분산 실시간 멀티미디어 데이터베이스 시스템을 위한 신축성있는 스케쥴링 기법

김 진 환†

요 약

본 논문에서는 분산 실시간 멀티미디어 데이터베이스 시스템에서 경성 실시간 태스크들과 멀티미디어 태스크들을 효율적으로 통합할 수 있는 신축적인 스케쥴링 기법이 제시된다. 경성 실시간 태스크가 최악의 경우에 대한 실행 시간을 기반으로 하는 반면 멀티미디어 태스크는 평균 실행 시간을 기반으로 한다. 동일한 시스템에 존재하는 두 가지 태스크들에 대하여 CPU 대역폭을 분할 조정하는 서버 기법이 기술된다. 제시된 기법에서는 한부류의 태스크들의 수와 도착 비율이 변동되는 과부하 문제를 해결하기 위하여 부류별로 CPU 대역폭의 비율이 조정될 수 있다. 경성 실시간 태스크가 서버의 주기내에서 실행될 수 있는 시간이 제한되는 반면 멀티미디어 태스크에 설정된 대역폭은 동적으로 변할 수 있다. 제시된 기법은 경성 실시간 태스크들의 실시간성을 모두 보장하는 한편 멀티미디어 태스크들의 평균 지연 시간을 최소화할 수 있다. 스케쥴링 기법의 성능은 시뮬레이션을 통하여 다른 스케쥴링 기법과 비교 분석된다.

Scalable scheduling techniques for distributed real-time multimedia database systems

Jinhwan Kim†

ABSTRACT

In this paper, we propose scalable scheduling techniques based on EDF to efficiently integrate hard real time and multimedia soft real time tasks in the distributed real-time multimedia database system. Hard tasks are guaranteed based on worst case execution times, whereas multimedia soft tasks are served based on mean execution times. This paper describes a served-based scheme for partitioning the CPU bandwidth among different task classes that coexist in the same system. To handle the problem of class overloads characterized by varying number of tasks and varying task arrival rates, this scheme shows how to adjust the fraction of the CPU bandwidth assigned to each class. This scheme fixes the maximum time that each hard task can execute in the period of the server, whereas it can dynamically change the bandwidth reserved to each multimedia task. The proposed method is capable of minimizing the mean tardiness of multimedia tasks, without jeopardizing the schedulability of the hard tasks. The performance of this scheduling method is compared with that of similar mechanisms through simulation experiments.

키워드 : 경성 실시간(hard real-time), 연성 실시간(soft real-time), 종료시한(deadline), 지연시간(delay), 주기(period)

1. 서 론

최근 공장 자동화와 방위 시스템과 같은 경성 실시간(hard real-time) 시스템 응용 분야에서 음성 및 영상 등의 멀티미디어 정보가 이용되고 있다[1]. 이러한 정보는 경성 실시간적 제어 정보보다는 덜 중요하나 지연 시간과 지터(jitter) 허용 등 연성 실시간적(soft real-time) 특성을 가지게 된다. 멀티미디어 정보와 경성 실시간 제어 정보를 모두 처리해야 하는 시스템의 경우 연성 실시간 태스크들과 경성 실시간 태스크들이 공존하게 되므로 이들을 효율적으로 스케쥴링할 수 있는 기법이 필요하다. 현재 고속 통신망 및 컴퓨터 기술, 대용량 저장 장치, 첨단 센서(sensor) 들의 급속한 발전은 멀티미디어 정보가 직접 이용되는 분산 실시간 제어 시스템의 구축을 가능하게 하고 있다. 멀티미디어 정보와 처리 능력이 제어 환경에 포함되는 이러한 시스템에서는 향후 제어 프로세스의 직관적인 가시화, 프로세스의 안전도 및 품질 향상, 관리 운영 차원에서의 의사 결정 수행 방법 개선 등의 장점이 추구될 수 있다[2].

각종 센서들이 탐지한 영상 정보들을 이용하면 분산 제어 시스템의 전체 상태를 3차원 영상으로도 표현할 수 있으며

※ 이 논문은 한국과학재단의 해외 post doc 연수 지원에 의하여 연구되었음.
† 정 회 원 : 한성대학교 정보전산학부 부교수
논문접수 : 2001년 2월 28일, 심사완료 : 2001년 11월 13일

시스템의 전체 상태도 실시간으로 갱신할 수 있다[2]. 시스템의 상태를 나타내는 화면에 특별한 물체가 출현하거나 비정상적인 상태가 발생하는 경우 관리 및 제어 기능 목적의 프로세스가 즉각 수행되며 이러한 프로세스들은 대부분 경성 실시간 특성을 가지게 된다. 본 논문에서는 다양한 첨단 센서(특히 일반 촬영장치외에 X레이, 초음파, 적외선 장치 등)들로부터 수집된 영상 정보가 고속 통신망을 통하여 전송된 후 전체 시스템의 상태를 화면으로 구성하는 한편 분산 멀티미디어 데이터베이스 시스템에 저장될 수 있도록 하는 실시간 스케쥴링 기법을 제시한다. 분산 실시간 멀티미디어 데이터베이스 시스템에서는 멀티미디어 데이터를 처리하는 연성 실시간 프로세스들과 시스템의 이상 현상을 제한된 시간 내에 해결하고자 하는 경성 실시간 프로세스들이 공존하게 되므로 서로 다른 두 형태의 태스크들을 효율적으로 스케쥴링할 수 있는 기법이 필요하다[3].

일정한 주기를 갖는 경성 실시간 태스크들은 대부분 최악의 실행 시간(worst case execution time)을 기반으로 하는 특성이 있고 기존의 실시간 스케쥴링 기법인 최조종료시한(EDF ; Earliest Deadline First) 기법이나[4, 5] RM(Rate Monotonic) 기법에[4, 6] 기반하여 스케쥴링되고 있다. 그러나 연성 실시간 특성을 갖는 멀티미디어 태스크들은 처리하고자 하는 데이터의 규모에 따라 실행시간이 가변적이기 때문에 최악의 실행시간을 기반으로하는 실시간 스케쥴링 기법을 적용할 경우 CPU 대역폭의 낭비가 심해져서 자원의 활용도가 낮아지는 문제가 발생하게 된다[3]. 경성 실시간 태스크들과는 달리 연성 실시간 태스크들은 종료시한이 경과되더라도 시스템에 미치는 영향이 심각하지 않은 것으로 간주되기 때문에 최악의 실행시간보다는 평균 실행 시간을 기반으로 하는 스케쥴링 기법들이 제시된 바 있다[3, 7, 8]. 최악의 실행 시간이 설정된 경성 실시간 태스크들과 평균 실행 시간이 설정된 멀티미디어 태스크들이 동일 시스템에 존재하는 경우 멀티미디어 태스크마다 일정한 주기와 실행 시간을 갖는 별도의 서버를 유지함으로써 멀티미디어 태스크들이 스케쥴링되는 CBS(Constant Bandwidth Server) 기법[3]에서는 모든 경성 실시간 태스크들의 종료시한이 보장되는 반면 멀티미디어 태스크들의 평균 실행 시간의 변동이 클 경우 종료시한이 경과되는 시간이 증가될 수 있다. 이외에도 멀티미디어 태스크들과 경성 실시간 태스크들을 스케쥴링하기 위하여 다양한 서버 할당 정책과 여러 단계의 스케쥴링 기법이 제시된 논문[1]에서도 멀티미디어 태스크의 평균 실행 시간의 변동에 따른 멀티미디어 태스크의 종료시한 경과 시간에 대한 분석 결과가 제시되지 않았다.

본 논문에서는 경성 실시간 태스크들의 종료시한을 모두 보장하면서 멀티미디어 태스크들의 종료시한 경과 시간이 가능한 최소화되도록 두 종류의 태스크들이 CPU 대역폭을 효율적으로 사용할 수 있는 신축적인 스케쥴링 기법이 제시되었다. 태스크들 중 최소 주기를 서버의 주기로 구성하며 경성 실시간 태스크들과 멀티미디어 태스크들이 서버의 주기 내에서 실행되는 시간을 각각 설정함으로써 경성 실시간 태스크들로 인하여 멀티미디어 태스크들의 실행 시간이 지연되는 경우가 최소화되도록 하였다. 또한 다양한 주기와 실행 시간으로 구성되는 태스크들의 부하가 변동되는 경우 이를 신축성있게 스케쥴링하기 위하여 주기내 설정된 해당 태스크들에 대한 실행 시간이 조정될 수 있다. 제시된 스케쥴링 기법에서는 각 멀티미디어 태스크가 센서로부터 주기적으로 전송되는 다양한 크기의 프레임(동영상 정보의 처리 단위)을 처리하는 지연 시간을 최소화함으로써 시스템의 현재 상태를 실시간으로 반영하며 신속한 제어가 수행될 수 있다. 분산 실시간 멀티미디어 데이터베이스 시스템을 위한 신축성있는 스케쥴링 기법의 알고리즘과 실험 결과가 본 논문에서 기술되었다.

2. 분산 실시간 멀티미디어 데이터베이스 시스템

2.1 시스템 구성 배경

분산 제어 시스템 등의 실시간 응용을 위한 멀티미디어 요건과 실시간 제어 처리 요건들이 통합된 분산 시스템의 구조는[2] (그림 1)과 같이 가정된다. 주요 구성 요소들로는 산업현장에 분산된 지역 처리단위(LC ; Local Cell), 통합 데이터 서버(IDS ; Integrated Data Server), 관리용 감시 및 제어 모듈(Supervisory Monitoring and Control Module), 관리 모듈(Management Module) 등이 존재하며 ATM 고속통신망을 통하여 구성요소들이 연결된다. 지역 처리단위는 특정 부분에 대한 각종 센서와 실행 장치(actuator)들을 직접 관리하며 비디오, X 레이, 초음파로 촬영된 동영상 스트림등의 멀티미디어 정보를 통합 데이터 서버로 전송한다.

통합 데이터 서버는 다양한 지역처리단위들로부터 센서 및 멀티미디어 정보를 수집하여 시스템의 상태를 표현하는 전체 화면을 구성하며 필요한 정보를 저장하게 된다. 즉 통합 데이터 서버는 모든 정보를 총괄하며 서비스 품질(QoS ; Quality of Service) 협상과 승인 제어에 대한 기능을 수행하게 된다. 통합 데이터 서버의 필수 기능들은 실시간 데이터 및 제어 서버에서 수행되며 시스템 상태를 나타내는 정보는 실시간 멀티미디어 데이터베이스에 저장된다. 실시간 데이터 및 제어 서버는 각 지역 처리단위로부터 유입되는 실시간 데이터를 수집하고 관리용 감시 및 제어 모듈에게 필요한 정보를 제공한다. 예를 들어 예기치않은 물체가 비디오 스트림에 나타난 경우 시스템은 즉시 문제를 해결하고자 특별한 조치를 강구하게 된다. 따라서 실시간 데이터 및 제어 서버

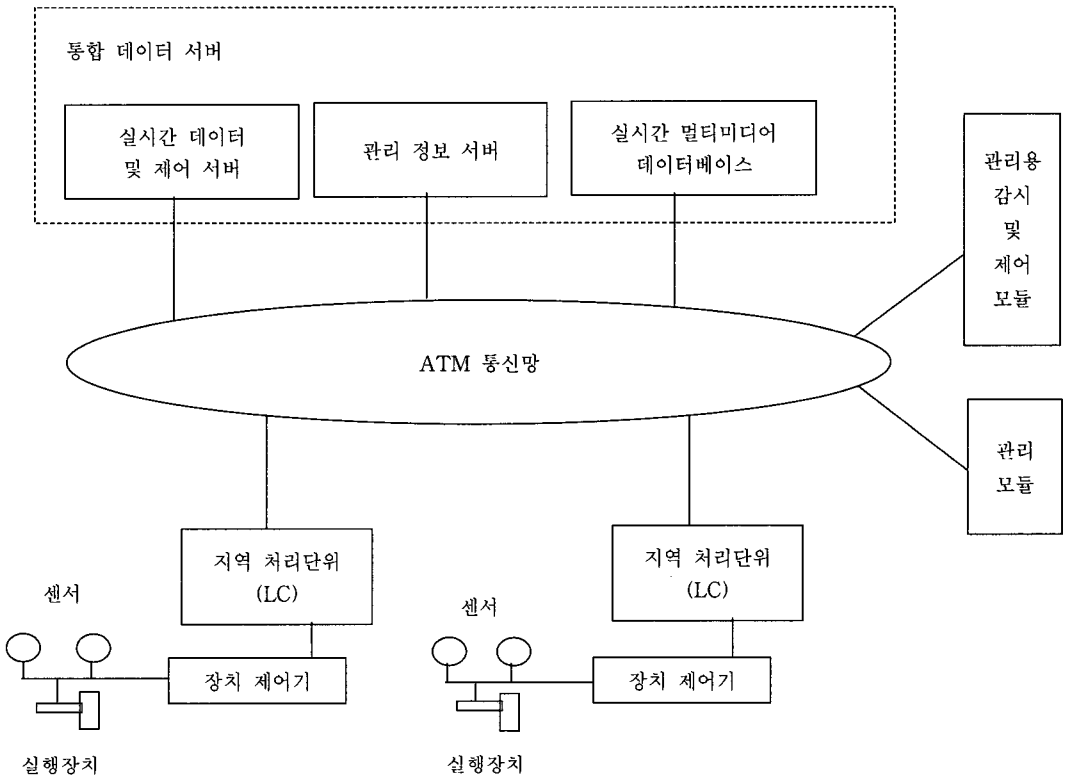
는 다양한 형태의 경성 실시간과 연성 실시간 형태의 태스크들에 대한 승인 제어와 스케쥴링 기능들이 수행되어야 한다. 스케쥴링 가능성 분석 결과에 따라 태스크가 요구한 자원의 할당 여부가 결정되며 서비스 품질이 보장되는 것이다.

한편 실시간 멀티미디어 데이터베이스는 멀티미디어 데이터와 일반 데이터에 대한 저장, 변환, 추출 기능이 수행되며 관리 정보 서버는 실시간 멀티미디어 데이터베이스에 대한 검색과 추출 등 질의 연산이 처리된다.

2.2 자원 관리 정책

멀티미디어 응용은 최적의 성능을 유지하기 위해서 최소한의 자원을 확보 유지하는 것이 필요하다. 즉 (그림 1)의 실시간 데이터 및 제어 서버에게 요구되는 자원은 프로세서 시간, 기억장치 버퍼와 입출력 대역폭 등으로 표현된다. 예를 들어 멀티미디어 태스크로 수행되는 MPEG 비데오 디코더는 센서들이 통신망을 통하여 전송하는 프레임을 처리하거나 하드디스크에 저장된 프레임을 초당 30개씩 처리한다고 가정한다. 이 때 프레임당 처리 시간은 10ms가 되며 주기는 33ms (1/30sec)가 된다. 이 처리시간에는 실제 프레임을 유지하기 위한 물리적 기억 장치의 버퍼 공간 할당과 MPEG 프레임을 판독하고 디코딩된 프레임을 비데오 프레임버퍼에 33ms마다 기록하기 위한 버스 주기(bus cycle)가 포함된다. 본 논문에서는 이론상 CPU 자원의 활용도가 69%인(평균 경우에 대한 분석시 88%까지 증가됨[9]) RM 스케쥴링 기법보다 활용도를 100%까지 증가시킬 수 있는 EDF 스케쥴링 기법[4]을 채택함으로써 자원의 활용도를 극대화하고자 하였다.

CPU에 대한 처리 시간은 프로세스들 간에 효율적으로 공유될 수 있다. 그러나 멀티미디어 응용 특성상 주어진 시간 내에 다량의 데이터를 CPU이외의 장치로 전송하기 위해서는 버스에 대한 보장된 액세스가 필요하다. 운영체제는 기억장치와 주변 장치간에 다량의 데이터를 전송할 수 있는 시스템 호출 기능과 버스 예약을 위한 선점(preemption) 기법이 제공되어야 한다[10]. 본 논문에서도 각 태스크마다 CPU, 버스 그리고 기억장치 버퍼에 대한 자원을 예약할 수 있는 자원 관리자가[10] 스케쥴링 기법과 함께 자원에 대한 예약 서비스를 수행하는 것으로 가정한다[11, 12]. 멀티미디어 응용은 전형적으로 많은 기억장치 버퍼를 사용하는 특성이 있으며 본 논문에서도 기억장치가 논리적으로 세 부분으로 구성되어 있음을 가정한다. 첫 번째 A 부분은 멀티미디어 응용만을 위한 전용 기억장치 공간이며 B 부분은 모든 응용들을 위한 공용 기억장치 공간이다. C 부분은 분산 멀티미디어 응용을 위한 임시 기억장치 공간으로 사용된다. 따라서 분산 멀티미디어 응용은 기억장치의 세 부분 간에 대하여 논리적인 분할이 동적으로 수행된다[10].



(그림 1) 분산 실시간 제어 시스템 구조

기억장치 버퍼는 CPU 또는 버스 전송 예약과 항상 연관이 있기 때문에 태스크나 버스 전송이 수행되기 전에 버퍼가 항상 주기억 장치에 존재하여야 한다. 멀티미디어 태스크가 요구한 공간이 A 부분의 공간에 있는 경우는 전송이 즉시 시작되나 버퍼가 확보되지 않은 경우에는 B 부분의 기억 장치 공간이 할당됨으로써 기억장치 공간의 예약이 수행된다. 예약된 기억장치 버퍼로 데이터를 전송하는 실제 태스크는 스케줄러에 의해 우선순위가 높게 설정된다[10]. 본 논문에서는 CPU 사용에 연관된 기억장치 버퍼 및 입출력 장치에 예약과 선점 기법이 스케줄링 기법에 통합되어 있음을 가정하며 태스크들에 대한 CPU 스케줄링 기법을 제시한다.

3. 최소 주기 서버 스케줄러

3.1 스케줄링 정책

연성 실시간성 특성을 갖는 멀티미디어 태스크들과 경성 실시간 태스크들은 모두 주기를 가지고 있는 것으로 가정한다. 제시된 스케줄링 기법에서는 경성 실시간 또는 멀티미디어 태스크들 중 가장 주기가 작은 태스크의 주기를 서버의 주기로 활용하는 최소 주기 서버(MPS ; Minimal Period Server)가 설정된다. 제시된 서버는 매 주기마다 경성 실시간 태스크들과 멀티미디어 태스크들이 사용하는 CPU 시간을 각각 설정함으로써 CPU 장치의 활용도를 극대화하고자 하였다. 경성 실시간 태스크들을 $H_1, H_2, \cdots, H_n$이라 가정하고 멀티미디어 태스크들을 $M_1, M_2, \cdots, M_m$이라 기술한다. 임의의 경성 실시간 태스크 H_i는 주기 TH_i와 최악의 경우에 대한 실행 시간 CH_i로 구성되는 반면 임의의 멀티미디어 태스크 M_j는 주기 TM_j와 평균 실행 시간인 CM_j로 구성되며 다음과 같이 표현된다.

경성 실시간 태스크 $H_i = (CH_i, TH_i)$
멀티미디어 태스크 $M_j = (CM_j, TM_j)$

일정한 주기마다 수행되는 각 멀티미디어 태스크는 특정한 동영상 스트림이 진행되는 것을 의미한다. 경성 실시간 태스크들과 멀티미디어 태스크들 중 가장 주기가 작은 것을 택하여 두 태스크들을 스케줄링하는 서버의 주기로 설정하며 EDF 스케줄링 원칙에[5] 따라 각 태스크의 실행 시간을 주기로 나눈 값들의 합이 1보다 작거나 같을 때만 태스크들이 스케줄링될 수 있는 것으로 간주한다[13].

$$\sum_{i=1}^{n} \frac{CH_i}{TH_i} + \sum_{j=1}^{m} \frac{CM_j}{TM_j} \le 1 \qquad (1)$$

스케줄링 도중 새로운 경성 실시간 태스크 또는 멀티미디어 태스크가 시스템에 유입되는 경우는 식 (1)에 따라 승인 여부가 결정된다.

태스크들 중 가장 작은 주기로 결정된 서버의 주기를 T_s라 할 때 주기 내에서 경성 실시간 태스크들을 위한 실행 시간은 E_H로 멀티미디어 태스크들을 위한 실행 시간은 E_M으로 정의하며 다음과 같이 결정된다.

$$E_H = \sum_{i=1}^{n} \left(CH_i \frac{T_s}{TH_i} \right) \qquad (2)$$

$$E_M = \sum_{j=1}^{m} \left(CM_j \frac{T_s}{TM_j} \right) \qquad (3)$$

경성 실시간 태스크 H_i가 최소주기 서버내에서 차지하는 실행 시간은 자신의 실행 시간 CH_i를 T_s/TH_i 비율과 곱한 값으로 결정된다. 즉 최소 주기보다 주기가 큰 경성 실시간 태스크들은 서버가 두 번 이상 수행됨으로써 실행될 수 있다. 멀티미디어 태스크 M_j도 마찬가지 방법으로 자신의 평균 실행 시간 CM_j를 T_s/TM_j와 곱한 값이 최소 주기 내에서의 실행 시간으로 결정된다. 그리고 E_H와 E_M의 합 E_s는 최소주기 서버내에서의 실행 시간이 되며 주기내에서의 CPU 활용도 U_s는 식 (5)와 같이 설정된다.

$$E_s = E_H + E_M \qquad (4)$$

$$U_s = \frac{E_s}{T_s} = \frac{E_H + E_M}{T_s} = \frac{E_H}{T_s} + \frac{E_M}{T_s} = U_H + U_M \qquad (5)$$

식 (5)에서 U_H와 U_M은 경성 실시간 태스크와 멀티미디어 태스크에 대한 CPU 활용도를 나타내며 각각 $\frac{E_H}{T_s}$와 $\frac{E_M}{T_s}$로 설정된다.

경성 실시간 태스크는 최악의 경우에 대한 실행 시간을 가정하기 때문에 일정 회수 이상의 서버가 수행되면 반드시 종료시한내에 실행이 종료되는 것이 보장되는 반면 멀티미디어 태스크는 평균 실행 시간을 가정하기 때문에 실제로 요구된 실행 시간이 평균 시간보다 큰 경우 설정된 종료시한이 경과될 수 있다. 예제 1은 경성 실시간 태스크 H_1과 H_2 그리고 멀티미디어 태스크 M_1과 M_2에 대한 실행 시간과 주기를 나타내고 있으며 이에 대한 최소주기 T_s와 E_H, E_M, CPU 활용도 U_s가 계산된 결과이다. 모든 단위 시간은 ms이다.

예제 1. $H_1 = (5, 30)$, $H_2 = (15, 50)$, $M_1 = (8, 40)$,
$M_2 = (16, 60)$, $T_s = 30$,

$$E_H = CH_1 \times \frac{T_s}{TH_1} + CH_2 \times \frac{T_s}{TH_2}$$

$$= 5 \times \frac{30}{30} + 15 \times \frac{30}{50} = 14$$

$$E_M = CM_1 \times \frac{T_s}{TM_1} + CM_2 \times \frac{T_s}{TM_2}$$

$$= 8 \times \frac{30}{40} + 16 \times \frac{30}{60} = 14$$

$$E_s = E_H + E_M = 14 + 14 = 28$$

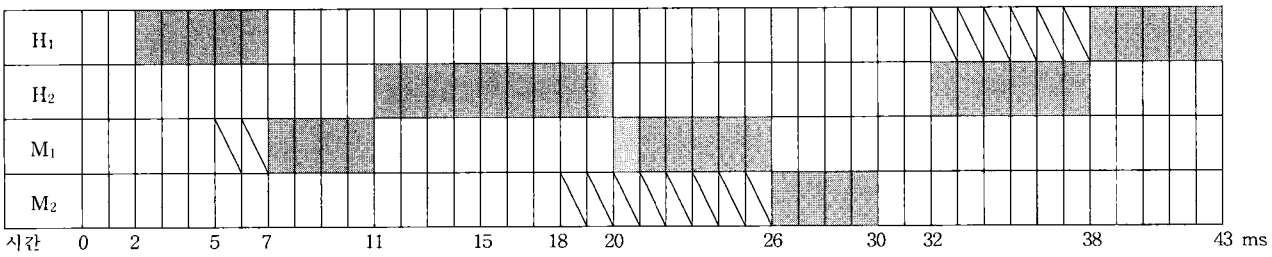
$$U_s = U_H + U_M = \frac{14}{30} + \frac{14}{30} = \frac{28}{30}$$

예제 1에서 H_2는 서버의 주기 T_s(= 30ms)마다 9(= $15 \times \frac{30}{50}$) ms의 실행 시간을 가지게 되며 M_1과 M_2는 각각 6(= $8 \times \frac{30}{40}$) ms와 8(= $16 \times \frac{30}{60}$)ms 평균 실행 시간을 가지게 된다. 경성 실시간 태스크들은 종료시한을 항상 보장하기 위하여 서버의 주기마다 할당된 시간만큼만 정확히 실행되는 반면 멀티미디어 태스크들에 대해서는 실행시간의 가변성을 고려하므로 처리 방법이 상이하다. M_1의 평균 실행 시간이 8ms로 설정되어 있지만 처리하는 데이터의 크기에 따라 실제 수행시 이보다 많은 실행 시간이 요구될 수 있으며 M_1의 실행 시간이 10ms라고 가정한다. 이 때 서버의 주기내에서 M_1에 할당된 시간은 6ms이나 M_1의 연속적 실행을 위하여 서버는 M_2에 할당된 8ms 시간중 4ms 시간을 M_1에 할당하게 된다. 즉 제시된 최소주기 서버는 특정 멀티미디어 태스크의 실행 시간이 평균 실행 시간보다 커지는 경우에 다른 멀티미디어 태스크에 할당된 시간을 신축성있게 활용함으로써 종료시한이 경과되는 시간을 최소화할 수 있다. 물론 멀티미디어 태스크의 실행은 경성 실시간 태스크들에 아무런 영향을 미치지 않도록 우선순위에 따른 선점 정책이 사용된다. 즉 우선순위가 높은 경성 실시간 태스크는 항상 우선순위가 낮은 멀티미디어 태스크들을 선점함으로써 서버의 주기내에서 할당된 시간만큼 실행되며 자신의 종료시한이 항상 보장된다. 그러나 경성 실시간 태스크들 간에는 종료시한에 따른 EDF 기반의 우선순위에 의해서만 실행 순서가 결정되며 선점 현상이 없고 멀티미디어 태스크들 간에도 EDF 기반의 우선순위로 실행순서가 결정되며 역시 선점 현상이 없다[14]. 경성 실시간 태스크들과 멀티미디어 태스크들 간에만 선점 정책이 적용된다.

제시된 MPS 스케쥴링 기법에서는 특정 멀티미디어 태스크가 예상했던 평균 실행 시간보다 실제 실행 시간이 적어지거나 오래걸리는 경우 서버의 주기내 설정된 멀티미디어 태스크들의 실행 시간(E_M)의 범위내에서 다른 멀티미디어 태스크의 실행 시간을 조정함으로써 신축성있게 태스크들이 스케쥴링될 수 있다. 또한 경성 실시간 태스크들과 멀티미디어 태스크들에 대한 주기내 설정된 실행 시간을 변경함으로써 일시적으로 급증하는 태스크들도 신축성있게 스케쥴링할 수 있다. 특히 매우 중요한 경성 실시간 태스크가 시스템에 유입되는 경우에 기존 멀티미디어 태스크들의 실행 시간(E_M)을 감소시킨만큼 경성 실시간 태스크들의 실행 시간(E_H)을 증가시킴으로써 멀티미디어 태스크들의 실행을 지연시키고 새로 유입된 경성 실시간 태스크의 종료시한을 보장하게 된다. 멀티미디어 태스크들의 실행 시간 지연에 따른 서비스 품질의 저하 기법과 이에 대한 구체적 사항은 본 논문에서 생략한다.

(그림 2)는 예제 1에 대한 태스크들의 실행 순서와 시간을 나타내고 있으며 실제 태스크가 실행된 시간은 음영 부분으로 표현되며 태스크가 도착한 후 실행되기 전까지의 시간은 사선으로 표현된다. 최소주기를 갖는 H_1의 주기를 서버의 주기로 설정하였기 때문에 서버는 항상 H_1과 같은 주기로 수행되며 H_1이 2ms에서 시작되었다고 가정한다. H_1은 서버가 유휴 상태에 있을 때 도착하였으므로 5ms 동안 실행되고 7ms에서 종료된다. 이때 E_H는 14ms에서 9ms로 감소된다. M_1은 5ms에서 도착하였으나 서버가 경성 실시간 태스크인 H_1을 실행 중이므로 멀티미디어 태스크를 위한 대기 행렬 Queue_M에 유입된다. 7ms에서 H_1이 종료된 후 M_1이 시작되며 실제 실행 시간은 평균 시간 8ms보다 2ms가 늘어난 10ms라고 가정한다. M_1이 실행중인 11ms에서 H_2가 도착한 경우 서버는 M_1을 다시 Queue_M에 저장하고 H_2를 실행하게 된다. 즉 우선순위가 낮은 멀티미디어 태스크 M_1이 경성 실시간 태스크 H_2에 의해 선점된 것이다. 이 때 E_M은 14ms에서 10ms로 감소되며 M_1의 남은 실행 시간은 6ms가 된다. H_2는 서버의 주기내에 할당된 9(= $15 \times \frac{30}{50}$)ms 동안 실행된 후 나머지 6(= 15 - 9)ms의 실행 시간을 서버의 다음 주기에 사용하고자 경성 실시간 태스크들의 대기 행렬인 Queue_H에 유입되며 E_H는 0ms가 된다.

그리고 18ms에서 시스템에 도착한 M_2는 서버가 작동중



(그림 2) 예제 1에 대한 MPS 스케쥴링 방법

이므로 역시 Queue_M에 유입된다. 20ms에서 H_2의 실행이 종료되고 서버는 EDF 우선순위 정책에 따라 M_1과 M_2중 종료시한이 더 빠른 M_1을 실행하게 된다. M_1은 자신의 남은 실행 시간인 6ms 동안 실행된 후 26ms에서 종료되며 E_M은 10ms에서 4ms로 감소된다. 이후 M_2는 평균 실행 시간과 일치하는 16ms의 실행 시간이 요구된 것으로 가정하고 남은 E_M 시간인 4ms만큼만 CPU를 사용하게 된다. 30ms에서는 E_H와 E_M이 모두 0이 되며 32ms에서 다시 H_1이 도착하면서 새로운 서버가 수행되고 E_H와 E_M이 각각 14ms로 다시 설정된다. 이때는 Queue_H에 이미 H_2가 존재하며 H_2의 종료시한(61ms = 11ms + 50ms)이 H_1의 종료시한(62ms = 32ms + 30ms)보다 빠르므로 H_2가 먼저 실행된다. 주기내에 H_2에 할당된 시간은 9ms이나 남은 실행 시간이 6ms이므로 38ms에서 H_2는 종료되고 새로 도착한 이후 Queue_H에 유입되었던 H_1이 5ms 동안 실행된다. 그리고 H_1이 종료된 43ms에서는 Queue_M에 있던 M_2가 다시 실행된다. 매 주기마다 도착하는 태스크들은 최소주기 서버(MPS)에 의하여 이와 같은 방법으로 스케쥴링된다.

3.2 MPS 알고리즘

제시된 MPS 알고리즘은 (그림 3)과 같이 기술된다. 최소주기 서버가 설정된 후 매 주기마다 서버는 경성 실시간 태스크들의 대기 행렬인 Queue_H를 먼저 탐색하며 태스크 H_i의 주기 내 할당 시간인 Allot_time(H_i)이 0보다 크면서 우선순위가 가장 높은 태스크 H_s를 실행시킨다.

H_s의 우선순위는 알고리즘에서 Prior(H_s)로 정의되며 지금까지의 우선순위가 가장 높았던 경성 실시간 태스크의 우선순위 High_prior_H와 비교된다. 우선순위가 더 높아도 주기내 할당 시간이 이미 0이 된 태스크는 설정된 시간을 모두 사용했기 때문에 다음번 서버가 도착할 때까지 Queue_H에 머무르게 된다. H_s는 자신의 할당 시간인 Allot_time(H_s) 이하의 시간만큼(실제로 H_s의 남은 실행 시간이 Allot_time(H_s)보다 작을 수 있으며 이 경우 남은 실행 시간만큼만 CPU를 사용하게 됨) CPU를 사용하며 이 과정은 (그림 3)에서 Execute(Allot_time(H_s))로 표현된다. 이후 H_s의 실행 시간 CH_s가 0이 된 경우 종료되며(Kill(H_s)) 과정으로 기술됨) 0보다 큰 경우는 다음 번 실행을 위하여 다시 Queue_H에 유입된다(Queue(H_s) in Queue_H 과정으로 표현됨).

이 때 서버는 Queue_H를 다시 탐색하며(goto Search_Queue_H 과정으로 기술됨) 조건에 부합하는 경성 실시간 태스크들이 없는 경우에 멀티미디어 태스크들의 대기 행렬인 Queue_M을 탐색한다. 서버는 Queue_M에 존재하는 태스크들 중 우선순위가 가장 높은 것을 High_prior_M으로 정의하면서 모든 태스크들을 비교하여 종료시한이 가장 이른 (즉 우선순위가 가장 높은) 태스크 M_s를 선정하여 실행하게 된다(Execute(M_s))로 기술함). M_s 태스크의 우선순위는 Prior(M_s)로 정의된다. 이때 태스크 M_s의 실제 실행 시간이 평균 실행 시간 CM_s보다 커지는 경우에도 계속 CPU를 사용할 수 있으나 서버의 주기내에서 멀티미디어 태스크들에 할당된 실행 시간 E_M을 초과할 수는 없다. M_s 태스크는 실행 도중에 새로운 경성 실시간 태스크 H_{new}가 시스템에 도착하는 경우 선점되며(Preempt(M_s)로 기술됨) 이제까지의 실행 시간(M_s가 실행된 시간을 Execute_time(M_s)로 기술함)을 E_M과 CM_s에서 각각 감하게 된다. H_{new}가 없었던 경우에는 M_s 태스크가 선점 현상없이 정상적으로 실행된 시간(Execute_time(M_s))을 E_M과 CM_s에서 각각 감하게 된다. CM_s가 0이 되는 경우에는 해당 멀티미디어 태스크 M_s의 실행이 종료되며 시스템에서 제거되고(Kill(CM_s)로 기술됨) CM_s가 0보다 큰 경우에는 다음 번 서버의 주기때 실행되도록 Queue_M에 다시 유입된다. 서버가 작동 중인 경우 시스템에 도착한 경성 실시간 태스크나 멀티미디어 태스크는 일단 해당 대기 행렬에 유입된 후 서버에 의하여 스케쥴링되는 것을 가정한다. 따라서 서버는 Queue_H에 도착한 새로운 태스크 H_{new}를 다른 태스크들의 우선순위와 주기내 할당된 실행시간을 모두 비교한 후 실행할 태스크로 선정하게 되면 Allot_time(H_{new})

```
Search_Queue_H :
    for Hi in Queue_H
        if Allot_time(Hi) > 0 and Prior(Hi) > High_prior_H
            Hs = Hi ;
    if Hs is not Null and EH > 0
     begin
        Execute(Allot_time(Hs)) ;
        EH = EH - Allot_time(Hs) ;
        CHs = CHs - Allot_time(Hs) ;
        Allot_time(Hs) = 0 ;
        if CHs = 0
          Kill(Hs) ;
        else
          Queue(Hs) in Queue_H ;
      end
    goto Search_Queue_H ;
    for Mi in Queue_M
        if Prior(Mi) > High_prior_M
          Ms = Mi ;
    if Ms is not null and EM > 0
     begin
        Execute(Ms) ;
          if Hnew arrives
            Preempt(Ms) and Queue(Hnew) in Queue_H ;
        EM = EM - Execute_time(Ms) ;
        CMs = CMs - Execute_time(Ms) ;
        if CMs = 0
          Kill(Ms) ;
        else
          Queue(Ms) in Queue_M ;
     end
    goto Search_Queue_H ;
```

(그림 3) 최소주기 서버 알고리즘

동안 H_{new}를 실행하게 된다.

4. 성능 분석

4.1 시뮬레이션 모델

제시된 최소주기 서버의 스케쥴링 기법은 주기가 상이한 5개의 경성 실시간 태스크들과 5개의 멀티미디어 태스크들을 대상으로 성능이 분석되었다. 먼저 경성 실시간 태스크들의 주기와 최악의 경우에 대한 실행 시간(단위 ms)은 다음과 같이 정의된다.

$H_1 = (30, 3)$, $H_2 = (50, 5)$, $H_3 = (70, 7)$, $H_4 = (90, 9)$, $H_5 = (110, 11)$

멀티미디어 태스크들의 주기와 평균 실행 시간은 다음과 같다.

$M_1 = (40, 4)$, $M_2 = (60, 6)$, $M_3 = (80, 8)$, $M_4 = (100, 10)$, $M_5 = (120, 12)$

멀티미디어 태스크들의 실제 실행 시간은 1과 2*평균 실행 시간 -1 사이에 존재하는 임의의 값이 설정되도록 균등 분포 함수가 이용되었다. 경성 실시간 태스크들과 멀티미디어 태스크들의 CPU 자원에 대한 활용도 U_H와 U_M 그리고 전체 태스크들에 대한 스케쥴링 가능성을 나타내는 U_s는 다음과 같이 기술된다.

$$U_H = \frac{3}{30} + \frac{5}{50} + \frac{7}{70} + \frac{9}{90} + \frac{11}{110} = 0.5,$$

$$U_M = \frac{4}{40} + \frac{6}{60} + \frac{8}{80} + \frac{10}{100} + \frac{12}{120} = 0.5,$$

$U_s = U_H + U_M = 1.0 \le 1.0$ [5].

10개의 태스크들에 대한 CPU 자원의 활용도 U_s가 1.0이며 이는 EDF 스케쥴링 방법에서 이론적 최대값인 1.0과 동일하므로[5] 모든 태스크들이 스케쥴링될 수 있음을 의미한다. U_H와 U_M이 동일하게 0.5이며 이는 최소 주기가 30ms인 H_1의 주기를 서버의 주기로 설정할 때 경성 실시간 태스크들과 멀티미디어 태스크들은 각각 15ms만큼의 실행 시간 E_H와 E_M을 각각 가지게 된다. 최소 주기 T_s(= 30ms)내에서 경성 실시간 태스크 H_i에 할당되는 실행 시간은 다음과 같이 계산된다.

$$\text{Allot}(H_i) = CH_i \times \frac{T_s}{TH_i}$$

실제 5개의 경성 실시간 태스크들은 서버의 주기내에서 각각 3ms의 실행 시간이 할당됨으로써 15ms인 E_H가 구성된다.

$$E_H = \sum_{i=1}^{5} \text{Allot}(H_i) = 3 \times \frac{30}{30} + 5 \times \frac{30}{50} + 7 \times \frac{30}{70} + 9 \times \frac{30}{90} + 11 \times \frac{30}{110}$$

$= 3 + 3 + 3 + 3 + 3$

$= 15\text{ms}$

그리고 멀티미디어 태스크들에 대한 실행 시간 E_M도 다음과 같이 계산된다.

$$E_M = \sum_{j=1}^{5} CM_j \times \frac{T_s}{TM_j} = 4 \times \frac{30}{40} + 6 \times \frac{30}{60} + 8 \times \frac{30}{80} + 10 \times \frac{30}{100} + 12 \times \frac{30}{120}$$

$= 3 + 3 + 3 + 3 + 3$

$= 15\text{ms}$

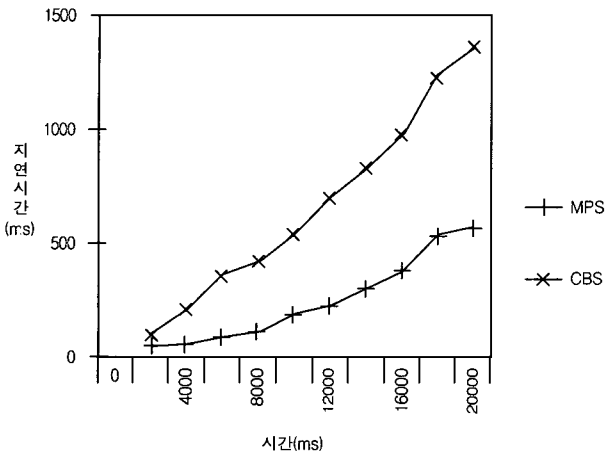
본 실험에서 CPU에 대한 태스크들의 문맥 교환(context switching)시 이에 수반되는 오버헤드는 고려하지 않았다. 실제 구현시 각 태스크는 쓰레드(thread) 단위로 실행됨으로써 오버헤드가 최소화될 수 있다[1].

제시된 최소주기 서버 스케쥴링 방법은 고정 대역폭 스케쥴링(CBS ; Constant Bandwidth Scheduling)[3]과 성능이 비교 분석되었다. CBS 기법은 멀티미디어 태스크의 평균 실행 시간과 주기를 기반으로 모든 멀티미디어 태스크마다 별도의 서버가 설정되며 평균 시간을 초과하는 CPU 사용 시간이 요청된 경우 서버의 다음 번 주기때 이를 수용하는 스케쥴링 방법이다. 이 방법에서는 특정 멀티미디어 태스크의 실행 시간이 예상 평균 시간보다 증가한 경우 다른 멀티미디어 태스크의 실행 시간을 이용할 수 없다. 또한 멀티미디어 태스크의 주기에 대한 특별한 고려가 없으며 데이터 크기 변동에 따른 실행 시간의 변동시 멀티미디어 태스크들의 종료시한이 경과되는 시간이 증가하게 된다.

4.2 종료시한이 경과된 지연 시간

최소주기 서버의 목표는 경성 실시간 태스크들의 종료시한을 모두 보장하면서 멀티미디어 태스크들의 종료시한이 경과되는 시간을 최소화시키는 것이다. 본 실험에서는 경성 실시간 태스크 5개와 멀티미디어 태스크 5개를 동일한 시스템에서 수행함으로써 멀티미디어 태스크들의 종료시한에 대한 지연 시간을 측정하였다. MPS 기법과 CBS 기법 모두 경성 실시간 태스크들의 종료시한을 정확히 보장하는 실험 결과가 확인되었다. (그림 4)는 멀티미디어 태스크들에 대해서 매 2000ms 마다 종료시한이 경과된 시간의 합을 해당 멀티미디어 태스크들의 수로 나눈 평균 지연 시간을 측정한 결과이다. 제시된 MPS 기법은 CBS 기법보다 종료시한을 초과하는 평균 지연 시간이 보다 작은 것으로 확인되었다. 즉 평균 실행 시간을 초과하는 멀티미디어 태스크를 최소주기 서버내의 멀티미디어 태스크들에 설정된 실행 시간 E_M의 범위내에서 가능하면 계속적으로 실행하는 것이 종료시한이 경과되는 시간

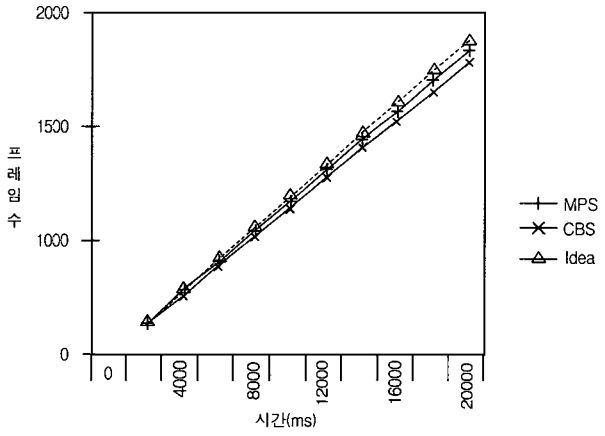
을 감소시킬 수 있음을 의미한다. CBS 기법의 경우 항상 주기내에 설정된 평균 시간만큼만 해당 태스크를 스케쥴링하기 때문에 MPS 기법보다 종료시한이 경과되는 지연 시간이 커지게 된다.



(그림 4) 종료시한이 경과된 지연 시간

<표 1>은 매 1000ms마다 종료시한이 충족된 멀티미디어 태스크들의 비율을 백분율로 나타내고 있다. 5000ms이후에 MPS 방법의 실기율이 100%에 도달하는 이유는 이미 종료시한이 경과된 멀티미디어 태스크들이 계속 시스템에 존재하게 되고 새로 도착하는 멀티미디어 태스크들은 이미 종료시한이 경과된 태스크들보다 우선순위가 낮기 때문에 실행이 시작되는 시간 자체가 종료시한에 임박하거나 경과한 이후가 된다. 결과적으로 5000ms 이후의 모든 멀티미디어 태스크들은 종료시한이 경과하게 되며 CBS 방법은 이러한 현상이 MPS 방법보다 더 빨리 나타났다. 즉 1000ms 이후에는 CBS 방법의 경우 모든 멀티미디어 태스크들이 종료시한을 경과하게 된다. <표 2>는 CPU가 실제로 사용된 활용도를 백분율로 나타내고 있다. <표 2>에서 CBS 방법의 CPU 활용도가 100%에 도달하지 못하는 이유는 서버의 주기마다 멀티미디어 태스크들에 할당된 CPU 시간을 해당 태스크들이 100% 모두 활용하지 못한 것으로 분석된다. 즉 CBS 기법은 임의의 멀티미디어 태스크가 예상 평균 시간을 초과하여 CPU를 사용하려는 경우 서버의 다음 번 주기째 이에 대한 요청이 수용되며 다른 멀티미디어 태스크가 사용하고 남은 CPU 자원을 활용하지 못하는 것이다.

4.3 디코드된 프레임의 수



(그림 5) 디코드된 프레임 수

(그림 5)는 매 2000ms 시간마다 멀티미디어 태스크들에 의해 실제로 디코드된 프레임의 수를 나타내고 있다. Ideal 경우는 각 태스크가 종료시한을 경과하지 않은 상태에서 프레임을 디코드한 결과를 의미한다. 제시된 MPS 기법이 Ideal 경우보다는 디코드한 프레임들의 수가 작지만 CBS 기법보다는 동일한 시간 간격동안 더 많은 수의 프레임들을 디코드하는 것으로 파악되었다. 서버의 주기마다 멀티미디어 태스크들이 사용할 수 있는 실행 시간을 최대한 활용한 결과로 분석된다. 그리고 (그림 6)은 주기가 80ms이며 평균 실행 시간이 8ms인 멀티미디어 태스크 M_3에 대하여 한 프레임을

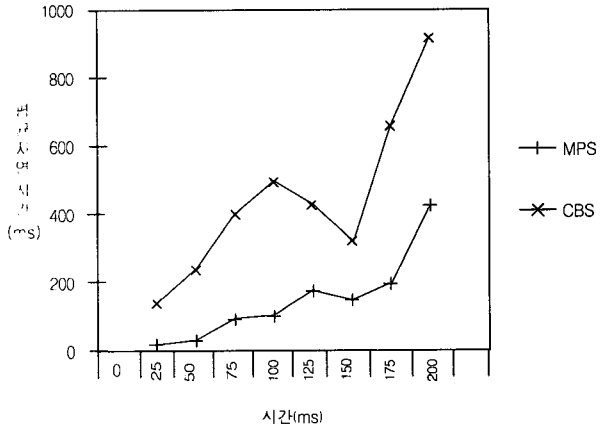
〈표 1〉 멀티미디어 태스크들의 종료시한 실기율

시간 / 스케쥴러	1000ms	2000ms	3000ms	4000ms	5000ms	6000ms	7000ms	8000ms
MPS	17.64%	35.25%	64.10%	82.21%	100.0%	100.0%	100.0%	100.0%
CBS	69.82%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

〈표 2〉 CPU 활용도

시간 / 스케쥴러	1000ms	2000ms	3000ms	4000ms	5000ms	6000ms	7000ms	8000ms
MPS	95.98%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
CBS	93.02%	94.23%	94.86%	95.46%	96.34%	97.73%	98.46%	99.24%

처리하는 소요시간을 측정하여 비교한 결과를 보여주고 있다. 각 기법에서 태스크에 따라 평균 지연 시간이 차이나는 것은 디코드되는 프레임의 크기에 따라 실제 실행 시간이 예상 평균 시간과 다르기 때문이다. 실험 결과 동일한 프레임을 처리하는 경우 MPS 기법은 CBS 기법보다 한 프레임을 처리하는 실제 실행 시간이 작은 것으로 파악되었으며 이러한 결과는 나머지 네 개의 태스크들에서도 모두 유사한 결과가 확인되었다. 본 논문에서는 M_3에 대한 프레임 처리 시간을 비교한 결과만을 제시한다.



(그림 6) 주기가 80ms인 멀티미디어 태스크의 평균 지연 시간

5. 결 론

멀티미디어 정보와 데이터를 제어 목적으로 활용할 수 있는 분산 실시간 멀티미디어 데이터베이스 시스템을 위한 스케쥴링 기법이 본 논문에서 제시되었다. 경성 실시간 태스크들과 연성 실시간성인 멀티미디어 태스크들이 동일한 시스템에 수행되는 경우 이 태스크들을 효율적으로 스케쥴링할 수 있는 최소주기 서버가 본 논문의 스케쥴링 기법에서 설정된다. 태스크들의 주기중 가장 작은 것이 서버의 주기로 설정되며 주기내에서 경성 실시간 태스크들과 멀티미디어 태스크들이 CPU를 사용할 수 있는 시간을 각각 설정한다. 특히 경성 실시간 태스크들은 서버의 주기마다 일정 시간만큼만 실행되는 반면 멀티미디어 태스크들은 예상했던 평균 시간보다 CPU를 더 많이 사용하려는 경우에도 멀티미디어 태스크들에 대한 전체 실행 시간 범위내에서는 해당 태스크가 계속적으로 실행될 수 있도록 신축성있게 스케쥴링되고 있다. 결과적으로 경성 실시간 태스크들은 모두 종료시한내에 종료될 수 있으며 멀티미디어 태스크들은 종료시한이 경과되는 지연 시간이 최소화될 수 있다. 실제 프레임을 구성하는 데이터의 크기가 예상보다 큰 경우에도 가능하면 화면에 출력되는 시간이 최소화되도록 함으로써 멀티미디어 태스크의 실시간성이 향상되는 분산 제어 환경이 구축되도록 하였다.

경성 실시간 태스크들의 실시간성을 모두 보장하면서 멀티미디어 태스크들의 종료시한이 경과되는 시간을 최소화하고자 하는 스케쥴링 기법은 일정 개수의 프레임들을 하나의 그룹으로 처리하는 MPEG 디코더에 적용되도록 향후 스케쥴링 알고리즘이 개선될 예정이다.

참 고 문 헌

[1] H. Kaneko, and et al., "Integrated Scheduling of Multimedia and Hard Real-time Tasks," In proc. of IEEE Real-Time Systems Symposium, Dec. 1996.

[2] O. Gonzalez and et al., "Incorporation of Multimedia Capabilities in Distributed Real-time Applications," Workshop on Databases : Active and Real-Time, Nov. 1996.

[3] L. Abeni and et al., "Integrating Multimedia Applications in Hard Real-time Systems," In proc. of IEEE Real-Time Systems Symposium, Dec. 1998.

[4] C. L. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment," Journal of the ACM, Vol.20, No.1, 1973.

[5] K. Jeffay, "Scheduling Sporadic Tasks with Shared Resources in Hard Real-time Systems," In proc. of IEEE Real-Time Systems Symposium, Dec. 1992.

[6] L. Sha and et al., "Priority Inheritance Protocols : an Approach to Real-time Synchronization," IEEE Transactions on Computers, Vol.39, No.9, 1990.

[7] C. W. Mercer and et al., Processor Capacity Reserves for Multimedia Operating Systems, Technical Report CMU-CS-93-157, Carnegie Mellon University, May, 1993.

[8] L. Abeni and G. Buttazzo, "Adaptive Bandwidth Reservation for Multimedia Computing," In proc. of IEEE Conf. on Real-Time Computing Systems and Applications, Dec. 1999.

[9] J. P. Lehoczky and et al., "The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behavior," In proc. of IEEE Real-Time Systems Symposium, Dec. 1989.

[10] S. Lakshminarayanan and K. Mahesh, "Efficient End-host Resource Management with Kernel Optimizations for Multimedia Applications," In proc. of ECMAST, pp. 46-57, 1999.

[11] C. W. Mercer, S. Savage and H. Tokuda, "Processor capacity reserves : operating systems support for multimedia applications," In Proc. of IEEE International Conference on Multimedia Computing and Systems, May, 1994.

[12] C. Lee, R. Rajkumar and C. Mercer, "Experience with pro-

cessor reservation and dynamic QoS in real-time Mach," In Proc. of Multimedia Japan, March, 1996.

[13] I. Stoica, H. Abdel-Wahab and K. Jeffay, "On the duality between resource reservation and proportional share resource allocation," In Proc. of Mutltimedia Computing and Networking, Feb. 1997.

[14] M. Caccamo, G. Lipari and G. Buttazzo, "Sharing resources among periodic and aperiodic tasks with dynamic deadlines," In Proc. of IEEE Real-Time Systems Symposium, Dec. 1999.

김 진 환

e-mail : kimjh@ice.hansung.ac.kr

1986년 서울대학교 컴퓨터공학과 졸업(학사)

1988년 서울대학교 대학원 컴퓨터공학과(석사)

1994년 서울대학교 대학원 컴퓨터공학과(박사)

1994년~1996년 서울대학교 컴퓨터신기술공동연구소 특별연구원

1995년~현재 한성대학교 정보전산학부 부교수

관심분야 : 분산실시간 시스템, 멀티미디어 시스템, 지능형 교통 시스템 등