MPEG-2 디코딩을 위한 멀티미디어 시스템에서 우선순위에 의한 태스크 스케쥴링 기법

김 진 환†

요 약

본 논문에서는 MPEG-2 비디오 스트림의 프레임을 디코딩하는 멀티미디어 태스크에 대한 효율적인 실시간 스케쥴링 기법이 제시된다. 태스크 모델에서 각 프레임은 각각의 멀티미디어 태스크에 의하여 디코딩되며 각 태스크는 비디오 스트림내 프레임의 순서와 중요도에 따라 우선순위가 설정된다. MPEG-2 비디오 스트림의 디코딩 기능을 수행하는 멀티미디어 태스크마다 CPU 대역폭을 효과적으로 할당하고자 우선순위에 기반한 스케쥴링 기법을 사용하는 것이다. 본 논문에서 멀티미디어 태스크에 할당된 CPU 대역폭을 우선순위에 따라 동적으로 제어하는 방법이 기술된다. 우선순위에 의한 스케쥴링 기법의 주요 목적은 멀티미디어 태스크들의 디코딩 시간을 감소시키는 한편 종료시한 이후에 실행이 완료되는 디코딩 태스크의 수를 최소화함으로써 멀티미디어 시스템의 실시간적 성능을 향상시키는 것이다. 제시된 스케쥴링 기법의 성능은 시뮬레이션 실험을 통하여 다른 기법과 비교 분석된다.

A Prioritized Task Scheduling Method in Multimedia Systems for MPEG-2 Decoding

Jinhwan Kim†

ABSTRACT

In this paper, we propose an efficient real-time scheduling method of multimedia tasks for decoding frames of MPEG-2 video streams. In our task model, each frame is decoded by a separate multimedia task. The decoding task for each frame is assigned to the priority according to the precedence and importance of frames in a video stream. We use a priority-based scheduling policy in order to effectively allocate the CPU bandwidth to multimedia tasks for MPEG-2 decoding. We show how to dynamically control the fraction of the CPU bandwidth allocated to each multimedia task according to the priority. The primary purpose of our scheduling method is to enhance the real-time performance of the multimedia system by minimizing the number of decoding tasks that have missed their deadlines while reducing the decoding times of these multimedia tasks. The performance of this scheduling method is compared with that of similar mechanisms through simulation experiments.

키워드 : 우선순위(priority), 실시간(real-time), 멀티미디어 태스크(multimedia task), 종료시한(deadline), 스케쥴링(Scheduling), MPEG

1. 서 론

현재 MPEG-2는 비디오와 오디오 스트림의 압축을 위하여 널리 사용되고 있는 표준 기법이며 다양한 크기와 품질을 수용하고 있다[1]. 멀티미디어 시스템에서 서비스 품질(QoS; Quality of Service)과 자원 예약은 MPEG-2 비디오와 오디오 스트림 처리시 중요한 역할을 수행하게 된다[2]. 실제 비디오와 오디오 스트림 같은 연속적인 미디어를 처리하는 응용은 시스템 자원을 다양하게 요구할 수 있으며 이는 멀티미디어 시스템에 대한 부하를 동적으로 변경시키는 요인이 될 수 있다[3]. 멀티미디어 시스템의 서비스 품질을 보장하기 위해서는 자원에 대한 요구가 수용될 수 있는 지에 대한 여부를 결정하는 수락 제어(admission control) 절차가 운영체제 내에 반드시 필요하다. 그러나 일반적인 운영체제는 멀티미디어 태스크의 다양한 요구를 수용하기 어려우며 비디오 지터(jitter) 현상 혹은 느린 상호 응답 시간으로 인하여 예측성이 어려워지는 한편 시스템의 성능도 저하된다[4]. 따라서 MPEG-2 비디오 스트림을 디코딩하는 멀티미디어 시스템에서 응용의 서비스 품질을 보장하기 위해서는 실시간 CPU 스케쥴링 기법이 반드시 필요하다. 최근 일부 스케쥴링 기법이 멀티미디어 스케쥴링 특성을 반영하고 있기는 하나 MPEG-2 압축 표준 기법의 특성이 충분히 고려되지 않고 있다[4, 5].

* 본 연구는 2005년도 한성대학교 교내연구비 지원과제임.
† 정 회 원 : 한성대학교 컴퓨터공학부 부교수
논문접수 : 2004년 4월 23일, 심사완료 : 2005년 1월 24일

이외에도 멀티미디어 태스크를 스케줄링하는 기법들은 이미 여러 논문에서 발표된 바 있다[2, 3, 6-8]. 비디오 및 오디오 등의 멀티미디어 정보가 이용되는 시스템은 종료시한(deadline)을 반드시 보장해야 하는 경성 실시간(hard real-time) 시스템과는 달리 지연 시간(tardiness)과 지터(jitter) 허용 등 연성 실시간적(soft real-time) 특성이 존재한다[9]. 최악의 실행 시간(WCET; Worst Case Execution Time)을 기반으로 하는 경성 실시간 태스크들과는 달리 연성 실시간 태스크들은 종료시한이 경과되더라도 시스템에 미치는 영향이 심각하지 않기 때문에 대부분 평균 실행 시간을 기반으로 스케줄링되고 있다. 기존의 실시간 스케줄링 기법인 최조종료시한(EDF; Earliest Deadline First) 기법이나[10, 11] RM(Rate Monotonic) 기법은[10, 12] 사실상 경성 실시간 태스크들을 위한 기법이므로 각 태스크의 WCET가 종료시한 내에 실행되는 것을 보장하고 있다. 그러나 처리하고자 하는 데이터의 규모에 따라 실행시간이 가변적인 멀티미디어 태스크는 WCET를 기반으로 하는 경성 실시간 태스크의 스케줄링 기법을 적용할 경우 CPU 대역폭의 낭비가 심해지며 결과적으로 자원의 활용도가 낮아지는 문제가 발생하게 된다[6]. MPEG-2 비디오 스트림을 디코딩하기 위한 실시간 스케줄링 기법[3]에서 각 태스크마다 WCET가 적용된 바 있다. 그러나 MPEG-2 비디오 스트림을 처리하는 멀티미디어 태스크의 경우 프레임의 크기에 따라 실제 디코딩 시간이 달라지기 때문에 WCET를 설정하는 것 자체도 어려운 문제이며 WCET보다는 평균 실행 시간을 고려하여 스케줄링하는 것이 바람직하다. 멀티미디어 태스크의 평균 실행 시간과 주기를 설정하고 CPU의 대역폭을 예약하여 태스크들을 스케줄링하는 CBR(Constant Bandwidth Reservation) 기법[6, 7]이 제시되었으나 MPEG 비디오 스트림의 특성이 반영되지 않았다.

본 논문에서는 MPEG 비디오 스트림을 디코딩하기 위하여 평균 실행 시간과 주기가 설정된 멀티미디어 태스크들을 효율적으로 스케줄링할 수 있는 CPU 대역폭 할당 기법이 제시된다. 스케줄링 기능을 전담하는 서버는 일정한 CPU 대역폭을 태스크의 주기와 평균 실행 시간을 고려하여 태스크별로 할당한다. 그리고 MPEG-2 비디오 스트림의 프레임 특성을 고려하여 설정된 우선순위를 멀티미디어 태스크에 적용하는 한편 CPU 대역폭을 동적으로 조정하는 태스크 스케줄링 기법을 수행한다.

제시된 기법으로 MPEG-2 비디오 스트림을 디코딩하는 멀티미디어 태스크들을 스케줄링하는 경우 각 태스크의 종료시한 이후 태스크가 실제 종료될 때까지 소요된 지연 시간과 디코딩 시간을 감소시킴으로써 종료시한 이후 실행이 완료되는 태스크의 수를 최소화할 수 있다. 제시된 기법의 성능은 시뮬레이션을 이용한 실험 결과에서 분석된다. 본 논문의 구성은 다음과 같다. 2장에서는 MPEG-2 디코딩을 위한 멀티미디어 태스크 모델과 CPU 대역폭 할당 기법이 정의되며 3장에서는 멀티미디어 태스크를 스케줄링하는 알고리즘과 예제가 기술된다. 4장에서는 제시된 기법의 성능이 시뮬레이션 결과를 토대로 분석되며 5장에서는 결론이 기술된다.

2. 멀티미디어 태스크를 위한 CPU 대역폭 할당

2.1 MPEG-2프레임을 위한 우선순위

MPEG 압축 알고리즘은 이미지의 시간과 공간의 중복성을 최대한 이용하여 동영상 데이터를 압축하며 세계적인 표준으로 활용되고 있다[1]. 압축이 수행된 MPEG-1 또는 MPEG-2 비디오 스트림은 계층 구조상 일정 개수의 픽처(picture) 또는 프레임들로 구성된 GoP 층이 형성된다[5]. 이 GoP 층은 I, P, B, D 프레임 등의 픽처들이 저장되며 임의로 접근할 수 있는 단위가 된다. 실제로 MPEG-1과 MPEG-2 비디오는 한 GoP를 구성하는 프레임 수에 의해 QoS 수준이 결정된다. 이는 MPEG 표준의 압축 변수인 N과 M에 의하여 달라지며 N은 한 GOP내의 프레임 수를 의미하고 M은 I 또는 P 프레임이 나타나는 주기를 의미한다. 예를 들어 N=15, M=3일 경우 GoP 내에서 프레임들의 배열은 IBBP BBPBBPBBPBB 순서가 된다. 본 논문에서는 한 개의 태스크가 한 개의 MPEG 비디오 스트림을 디코딩하게 되며 태스크는 일정한 주기와 평균 실행 시간을 갖는 것으로 가정한다. 그리고 태스크의 주기마다 해당 태스크의 인스턴스(instance)가 생성되는 것으로 가정하며 태스크 인스턴스는 해당 태스크와 동일한 평균 실행시간을 가지게 된다.

본 논문에서는 각 프레임을 디코딩하는 멀티미디어 태스크 인스턴스마다 우선 순위가 설정된다. MPEG 압축 기법의 특성상 GoP 내의 기준 프레임인 I 프레임을 먼저 디코딩하지 않고서는 P 프레임이나 B 프레임들을 디코딩할 수 없다[5, 13]. 결국 GoP 내의 모든 B 프레임들을 디코딩하기 위해서는 I 프레임과 P 프레임을 먼저 디코딩해야 한다. 본 논문에서 I 프레임을 디코딩하는 태스크 인스턴스의 우선순위는 1로 설정되며 P 프레임을 처리하는 태스크 인스턴스와 B 프레임을 처리하는 태스크 인스턴스의 우선순위는 2와 3으로 각각 설정된다. 우선순위 번호가 작을수록 논리적인 우선순위가 높게 설정됨을 의미하며 멀티미디어 태스크 인스턴스들은 우선순위별로 구성된 큐(queue)에서 각각 대기하게 된다. 서버가 멀티미디어 태스크 인스턴스를 스케줄링할 경우 우선순위 순서에 따라 우선순위가 가장 높은 큐부터 태스크 인스턴스를 찾게 된다. 이때 동일한 우선순위의 태스크 인스턴스들이 여러 개 있는 경우에는 EDF 방법을 적용하여 종료시한이 가장 빠른 태스크 인스턴스를 선택하게 된다.

따라서 I 프레임을 디코딩하는 태스크 인스턴스가 P 프레임을 디코딩하는 태스크 인스턴스보다 먼저 스케줄링되며 B 프레임을 디코딩하는 태스크 인스턴스보다는 P 프레임을 디코딩하는 태스크 인스턴스가 먼저 스케줄링될 수 있는 기회를 갖게 된다. 단 서버가 선택한 멀티미디어 태스크 인스턴스가 실행되는 동안 우선순위가 더 높은 다른 멀티미디어 태스크 인스턴스에 의하여 선점(preemption)되는 경우는 허

용되지 않는다.

2.2 평균 실행시간 설정

평균 실행시간을 기반으로 스케줄링되는 멀티미디어 태스크의 특성을 감안할 경우 MPEG 비디오 스트림은 프레임의 크기가 종류별로 상이하기 때문에 태스크마다 동일한 평균 실행 시간을 설정하는 것이 사실상 어렵다. 프레임의 크기가 가장 큰 I 프레임에 대한 디코딩 시간을 평균 실행 시간으로 설정할 경우 I 프레임보다 작은 P 프레임과 B 프레임을 처리할 때 CPU 대역폭이 낭비되는 결과가 발생한다. 또한 I 프레임보다는 작고 B 프레임보다는 큰 P 프레임의 디코딩 시간을 평균 실행 시간으로 설정할 경우 B 프레임을 처리할 때는 대역폭의 낭비가 발생되는 반면 I 프레임을 처리할 때는 대역폭이 부족한 현상이 발생하게 된다. 따라서 본 논문에서는 특정 프레임에 대한 평균 실행 시간을 직접 반영하는 대신 한 GoP에 대한 평균 실행 시간을 설정한 후 이 시간을 GoP를 구성하는 프레임의 수(예, N= 15)로 나눈 값을 멀티미디어 태스크의 평균 실행시간으로 설정함으로써 평균 실행시간의 변동성을 다소 감소시키고자 하였다. 한 GoP를 구성하는 프레임 수 N이 15인 경우 15개의 태스크 인스턴스들이 동일한 평균 실행 시간을 갖게 된다. 이러한 방법으로 설정된 평균 실행 시간은 GoP 내의 프레임 크기와 수에 따라 변동성이 있으나 I 프레임과 P 프레임을 디코딩하기 위한 평균 실행 시간보다는 작게 되며 B 프레임을 디코딩하기 위한 평균 실행 시간보다는 큰 값을 갖게 된다.

2.3 CPU 대역폭 할당

본 논문에서 서로 다른 n개의 멀티미디어 태스크들이 시스템에 존재할 경우 임의의 멀티미디어 태스크 T_i는 MPEG-2 비디오 스트림을 디코딩하기 위한 평균 실행시간 $Mean(T_i)$와 일정한 주기 $Period(T_i)$를 갖는 것으로 가정한다. 스케줄링 기능을 수행하는 멀티미디어 서버의 주기는 n개의 멀티미디어 태스크들 중 가장 주기가 작은 것으로 결정된다. 따라서 멀티미디어 서버도 일정한 주기를 갖는 실시간 태스크로 수행되며 이 서버는 T_s로 표기된다. 멀티미디어 서버 T_s의 주기는 $Period(T_s)$로 표기되며 T_s가 사용할 수 있는 CPU 실행 시간이 $Comp_time(T_s)$로 정의된다. $Comp_time(T_s)$를 해당 주기 $Period(T_s)$로 나눈 값이 실제 서버의 CPU 대역폭을 의미하며 이는 $Bandwidth(T_s)$로 정의된다.

$$Bandwidth(T_s) = Comp\_time(T_s) / Period(T_s) \quad (1)$$

n개의 멀티미디어 태스크들에 대한 $Comp_time(T_s)$는 다음과 같이 계산된다.

$$Comp\_time(T_s) = \sum_{i=1}^{n} Budget(T_i) \quad (2)$$

수식 2에서 $Budget(T_i)$는 서버 T_s의 주기 내에서 임의의 멀티미디어 태스크 T_i에 할당된 CPU 시간을 의미한다. 멀티미디어 태스크 T_i는 자신의 주기 $Period(T_i)$ 동안 평균 실행시간 $Mean(T_i)$가 설정되지만 서버의 주기 $Period(T_s)$가 $Period(T_i)$보다 작거나 같기 때문에 $Period(T_s)$ 동안에는 $Mean(T_i)$ 대신 $Period(T_s)$를 $Period(T_i)$로 나눈 값에 $Mean(T_i)$를 곱한 시간인 $Budget(T_i)$가 할당된다.

$$Budget(T_i) = Mean(T_i)\frac{Period(T_s)}{Period(T_i)} \quad (3)$$

그리고 T_i의 CPU 대역폭 $Bandwidth(T_i)$는 다음과 같이 기술된다.

$$Bandwidth(T_i) = Budget(T_i) / Period(T_s) \quad (4)$$

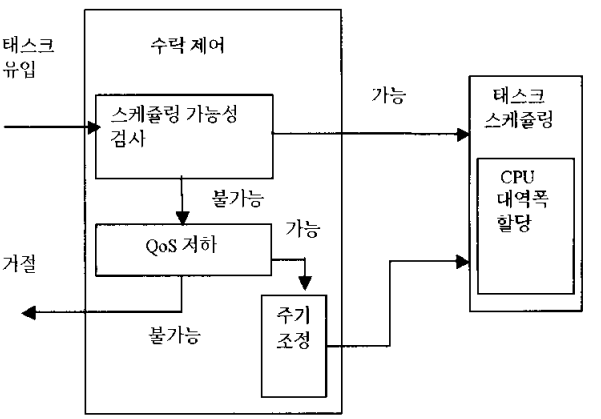
수식 1에서 서버 T_s의 $Bandwidth(T_s)$는 n 개 태스크들의 대역폭의 합과 같으며 다음과 같이 표현된다.

$$Bandwidth(T_s) = \sum_{i=1}^{n} Bandwidth(T_i) \quad (5)$$

2.4 수락 제어

서버 T_s의 대역폭은 각 멀티미디어 태스크의 평균 실행시간과 주기를 고려하여 각 태스크에 할당된다. 본 논문에서는 이론상 CPU 자원의 활용도가 69%인(평균적인 경우를 분석할 시 88%까지 증가될 수 있음[10, 12]) RM 스케줄링 기법보다 활용도를 100%까지 증가시킬 수 있는 EDF 스케줄링 기법[10, 11]을 채택함으로써 자원의 활용도를 극대화하고자 하였다. EDF 스케줄링 원칙에 따라 각 태스크의 실행 시간을 주기로 나눈 값들의 합(이하 자원 활용률 U로 정의함)이 U의 low upper bound인 U_{lub}(= 1.0)보다 작거나 같을 때만 태스크들이 스케줄링될 수 있는 것으로 간주한다[10].

$$U = \sum_{i=1}^{n} \frac{Mean(T_i)}{Period(T_i)} \le U_{lub} = 1.0 \quad (6)$$



(그림 1) 수락 제어와 태스크 스케줄링 관계

주기와 평균 시간이 설정된 멀티미디어 태스크가 시스템에 유입될 때 '수락 제어' 과정에서 수식 6에 의하여 스케쥴링 가능성이 검사된다. 스케쥴링이 가능한 태스크는 CPU 대역폭을 할당받고 서버에 의해 스케쥴링된다. 수식 6에 의하여 스케쥴링이 일단 불가능한 것으로 판명된 태스크는 수락 제어 정책에 따라 현재 QoS를 유지하기 위하여 실행이 거절될 수 있다. 한편 새로운 멀티미디어 태스크가 시스템에 유입되어 U가 1.0을 초과할 경우 QoS를 저하시킴으로써 새로운 멀티미디어 태스크의 실행이 허용될 수 있다. 이때 기존 n 개의 태스크들과 새로 유입된 태스크의 주기는 모두 증가된다. 1.0을 초과한 U를 U'이라 정의하고 각 태스크의 현재 주기 Period(T_i)에 U'을 곱한 값을 새로운 주기 Period'(T_i)로 설정하게 된다.

$$\text{Period}'(T_i) = \text{Period}(T_i)*U' \qquad (7)$$

각 태스크의 Mean(T_i)는 변동이 없으며 주기만 증가하기 때문에 U는 결국 1.0으로 조정되며 수식 6의 조건을 만족하게 되어 모든 멀티미디어 태스크는 주기가 증가된 상태에서 스케쥴링이 가능하게 된다.

3. 우선순위화된 스케쥴링 기법

3.1 스케쥴링 알고리즘

멀티미디어 태스크들 중 가장 작은 주기를 갖는 멀티미디어 서버 T_s는 매 주기마다 그림 2에서 기술된 알고리즘에 따라 태스크들을 스케쥴링하게 된다. 그림 2에 기술된 스케쥴링 알고리즘에서 사용되는 용어와 변수들은 다음과 같이 정의된다.

Queue(i): 우선순위가 i(1 이상 3 이하의 정수)인 태스크 인스턴스들의 큐(queue)
Length(i): 우선순위가 i인 큐에 있는 태스크 인스턴스의 수
Deadline(T_i): T_i의 종료시한
Min_Deadline: 검색된 태스크 인스턴스들 중 가장 빠른 종료시한
T_m: 검색된 태스크 인스턴스들 중 우선순위가 가장 높고 종료시한이 가장 빠른 태스크 인스턴스
Executes(T_m): 서버가 T_m을 실행함.
Exec_time(T_m): T_m이 실행된 시간

멀티미디어 태스크들을 위한 서버의 주기내 허용된 CPU 실행시간 Comp_time(T_s)가 0보다 큰 경우 우선순위가 1인 Queue(1)(I 프레임을 디코딩하는 태스크 인스턴스들의 큐)을 대상으로 Length(1) 개의 태스크 인스턴스들 중 종료시한이 가장 빠른 태스크 인스턴스 T_m을 검색한다. Min_Deadline의 초기치는 이론상 무한대 값이 설정되나 검색 과정에서 실제 태스크 인스턴스의 종료시한으로 변경되며 해당 큐에서 항상 가장 빠른 종료시한으로 변경된다. 만일 Queue(1)에 스케쥴링 가능한 태스크 인스턴스들이 없으면 서버는 우선순위가 낮은 Queue(2)와 Queue(3)을 차례로 검색하여 T_m을 설정하게 된다. 서버는 T_m을 설정한 후 Queue(i)를 검색하는 for 순환문을 벗어나게 된다. 이 과정은 그림 2의 알고리즘에서 Breaks로 표현된다.

```
while (Comp_time(Ts) > 0 )
 begin
  for Queue(i), i = 1, 3
   begin
    for j = 1 to Length(i)   /* Length(i) is the number of tasks
        in Queue(i) */
     if Deadline(Tj) < Min_Deadline
      begin
       Min_Deadline = Deadline(Tj);
       Tm = Tj;
      end
    if Tm exists
     Breaks;
    end

  if Tm exists
   begin
    Executes(Tm);
    Comp_time(Ts) = Comp_time(Ts)      Exec_time(Tm);
  end
 end
```

(그림 2) 멀티미디어 서버의 스케쥴링 알고리즘

T_m이 실제로 존재하는 경우 서버는 T_m의 실행을 시작하게 된다(알고리즘에서 Executes(T_m)으로 기술됨). 이때 서버는 T_m을 Budget(T_m) 동안만 실행하는 것이 아니고 해당 프레임을 디코딩하는 시간만큼 필요한 대역폭을 사실상 허용하게 된다. I 프레임이나 P 프레임을 디코딩하는 경우에는 Budget(T_m) 보다 큰 대역폭이 필요하므로 서버는 전체 멀티미디어 태스크들을 위한 서버의 실행시간 Comp_time(T_s) 범위 내에서 다른 멀티미디어 태스크에 할당된 대역폭의 사용을 허용한다. 따라서 T_m은 서버의 주기에서 최대 Comp_time(T_s)만큼의 CPU 대역폭을 사용할 수 있으며 이를 사용하고도 해당 프레임의 디코딩이 완료되지 않은 경우에는 서버의 다음 주기에서 다시 실행될 기회를 갖게 된다. 선택된 태스크 인스턴스 T_m의 해당 프레임의 디코딩이 완료된 경우 또는 서버의 Comp_time(T_s)가 모두 사용된 경우 서버는 이제까지 실행된 시간 Exec_time(T_m)을 Comp_time(T_s)에서 감소시킨다. 이후 감소된 Comp_time(T_s)가 0보다 큰 경우에는 그림 2의 while 순환문에서 새로운 태스크 인스턴스 T_m을 선택하는 과정이 다시 수행된다. 만일 Comp_time(T_s)가 0이 되는 경우에는 새로운 서버 인스턴스가 도착하여 0보다 큰 Comp_time(T_s)를 다시 확보할 때까지 태스크들의 스케쥴링이 보류된다.

3.2 스케쥴링 예제

세 개의 멀티미디어 태스크 T_1, T_2, T_3의 평균 실행시간과 주기는 다음과 같이 설정되며 단위 시간은 ms(1/1000초)로 가정한다.

$Mean(T_1) = 12, Period(T_1) = 30$
$Mean(T_2) = 12, Period(T_2) = 40$
$Mean(T_3) = 12, Period(T_3) = 60$

세 멀티미디어 태스크의 GoP를 구성하는 프레임의 수 N이 15이며 프레임의 순서는 IBBPBBPBBPBBPBB로 가정한다. 그리고 15 개의 프레임을 디코딩하는 평균 시간을 180ms로 가정하여 $Mean(T_1)$, $Mean(T_2)$, $Mean(T_3)$는 모두 12(= 180/15)ms로 설정된다. 본 논문에서 태스크들 간의 문맥 교환(context switching) 오버헤드는 없는 것을 가정한다. 그리고 수식 6에서 정의된 자원 활용율 U는 다음과 같이 1.0이하이므로 EDF 방법에 따라 T_1, T_2, T_3의 스케줄링이 가능하다[10].

$$U = \sum_{i=1}^{3} \frac{Mean(T_i)}{Period(T_i)} = \frac{12}{30} + \frac{12}{40} + \frac{12}{60} = 0.4 + 0.3 + 0.2 = 0.9 \leq 1.0 \quad (수식\ 6)$$

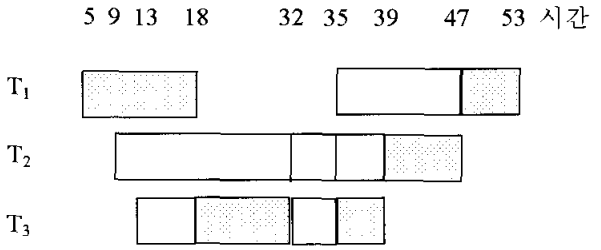
세 태스크들 중 주기가 가장 작은 T_1의 주기가 서버 T_s의 주기로 설정되며 이 주기 내에 각 태스크가 사용할 수 있는 CPU 시간은 수식 3에 의하여 다음과 같이 계산된다.

$$Budget(T_1) = Mean(T_1)\frac{Period(T_s)}{Period(T_1)} = 12 \times \frac{30}{30} = 12$$

$$Budget(T_2) = Mean(T_2)\frac{Period(T_s)}{Period(T_1)} = 12 \times \frac{30}{40} = 9$$

$$Budget(T_3) = Mean(T_3)\frac{Period(T_s)}{Period(T_1)} = 12 \times \frac{30}{60} = 6$$

서버 T_s의 $Comp_time(T_s)$는 수식 2에 의하여 27ms(= $Budget(T_1) + Budget(T_2) + Budget(T_3) = 12 + 9 + 6$)가 된다. 그림 2에서 기술된 멀티미디어 서버의 스케줄링 알고리즘에 따라 T_1, T_2, T_3 태스크들이 스케줄링되는 순서가 그림 3에서 기술된다. T_1, T_2, T_3 의 첫번째 태스크 인스턴스들이 시스템에 처음 도착한 시간은 각각 5, 9, 13ms로 가정한다. 이후에는 각 태스크의 주기에 따라 새로운 태스크 인스턴스들이 도착하게 된다. 시간 5ms(이하 단위 시간은 생략함)에서 서버는 최초로 T_1을 실행하게 된다. T_1은 P 프레임을 디코딩하는 것으로 가정하며 실제 실행 시간이 $Budget(T_1)$(= 12)보다 큰 13ms로 설정된다. 각 태스크 인스턴스의 실행 중인 시간은 그림 2에서 점이 있는 상자로 표시되며 해당 큐에서 대기중인 시간은 점이 없는 상자로 표시된다. 시간 9에서 도착한 T_2는 실제 실행시간이 $Budget(T_2)$(= 9)보다 1이 작은 8이며 B 프레임을 디코딩하는 것으로 가정한다. 그리고 서버는 현재 T_1이 실행 중이므로 T_2를 우선순위가 3인 Queue(3)에서 대기시킨다. 시간 13에서 도착한 T_3는 실제 실행 시간이 $Budget(T_3)$(= 6)보다 큰 18이며 I 프레임을 디코딩하는 태스크 인스턴스로 가정한다. 따라서 T_3는 우선순위가 1인 큐 Queue(1)에서 대기하게 된다. 실행 중인 T_1보다 T_3의 우선순위가 더 높다 하더라도 제시된 스케줄링 기법에서는 태스크들 간의 CPU 선점이 허용되지 않는다. 시간 18에서 T_1이 P 프레임의 디코딩을 완료하면 $Comp_time(T_s)$는 14(= 27 - $Exec\_time(T_1)$ = 27 - 13)가 된다. 그리고 서버는 Queue(1)에서 대기중인 T_3를 실행하게 된다. 즉 Queue(1)과 Queue(3)에서 각각 대기 중인 T_3와 T_2 중 우선순위가 더 높은 T_3가 먼저 실행되는 것이다.



(그림 3) T_1, T_2, T_3 스케줄링 순서

T_3의 실제 실행시간이 18이기 때문에 서버의 남은 $Comp\_time(T_s)$(=14)동안 T_3가 실행된 후 시간 32에서 $Comp_time(T_s)$는 0이 된다. 따라서 시간 32부터 새로운 서버 인스턴스가 도착하는 시간 35까지 서버는 멀티미디어 태스크들을 스케줄링할 수 없게 된다. 시간 35에서 새로운 서버 인스턴스가 도착하고 $Comp_time(T_s)$는 다시 27로 설정된다. 시간 35부터 새로운 서버 인스턴스의 $Comp_time(T_s)$를 활용하여 T_3는 시간 39에서 실행이 종료되며 $Comp_time(T_s)$는 23(= 27 - $Exec\_time(T_3)$ = 27 - 4)이 된다. 이때 $Exec_time(T_3)$는 시간 35이후 T_3가 4 동안 실행된 시간을 의미한다. 한편 시간 35에서 도착한 T_1의 두번째 태스크 인스턴스는 B 프레임을 디코딩하며 실제 실행시간이 6인 것으로 가정한다. 도착 당시 T_3가 실행 중이었기 때문에 T_1의 두번째 태스크 인스턴스는 우선순위가 3인 Queue(3)에서 대기하게 된다.

T_3가 종료된 시간 39에서 서버는 우선순위가 3인 큐 Queue(3)에서 대기 중인 T_1, T_2 태스크 인스턴스들 중 종료시한이 더 빠른 T_2를 선택하여 실행하게 된다. T_1의 종료시한은 65(=35 + $Period(T_1)$ = 35 + 30)이며 T_2의 종료시한은 49(=9 + $Period(T_2)$ = 9 + 40)이다. 즉 우선순위가 동일한 태스크 인스턴스들 중에서는 EDF 원칙에 따라 종료시한이 가장 빠른 태스크 인스턴스가 선택되는 것이다. T_2가 시간 47에서 종료되면 $Comp_time(T_s)$는 15(=23 - 8)가 되며 Queue(3)에 있는 T_1이 서버에 의해 시간 53까지 실행된다. 이후의 스케

쥴링 과정은 본 논문에서 생략한다.

4. 성능 분석

4.1 시뮬레이션 환경

본 논문에서 제시된 스케쥴링 기법의 성능은 고정 대역폭 스케쥴링(CBS; Constant Bandwidth Scheduling)[6, 7]을 기반으로 하는 스케쥴링 기법과 비교된다. 본래 CBS 기법은 멀티미디어 태스크의 평균 실행 시간과 주기를 기반으로 모든 멀티미디어 태스크마다 별도의 서버가 설정되며 평균 시간을 초과하는 CPU 대역폭이 요청된 경우 다음 주기의 서버에서 이를 수용하는 스케쥴링 방법이다. 실제 멀티미디어 태스크의 실행 시간이 예상 평균 시간보다 증가한 경우 다른 멀티미디어 태스크의 실행 시간을 이용할 수 없으며 MPEG 비디오 스트림의 프레임 특성이 제대로 반영되지 않고 있다. 우선순위를 태스크에 적용하지 않은 CBS 기반 스케쥴링 기법을 본 논문에서는 NP(Non-Prioritized) 스케쥴링 기법으로 표기하며 우선순위를 태스크에 적용한 본 논문의 스케쥴링 기법은 P(Prioritized) 스케쥴링 기법으로 표기한다. 시뮬레이션에서 NP 기법도 P 기법과 동일하게 태스크들 중 가장 작은 주기가 서버의 주기로 설정되며 태스크마다 서버의 주기 내에서 CPU를 사용할 수 있는 시간이 할당된다. 그러나 NP 기법에서는 멀티미디어 태스크의 우선순위가 이용되지 않으며 모든 멀티미디어 태스크는 서버의 주기마다 할당된 CPU 대역폭만을 사용하게 된다.

3.2 스케쥴링 예제에서 기술된 T_1, T_2, T_3 외에 T_4 태스크가 시뮬레이션에서 포함된다. T_4의 평균 실행 시간과 주기는 다음과 같이 설정되며 단위 시간은 ms이다.

$$Mean(T_4) = 12,\ Period(M_4) = 120$$

T_1, T_2, T_3, T_4 태스크들은 영화 "Mission Impossible"의 20분간 분량을 MPEG-1 인코더로 압축한 파일 mission.mpg를 디코딩하며 이 파일의 GoP는 15개의 프레임으로 구성된 IBBPBBPBBPBBPBB 패턴을 갖는다. 실제 I, P, B 프레임의 평균 크기는 16657, 8183, 4399 바이트이며 한 GoP에 대한 평균 디코딩 시간은 180ms로 가정한다. MPEG 스트림의 경우 프레임 간의 크기 비율은 스트림의 종류에 따라 다르다[13]. 움직임이 적절한 영화의 경우 I, P, B 프레임 간의 평균 크기 비율이 약 8:2:1이나 본 논문에서 선택한 "Mission Impossible"의 경우 평균 크기 비율이 3.79:1.86:1이다. 이 비율을 고려한 GoP 내 각 프레임의 평균 디코딩 시간은 다음과 같이 설정된다.

I 프레임의 평균 디코딩 시간: 32.130ms
P 프레임의 평균 디코딩 시간: 15.770ms
B 프레임의 평균 디코딩 시간: 8.479ms.

시뮬레이션에서 각 프레임의 실제 데이터 크기에 따라 태스크 인스턴스의 실행 시간이 결정되었다. Pentium-IV 2.0 GHz CPU와 512MB의 메모리로 구성된 컴퓨터에서 Linux 운영체제를 기반으로 C 언어와 CSIM[14] 소프트웨어로 구현된 시뮬레이터가 사용되었다.

4.2 결과 분석

4.2.1 디코딩이 완료된 태스크 수

시뮬레이션에서 자원 이용률 U는 T_1, T_2, T_3, T_4 태스크들의 주기와 평균 실행시간을 수식 6과 같이 적용하여 값이 0.5부터 1.0까지 설정되며 각 U에 대하여 구성된 태스크들은 표 1과 같다. 예를 들어 T_2와 T_3 태스크들을 스케쥴링할 경우 U는 0.5가 된다.

$$U = \sum_{i=1}^{2} \frac{Mean(Ti)}{Period(Ti)} = \frac{12}{40} + \frac{12}{60} = 0.3 + 0.2 = 0.5$$

〈표 1〉 U에 대한 태스크 구성

U	0.5	0.6	0.7	0.8	0.9	1.0
태스크	T_2, T_3	T_2, T_3, T_4	T_1, T_3, T_4	T_1, T_2, T_4	T_1, T_2, T_3	T_1, T_2, T_3, T_4

U를 0.5에서 1.0까지 0.1씩 증가시킨 상태에서 12000ms 동안 P 기법과 NP 기법에서 생성된 태스크들의 수는 동일하며 N_{gen}으로 정의한다. 그리고 생성된 태스크들 중 디코딩이 완료된 태스크들의 수를 N_{comp}를 비교한 결과는 표 2에 기술되어 있다. U가 증가함에 따라 시스템에 대한 작업 부하도 이에 비례하여 커지는 것을 의미한다. U가 0.9 이하일 때 P 기법의 N_{comp}는 N_{gen}과 거의 같은 결과가 나타남으로써 생성된 모든 태스크들의 디코딩이 완료된 반면 NP 기법의 N_{comp}는 P 기법보다 다소 작은 결과를 보이고 있다. U가 1.0일 때 P 기법에서 생성된 태스크들 1001개 중 디코딩이 완료되지 못한 태스크들이 17개로 다소 증가하였으며 NP 기법에서는 이러한 태스크들의 수가 더 많아지는 결과가 나타났다.

〈표 2〉 U가 1.0 이하 일때 디코딩이 완료된 태스크 수

U / N_{comp}	0.5	0.6	0.7	0.8	0.9	1.0
P 기법	500	600	700	800	899	984
NP 기법	490	589	691	790	887	924
N_{gen}	500	600	701	801	901	1001

〈표 3〉 U가 1.0을 초과한 경우 디코딩이 완료된 태스크 수

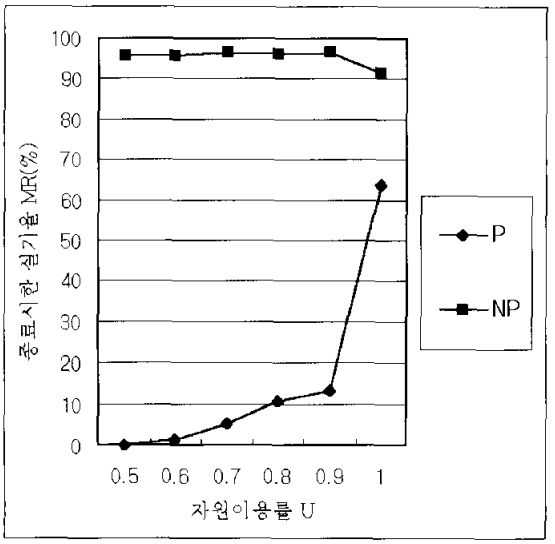
U / N_{comp}	1.1	1.2	1.3
P 기법	984	988	984
NP 기법	962	979	983
N_{gen}	1001	1002	1001

제시된 P 기법은 NP 기법보다 생성된 태스크들 중 디코딩을 완료하는 태스크들의 수가 더 큰 것으로 나타났다. 그리고 U를 1.1부터 1.3까지 증가시킨 상태에서도 두 기법의 N_{comp}를 비교하였으며 이 결과는 표 3에 기술되어 있다. U가 1.0을 초과하는 경우는 각 태스크의 주기에 1.0을 초과한 U를 곱함으로써 모든 태스크의 주기가 동일한 비율로 증가하게 된다. 평균 실행시간은 변동이 없으며 태스크의 주기만 증가시킴으로써 QoS를 저하시키는 방법을 본 논문의 2.4절에서 기술한 바 있다. U가 1.1인 경우는 U가 1.0 일때 구성된 태스크 T_1, T_2, T_3, T_4 외에 T_4가 추가로 구성된다. 즉 시스템에 도착하는 시간이 다른 두 개의 T_4가 다른 태스크들과 스케쥴링된다. 동일한 방법으로 U가 1.2인 경우는 T_3가 추가로 구성되며 U가 1.3인 경우는 T_2가 추가로 구성되었다. 1.0을 초과한 자원 이용률 U는 구성된 모든 태스크의 주기를 증가시킴으로써 자원 이용률 U는 다시 1.0으로 조정된 상태에서 스케쥴링된다. <표 3>의 결과는 U의 값이 모두 1.0으로 조정된 상태에서 측정된 결과이기 때문에 <표 2>에서 U가 1.0일 때 측정된 결과와 크게 다르지 않은 것으로 분석되었다.

4.2.2 종료시한 실기율

제시된 스케쥴링 기법의 목적은 종료시한이 경과된 이후에 실제로 종료되는 태스크 인스턴스들의 수를 최소화함으로써 멀티미디어 시스템의 실시간적 성능을 향상시키는 것이다. 디코딩이 완료된 태스크 인스턴스들 중 종료시한이 경과된 후 종료된 태스크 인스턴스들의 수를 N_{miss}라 할 때 생성된 태스크 인스턴스들 N_{gen}에 대한 종료시한 실기율 MR(Miss Ratio)은 다음과 같이 정의된다.

$$MR = \frac{N_{miss}}{N_{gen}} \quad (8)$$



(그림 4) 종료시한 실기율 MR(%)

U를 0.5에서 1.0까지 증가시킨 상태에서 P 기법과 NP 기법의 MR을 비교한 결과는 (그림 4)에 나타나 있다. P 기법의 경우 U의 값이 0.5에서 0.9까지 증가함에 따라 MR은 0.2%에서 13.24%로 증가하였으며 매우 낮은 실기율이 나타났다. 그리고 U의 값이 1.0인 경우 MR은 63.54%로 측정되었다. 반면 NP 기법의 MR은 항상 90%가 넘는 매우 높은 실기율이 나타났다. 이는 P 기법에 비하여 NP 기법의 태스크 인스턴스들은 대기 시간이 길고 한 프레임을 디코딩하는 시간 자체도 증가하기 때문이다.

4.2.3 지연 시간

종료시한이 경과된 이후 디코딩이 완료된 태스크 인스턴스의 경우 실제 실행이 완료된 시간과 종료시한의 차이를 지연 시간이라 정의한다. 두 기법에서 측정된 평균 지연 시간은 표 4에서 비교되고 있다. NP 기법의 평균 지연 시간이 P 기법의 평균 지연 시간보다 항상 큰 것으로 측정되었다. 특히 U가 1.0인 경우 P 기법의 평균 지연 시간이 107.76 ms 정도인 반면 NP 기법의 경우에는 419.40 ms로 측정됨으로써 큰 차이를 보이고 있다. 이러한 이유는 NP 기법은 태스크 인스턴스마다 서버의 주기내에서 할당된 대역폭만을 사용하므로 I 프레임이나 P 프레임과 같이 평균 실행 시간보다 큰 프레임을 디코딩할 경우 소요되는 시간이 커지기 때문이다. 따라서 종료시한이 상당히 경과된 후 디코딩이 완료되는 태스크 인스턴스들의 수가 증가하게 된다. 재생하고자 하는 프레임 순서에 따라 다른 태스크 인스턴스들의 디코딩 시작 시간 자체도 늦어지게 하는 현상을 자주 유발하므로 NP 기법에서 측정된 평균 지연시간은 제시된 P 기법에 비하여 항상 큰 것으로 분석되었다.

〈표 4〉 평균 지연 시간

(단위: ms)

기법 \ U	0.5	0.6	0.7	0.8	0.9	1.0
P	3.4	2.84	3.42	9.80	14.83	107.76
NP	85.07	88.17	86.58	85.79	84.67	419.40

U가 1.0인 경우 12000ms까지 두 기법에 의하여 각 프레임이 디코딩 과정동안 소요된 평균 시간이 표 5에 나타나 있다. 이 표에서 디코딩 시간은 각 태스크 인스턴스가 해당 프레임에 대한 디코딩을 일단 시작한 이후 완료될 때까지의 시간을 의미하기 때문에 도중에 큐에서 대기 상태로 머물렀던 시간도 모두 포함이 된다. P 기법은 프레임의 크기와 특성을 반영한 우선순위를 사용했기 때문에 NP 기법에 비하여 각 프레임의 평균 디코딩 시간이 작은 것으로 나타났다.

〈표 5〉 평균 디코딩 시간

(단위: ms)

기법 \ 프레임	I	P	B
P	23.97	18.67	8.31
NP	100.95	66.65	28.89

5. 결 론

MPEG-2 비디오 스트림을 위한 멀티미디어 태스크들을 우선순위에 의하여 효율적으로 스케쥴링할 수 있는 기법이 본 논문에서 제시되었다. 태스크들 중 가장 작은 주기를 서버의 주기로 정하며 서버의 CPU 대역폭은 각 멀티미디어 태스크의 주기와 평균 실행 시간을 고려하여 각 멀티미디어 태스크마다 CPU 대역폭이 할당된다. MPEG 기법에서 사용되는 각 프레임의 중요도와 특성을 반영한 우선순위를 멀티미디어 태스크에 설정하였으며 서버의 CPU 대역폭 범위 내에서 우선순위가 높은 태스크일수록 CPU 대역폭을 충분히 사용할 수 있도록 스케쥴링되고 있다. 제시된 스케쥴링 기법은 우선순위를 사용하지 않는 기법에 비하여 동일한 시간 동안 더 많은 수의 프레임들을 디코딩할 수 있으며 종료시한이 경과된 후 디코딩이 완료되는 태스크의 수도 감소시킬 수 있다. 또한 태스크의 평균 지연 시간과 프레임에 대한 평균 디코딩 시간을 감소시킴으로써 멀티미디어 시스템의 QoS와 실시간적 성능을 향상시킬 수 있는 것으로 분석되었다. 제시된 스케쥴링 기법은 MPEG-2 비디오 스트림을 디코딩하는 멀티미디어 시스템의 성능 향상을 위하여 활용될 수 있을 것으로 기대된다.

참 고 문 헌

[1] ISO/IEC., '13818-2: Information technology - Generic coding of moving pictures and associated audio information Part 2: Video', 1996.

[2] P. Altenbernd, L. Burchard and F. Stappert, "Worst-case execution times analysis of MPEG-2 decoding", Proc. of 12th Euromicro Conference on Real Time Systems, Stockholm, 2000.

[3] M. Ditze and P. Altenbernd, "A method for real-time scheduling and admission control of MPEG-2 streams", Proc. of 7th Australasian Conference on Parallel and Real-Time Systems (PART2000), Nov. 2000.

[4] Baiceanu, V., et al., "Multimedia applications require adaptive scheduling", In Workshop on Resource Allocation Problems in Multimedia Systems, Dec. 1996.

[5] Mitchell, J. L., et al., 'MPEG video compression standard', Chapman & Hall, 1996.

[6] L. Abeni and et al., "Integrating multimedia applications in hard real-time systems", Proc. of IEEE Real-Time Systems Symposium, Dec. 1998.

[7] L. Abeni and G. Buttasso, "Adaptive bandwidth reservation for multimedia computing", Proc. of IEEE Conf. on RTCSA, Dec. 1999.

[8] O. Gonzales and et al., "Incorporation of multi-media capabilities in distributed real-time applications", Workshop on Databases: Active and Real-time, 1996.

[9] Kaneko and et al., "Integrated scheduling of multimedia and hard real-time tasks", Proc. of IEEE Real-Time Systems Symposium, Dec. 1996.

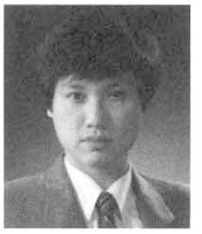
[10] C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," Journal of the ACM, Vol. 20, No. 1, 1973.

[11] K. Jeffay, "Scheduling sporadic tasks with shared resources in hard real-time systems," Proc. of IEEE Real-Time Systems Symposium, Dec. 1992.

[12] L. Sha and et al., "Priority inheritance protocols: an approach to real-time synchronization," IEEE Transactions on Computers, Vol. 39, No. 9, 1990.

[13] John Watkinson, 'MPEG Handbook', Butterworth-Heinemann, Oct. 2001.

[14] Mesquite Software, 'CSIM18 Simulation Engine(c version)', 1999.



김 진 환

e-mail :kimjh@hansung.ac.kr

1986년 서울대학교 컴퓨터공학과 졸업

1988년 서울대학교 대학원 컴퓨터공학과 졸업(석사)

1994년 서울대학교 대학원 컴퓨터공학과 졸업(박사)

1994년~1995년 서울대학교 컴퓨터신기술공동연구소 특별연구원

1995년~현재 한성대학교 컴퓨터공학부 부교수

관심분야 : 멀티미디어 시스템, 분산 실시간 시스템, ITS 등