있게 된다. 0이 된 B_{sum}은(18행) 게임 서버의 다음 주기에 T_s로 다시 설정되며 2행부터 21행까지의 동일한 스케쥴링 과정이 반복 수행된다.

우선순위 우선 스케쥴링 알고리즘에서는 우선순위가 높은 그룹의 태스크들이 항상 CPU 대역폭을 먼저 사용하게 되며 우선순위가 낮을수록 우선순위가 높은 그룹의 태스크들이 주기 내 사용하고 남은 CPU 대역폭을 사용하게 된다.

3.3 우선순위 라운드 로빈 스케쥴링

우선순위 라운드 로빈(PRR; Priority Round Robin) 스케쥴링 알고리즘은 3.2절의 우선순위 우선 스케쥴링 알고리즘과는 달리 우선순위가 낮은 그룹의 태스크들도 CPU를 공평하게 사용할 수 있는 기회가 보장된다. 이 스케쥴링 알고리즘은 그림 2에서 기술된다. 그림 1의 1행처럼 태스크들이 게임 서버의 주기 내에 사용할 수 있는 CPU 대역폭 B_{sum}은 서버의 주기 T_s와 같기 때문에 1행에서 B_{sum}은 T_s로 설정된다. 우선순위가 가장 높은 그룹 1의 태스크부터 CPU를 사용할 수 있도록 그룹 i는 1로 초기화된다(2행). B_{sum}이 0보다 크면(3행) 현재 우선순위 그룹 i부터 CPU를 사용할 태스크를 선택하게 된다. 이때 Length(i)가 1 이상일 경우(5행) EDF(Earliest Deadline First) 원칙에 따라 그룹 i에서 종료시한이 가장 빠른 태스크 $Task_i$가 선택되며 이 과정은 그림 1의 7행에서 기술된 Select($Task_i$)와 동일하다. 선택된 $Task_i$에 대한 7행부터 16행까지의 과정은 그림 1의 10행부터 19행

```
1: Bsum=Ts;
2: i=1;
3: while ( Bsum > 0)
4:  begin
5:    if (Length(i) ≥ 1)
6:         Select(Taski);
7:    if ( Exec_time(Taski) ≤ Bsum )
8:       begin     // Exec_time(Taski)동안 CPU 실행
9:          Bsum=Bsum-Exec_time(Taski);
10:          Exec_time(Taski)=0;
11:       end
12:    else
13:       begin     // Bsum동안 CPU 실행
14:          Exec_time(Taski)=Exec_time(Taski)-Bsum;
15:          Bsum=0;
16:       end
17:    i=i mod 4 + 1;
18: end
```

그림 2. 우선순위 라운드 로빈 스케쥴링 알고리즘

까지의 과정과 동일하게 적용되므로 설명을 생략한다. 우선순위 라운드 로빈 스케쥴링 알고리즘에서는 선택된 $Task_i$가 7행부터 16행까지의 과정이 정상적으로 수행된 이후 그룹 i는 i mod 4 +1 과정이 수행되어(17행) 다음 차례의 우선순위 그룹이 새로 설정된다. 예를 들어 그룹 1의 태스크가 정상적으로 실행되었을 경우 다음 차례인 그룹 2의 태스크가 실행될 수 있는 것이다. 만일 5행에서 Length(i)가 1 미만으로 해당 그룹 i의 태스크가 없었을 경우에는 7행부터 16행까지의 과정은 수행되지 않고 17행의 i mod 4 +1 과정이 수행되어 다음 차례의 우선순위 그룹이 설정된다. 즉 그룹 1의 태스크가 하나도 없었을 경우에는 그룹 2의 태스크가 스케쥴링될 수 있는 것이다. 이와 같이 서버는 그룹 1부터 4까지 해당 그룹의 태스크들이 큐에 존재할 경우 차례대로 CPU를 한번씩 사용할 수 있는 기회를 공평하게 보장하게 된다.

3.4 적응형 우선순위 라운드 로빈 스케쥴링

3.3절의 우선순위 라운드 로빈 스케쥴링 알고리즘에서는 각 우선순위 그룹의 태스크 수와 상관없이 차례대로 CPU를 사용할 수 있는 기회가 제공된다. 그러나 적응형 우선순위 라운드 로빈(APRR; Adaptive Priority Round Robin) 스케쥴링 알고리즘은 우선순위 차례대로 태스크들이 CPU를 사용하긴 하나 우선순위가 높은 그룹의 태스크들이 우선순위가 낮은 그룹의 태스크들보다 더 많을 경우 CPU를 계속 사용할 수 있는 기회가 부여되는 차이점을 가지고 있다. 이 알고리즘은 그림 3에서 기술된다.

그림 2의 1행처럼 CPU 대역폭 B_{sum}은 T_s로 설정되며(1행) 그룹 i는 1로 초기화된다(2행). B_{sum}이 0보다 크고(3행) Length(i)가 1 이상일 경우(5행) 현재 우선순위 그룹 i부터 CPU를 사용할 태스크를 선택하게 된다. 5행부터 16행까지의 과정은 그림 2의 5행부터 16행까지의 과정과 동일하게 수행되므로 설명을 생략한다. 적응형 우선순위 라운드 로빈 스케쥴링 알고리즘에서는 그룹 1부터 3까지 각 그룹의 태스크 수가 우선순위가 낮은 다음 그룹 i+1의 태스크 수보다 많을 경우(17행) 그룹 i는 변경되지 않기 때문에(18행) 다음 스케쥴링 차례에서도 현재 우선순위 그룹 i의 다른 태스크에게 CPU를 사용할 수 있는 기회가 부여된다. 예를 들면 현재 우선순위 그룹 2의 태스크 수가 그룹 3의 태스크 수보다 클 경우 스케쥴링

```
1: Bsum=Ts;
2: i=1;
3: while ( Bsum > 0)
4:  begin
5:    if (Length(i) ≥ 1)
6:       Select(Taski);
7:    if ( Exec_time(Taski) ≤ Bsum )
8:      begin    // Exec_time(Taski)동안 CPU 실행
9:        Bsum=Bsum-Exec_time(Taski);
10:         Exec_time(Taski)=0;
11:     end
12:   else
13:     begin    // Bsum동안 CPU 실행
14:       Exec_time(Taski)=Exec_time(Taski)-Bsum;
15:       Bsum=0;
16:     end
17:   if ((i ≤ 3) and (Length(i)>Length(i+1)))
18:      i=i;
19:   else
20:      i=i mod 4 + 1;
21: end
```

그림 3. 적응형 우선순위 라운드 로빈 스케쥴링 알고리즘

차례는 그룹 2로 계속 유지되는 것이다. 그러나 현재 우선순위 그룹 2의 태스크 수가 그룹 3의 태스크 수보다 크지 않을 경우(19행)에는 그룹 번호가 우선순위 라운드 로빈 스케쥴링 알고리즘과 동일한 방법(그림 2의 17행)인 i=i mod 4 + 1 과정이 수행되어 다음 우선순위인 그룹 3으로 변경된다(20행). 현재 우선순위 그룹이 그룹 4일 경우 그룹 4는 우선순위가 가장 낮은 그룹이므로 다른 우선순위 그룹과 태스크 수를 비교할 필요가 없으므로 19행에 이어 20행의 i=i mod 4 + 1 과정이 수행되고 다음 차례의 우선순위인 그룹 1로 변경된다. 이 스케쥴링 알고리즘에서는 현재 우선순위 그룹의 태스크 수가 우선순위가 더 낮은 다음 그룹의 태스크 수보다 클 경우에는 CPU를 사용할 수 있는 기회가 계속 제공되므로 CPU 사용 기회가 공평하게 적용되는 우선순위 라운드 로빈 스케쥴링 알고리즘에 비하여 우선순위가 높은 그룹의 태스크일수록 종료시한이 더 많이 보장될 수 있다.

4. 성능 분석

4.1 실험 환경과 변수

본 논문에서는 Quake 게임 서버가 게임 세계를 1초에 25회씩 갱신하는 것을 가정하여 재생율은 25

로 설정되며 이에 대한 갱신 주기 T_s는 40ms가 된다. 클라이언트가 발생한 이벤트는 게임 서버에서 태스크로 전환되며 각 태스크의 평균 실행 시간은 1ms이며 최소값 0.5ms와 최대값 1.5ms 분포 범위에서 실제 실행 시간이 설정된다. 서버의 갱신 주기 40ms마다 실행될 수 있는 평균 태스크의 수 N은 각 태스크의 평균 실행시간 1ms를 고려할 때 40이 되며 이때 CPU 활용율은 이론상 최대값인 100%가 된다. 본 논문에서는 CPU 활용율을 100%로 유지하기 위하여 325명의 클라이언트들이 매초 평균 3 개의 이벤트를 발생하는 것으로 가정하였다. 게임 서버는 1초 즉 1000ms마다 325명의 클라이언트들이 발생시키는 975(=325×3)개의 태스크와 게임 세계를 갱신하는 서버 태스크 25개를 합한 1000개의 태스크를 스케쥴링하게 된다. 이는 서버가 주기 40ms마다 평균 실행시간이 1ms인 40개의 태스크들을 처리하는 것을 의미하며 이때 CPU 활용율은 100%가 된다. 본 논문에서는 그룹 1을 제외한 그룹 2, 3, 4의 태스크들은 각각 20%, 60%, 20%의 비율로 구성되는 것을 가정하였다. 즉 클라이언트들에 의해 발생되는 초당 975개의 태스크들 중 그룹 2와 4의 태스크 수가 각각 195개(975개의 20%)이며 그룹 3의 태스크 수가 585개(975개의 60%)임을 의미한다. 각 태스크의 종료시한은 우선순위에 상관없이 100ms(이벤트에 대한 처리 결과를 사람이 감지할 수 있는 크기의 시간)를 고려하여 동일하게 설정되었다.

4.2 종료시한 성공율

CPU 활용율이 100% 이하인 상태에서는 기존의 실시간 스케쥴링 기법인 EDF 정책을 적용할 경우 태스크들의 종료시한 성공율(전체 태스크들 중 종료시한 내에 성공적으로 종료된 태스크들의 백분율)은 100%에 근접한 결과가 실제 실험에서 확인되었으며 이에 관한 내용은 본 논문에서 기술하지 않기로 한다. 본 논문에서는 종료시한 성공율을 성능 평가의 주요 지표로 설정하였다. 게임 서버의 CPU 활용율이 100%인 상태에서 일시적인 과부하가 발생할 경우 기존의 EDF 정책만으로는 실시간적 성능 향상을 기대할 수 없다. 본 논문에서는 클라이언트의 이벤트 평균 발생 시간 간격을 감소시켜 초당 발생되는 이벤트 수를 증가시켰으며 게임 서버가 주기마다 처리해야 하는 태스크의 수를 증가시킨 상황에서 종료시한 성공율을 측정하였다.

그림 4에서 각 클라이언트가 발생하는 평균 이벤트 수는 초당 3 개에서 3.3 개로 10% 증가되었으며 게임 서버의 CPU 활용율이 이미 100%인 상태에서 10% 정도의 과부하가 발생한 5초 동안의 스케쥴링 결과를 1초 간격으로 나타내고 있다. 실험에서 그룹 2와 4의 태스크는 초당 평균 10% 증가된 214.5개이며 그룹 3의 평균 태스크 수는 643.5개이다. 그룹 1의 태스크는 모든 스케쥴링 알고리즘에서 항상 종료시한 내에 종료되기 때문에 그룹 1을 제외한 그룹 2, 3, 4의 태스크들을 대상으로 종료시한 성공율을 비교하고 있다. EDF의 경우 우선순위를 고려하지 않기 때문에 처음 1초 동안은 75.70%의 높은 종료시한 성공율이 나타나지만 시간이 지날수록 종료시한이 경과된 이후 종료되는 태스크들의 수가 급증하기 때문에 종료시한 성공율이 급격히 감소하는 것으로 분석되었다. 5초가 되었을 때 EDF의 종료시한 성공률은 약 15.15%가 나타났다. 그림 5는 각 스케쥴링 알고리즘에서 5초가 되었을때 우선순위 그룹 2, 3, 4에 대한 종료시한 성공률을 비교하고 있다. EDF의 경우 그룹 2, 3, 4의 종료시한 성공률은 각각 14.65%, 14.97%, 16.13%로 큰 차이가 없다.

높은 우선순위 그룹의 태스크들을 항상 먼저 처리하는 우선순위 우선(PF) 알고리즘은 우선순위가 높은 그룹 2와 3의 태스크들이 대부분 종료시한 내에 종료되기 때문에 항상 종료시한 성공율이 78.42%를

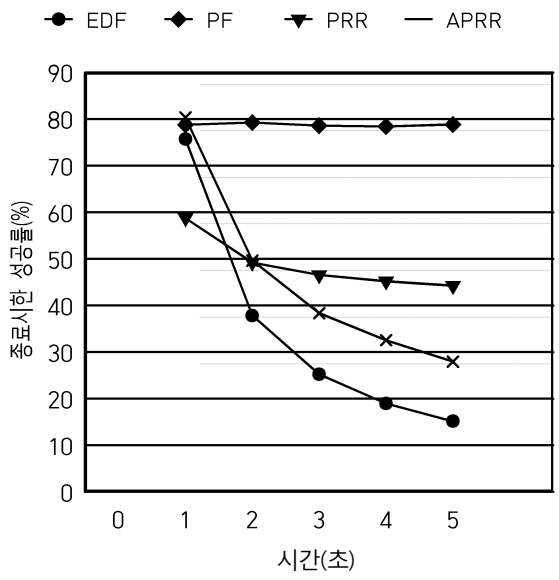

그림 4. 태스크들의 종료시한 성공율

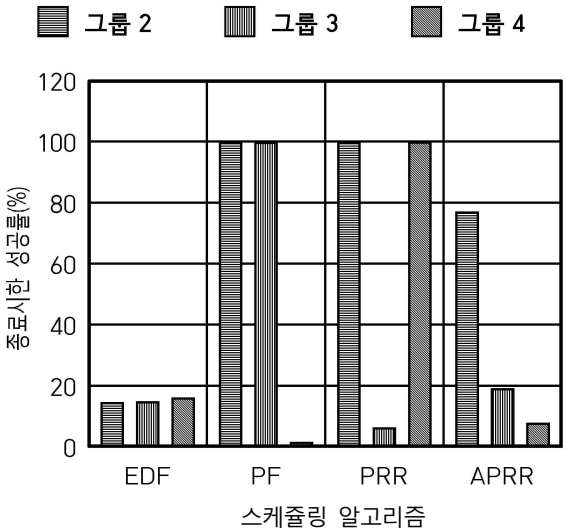

그림 5. 우선순위 그룹별 종료시한 성공율

초과하는 것으로 나타났으며(그림 4 참조) 그룹 2, 3, 4의 종료시한 성공률은 각각 99.90%, 99.94%, 1.66%로 나타났다(그림 5 참조). 우선순위가 높은 그룹 2와 3은 항상 종료시한 내에 종료될 수 있지만 우선순위가 낮은 그룹 4는 시간이 지날수록 종료시한 성공률이 매우 낮아지는 것으로 파악되었다.

우선순위 라운드 로빈(PRR) 알고리즘의 종료시한 성공율은 1초에서 58.75%로 측정되었으며 매초마다 다소 감소되어 5초에서 44.25%로 측정되었다(그림 4 참조). 그룹 2, 3, 4의 종료시한 성공율은 99.90%, 6.36%, 100.0%로 태스크들의 구성 비율이 20%인 그룹 2와 4에 비하여 구성비율이 60%인 그룹 3이 가장 낮은 종료시한 성공율을 나타냈다(그림 5 참조). PRR 알고리즘에서는 각 그룹의 태스크 수에 상관없이 그룹간 공평하게 CPU를 사용할 수 있는 기회가 부여되므로 구성비율이 큰 그룹 3의 종료시한 성공율이 현저히 감소된 것으로 분석되었다. 적응형 우선순위 라운드 로빈(APRR) 알고리즘의 종료시한 성공률은 80.35%에서 점차 감소된 27.86%로 나타났으며(그림 4 참조) 그룹 2, 3, 4의 종료시한 성공률은 각각 77.11%, 19.26%, 7.89%로 측정되었다(그림 5 참조). PRR 알고리즘에 비하여 그룹 2의 종료시한 성공률은 22.79% 감소되었고 그룹 3은 12.9% 증가된 결과가 나타났다. 우선순위가 가장 낮은 그룹 4는 92.11%가 감소된 결과가 나타났다.

4.3 평균 지연시간

표 2는 각 스케줄링 알고리즘에서 종료시한 이후 종료된 태스크들의 평균 지연시간을 1초부터 5초까지 측정한 결과이며 괄호안에 평균 지연시간의 표준편차가 기술되었다. EDF 알고리즘의 경우 6.23ms에서 193.28ms까지 점차 증가된 평균 지연시간이 측정되었으며 PF 알고리즘은 192.35ms에서 1100.61ms까지 급격히 증가된 평균 지연시간이 측정되었다. 5초에서 측정된 각 스케줄링 알고리즘의 그룹별 평균 지연시간과 표준편차(표 3 참조)를 분석하면 EDF 알고리즘의 경우 우선순위에 상관없이 비슷한 평균 지연시간과 표준편차가 측정되었지만 PF 알고리즘의 경우 우선순위가 높은 그룹 2와 3은 평균 지연시간이 0ms인 반면 우선순위가 낮은 그룹 4는 1100.61ms로 나타나 매우 큰 차이를 보이고 있다. PF 알고리즘의 경우 그룹 4의 평균 지연시간과 표준편차가 전체의 평균 지연시간과 표준편차를 의미하게 된다.

PRR 알고리즘은 73.46ms에서 489.67ms의 점차 증가된 평균 지연시간과 표준편차가 나타났고 APRR 알고리즘은 이보다 평균 지연시간의 증가 폭이 다소

표 2. 태스크들의 평균 지연시간 (단위: ms)

시간 / 알고리즘	1초	2초	3초	4초	5초
EDF	6.16 (3.24)	53.25 (31.83)	100.30 (58.65)	146.28 (84.49)	193.28 (111.70)
PF	192.35 (108.68)	439.88 (285.05)	716.56 (407.23)	920.52 (487.40)	1100.61 (600.38)
PRR	73.46 (38.70)	174.94 (89.96)	285.83 (161.52)	395.58 (224.08)	489.67 (268.52)
APRR	94.92 (45.41)	90.05 (111.89)	139.52 (144.35)	193.64 (188.85)	241.64 (233.43)

표 3. 그룹별 평균 지연시간 (단위: ms)

그룹 / 알고리즘	2	3	4
EDF	195.26 (111.63)	192.29 (112.39)	194.28 (109.79)
PF	0 (0)	0 (0)	1100.61 (600.38)
PRR	0 (0)	489.67 (268.52)	0 (0)
APRR	33.54 (34.47)	155.53 (91.31)	541.53 (282.45)

감소된 94.92ms부터 241.64ms까지의 평균 지연시간과 표준편차가 측정되었다. 그룹별 평균 지연시간도 PRR 알고리즘의 그룹 2와 4는 0ms인 반면 그룹 3은 489.67ms로 나타났고 APRR 알고리즘의 그룹 2, 3, 4는 각각 33.54ms, 155.53ms, 541.ms로 우선순위가 낮아짐에 따라 평균 지연시간이 점차 증가된 결과가 나타났다. PRR 알고리즘의 경우 그룹 3의 평균 지연시간과 표준편차가 알고리즘 전체의 평균 지연시간과 표준편차를 의미하고 있다. APRR 알고리즘의 경우 우선순위가 높은 그룹 2의 평균 지연시간에 비해 표준편차가 크게 나타난 것은 해당 그룹의 태스크들의 종료시한 이후 종료되는 시간 즉 지연시간의 변동폭이 크다는 것을 의미한다. 그룹 3의 태스크들 수에 따라 평균 지연시간의 크기가 달라지는 것으로 분석되었다. 상대적으로 우선순위가 낮은 그룹 3과 4의 표준편차는 다른 알고리즘의 결과와 유사한 것으로 분석되었다.

5. 결 론

본 논문에서는 MMOG 게임 서버의 CPU 처리시 일시적 과부하가 발생할 경우 우선순위와 평균 지연시간을 고려하는 실시간 스케쥴링 기법들을 제시하고 5초 동안 측정된 실험 결과를 비교 분석하였다. 기존의 EDF 알고리즘은 우선순위가 전혀 반영되지 않기 때문에 각 그룹의 종료시한 성공률이 5초일때 평균 15% 정도로 거의 유사하게 측정되었고 측정된 그룹간 태스크들의 평균 지연시간도 193ms를 다소 초과하는 정도로 나타났다. PF 알고리즘은 우선순위가 높은 그룹의 태스크들은 거의 100%의 종료시한 성공률이 보장되나 우선순위가 가장 낮은 그룹의 태스크들은 2% 미만의 종료시한 성공률이 측정되었고 평균 지연시간도 1100ms를 초과하는 것으로 나타났다. 이는 MMOG 시스템에서 이벤트의 특성을 반영하는 우선순위를 중요시할 경우 우선순위가 높은 태스크들의 종료시한 성공률이 최대화될 수 있으나 우선순위가 낮은 태스크들의 평균 지연시간이 급격히 증가하는 현상을 의미하게 된다. 우선순위 그룹간 공평한 기회를 보장하는 PRR 알고리즘에서는 그룹 2와 4의 종료시한 성공률이 거의 100%인 반면 그룹 3은 6%를 초과하는 종료시한 성공률과 489ms를 초과하는 평균 지연시간이 측정되었다. PRR 알고리즘에서는 그룹 3보다 우선순위가 낮은 그룹 4의 종료시한 성공률이 더 높게 측정되는 현상이 발생한 것이다. 우선순위 그룹의 태스크 수가 다음 우선순위 그룹의 태스크 수보다 클 경우 CPU 사용 기회를 계속 부여하는 APRR 알고리즘은 그룹 2, 3, 4의 종료시한 성공률이 약 77%, 19%, 8% 정도로 우선순위 순서를 반영하며 평균 지연시간은 약 33ms, 156ms, 541ms로 우선순위가 낮을수록 점차 증가하는 순서로 측정되었다.

수백명 이상의 클라이언트들이 동시에 참여하는 MMOG 시스템에서는 이벤트 특성을 반영하는 우선순위를 직접 고려하면서도 종료시한 성공률의 최대화와 평균 지연시간의 최소화를 도모하여 전체 시스템의 성능을 향상시킬 수 있는 실시간 스케쥴링 기법들이 필요하며 향상시키고자 하는 구체적인 성능 목표에 따라 스케쥴링 기법이 달라질 수 있다. 본 논문에서 제시된 스케쥴링 기법들 중 특히 APRR 알고리즘은 MMOG 시스템에서 우선순위의 중요성과 종료시한 준수 여부에 대한 두가지 실시간적 성능 목표를 모두 고려할 경우 적용될 수 있을 것으로 기대된다. 향후 모바일 환경의 MMOG 시스템의 구체적인 성능 향상 목표에 따라 게임 서버에 적용될 수 있는 더욱 효율적인 실시간 스케쥴링 알고리즘 개발과 이에 관한 실험 결과가 필요하다.

참 고 문 헌

[1] M. Rumney, *LTE and Evolution to 4G Wireless*, Agilent Technologies, 2009.

[2] P. Edara, R.K. Balan, and A. Datta, “Scalable Consistency Protocols for Massively Multi-

player Games," *Singapre Mangement University,* 2007.

[3] Y.W. Ahn, A.M.K. Cheng, J. Baek, and P.S. Fisher, "A Multiplayer Real-Time Game Protocol Architecture for Reducing Network Latency," *IEEE Transactions on Consumer Electronics,* Vol. 55, No. 4, pp. 1883-1889, 2009.

[4] S. Harcsik, A. Petlund, C. Griwods, and P. Halvorsen, "Latency Evaluation of Networking Mechanisms for Game Traffic," *6th Workshop on Networks and System Support for Games,* pp. 129-134, 2007.

[5] C.L. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM,* Vol. 2, No. 4, pp. 46-61, 1973.

[6] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-time Applications," *Internet Engineering Task Force, RFC 1889*, 1996.

[7] 김성용, "온라인 게임 클라이언트 기술 동향," 멀티미디어학회논문지, 제13권, 제1호, pp. 24-33, 2009.

[8] Y.E. Liu, J. Wang, M. Kwok, J. Diamond, and M. Toulouse, "Capability of IEEE 802.11g Networks in Supporting Multi-player Online Games," *Consumer Communications and Networking Conference,* pp. 1193-1198, 2006.

[9] A. Hsu, J. Ling, Q. Li, and C.C. Jay Kuo, "On the design of Multiplayer On-line Video Game Systems," *SPIE ITCOM,* pp. 180-191, 2003.

[10] Doom, Quake, ID Software, Inc. http://www.idsoftware.com, 2012

[11] A. Abdelkhalek, A. Bilas, and A. Moshovos, "Behavior and Performance of Interactive Multi-player Game Servers," *IEEE Int'l Symp. on Performance Analysis of Systems and Software,* 2001.

김 진 환

1982년 3월~1986년 2월 서울대학교 컴퓨터공학과 학사
1986년 3월~1988년 2월 서울대학교 컴퓨터공학과 석사
1988년 3월~1994년 2월 서울대학교 컴퓨터공학과 박사
1994년 3월~1996년 2월 서울대학교 컴퓨터신기술공동연구소 특별연구원
1995년 3월~현재 한성대학교 멀티미디어공학과 교수
관심분야: 멀티미디어 시스템, 분산 실시간 및 게임 시스템