연속매체 재연에 적합한 스케줄링 주기 확장을 허용하는 동적 Sweep 기법

임 성 채†

요 약

컴퓨터 성능 및 네트워크 기술의 발전으로 웹 상에서 연속매체(Continuous Media: CM)의 온라인 스트리밍(streaming) 서비스가 일반화 되었다. 이런 서비스를 위해서는 다양한 종류의 CM 스트림을 효과적으로 처리할 수 있어야 하며 이를 위해 Sweep 기법이 연구되었다. 이 기법은 끊김현상 없이 스트리밍 서비스를 할 수 있고 탐색지연 시간을 최적화 할 수 있다는 장점을 가지고 있다. 하지만 서비스 하고 있는 스트림의 스케줄링 주기를 사이클이라고 하는 단일한 크기로 맞춰야 하기 때문에 스케줄링의 유연성이 떨어지며, 이로 인해 발생하는 디스크 대역폭의 낭비가 매우 커질 수 있다는 문제를 가지고 있다. 논문에서는 이런 문제점을 해결하기 위해 CM 스트림의 스케줄링 주기를 동적으로 확장할 수 있는 기법을 연구하며, 이 기법에 적합한 새로운 승인제어 방식을 이용하여 끊김현상을 막는다. 논문에서는 제안된 기법의 성능상의 우수성을 보이기 위해 모의실험이 수행되었으며, 모의실험 결과를 통해 제안된 동적 Sweep 기법이 디스크 이용효율과 스케줄링 유연성의 측면에서 기존 Sweep 기법에 비해 우수한 특성이 있음을 알 수 있다.

키워드 : 실시간 디스크 스케줄링, 연속매체, 스트리밍 서비스, 멀티미디어

A Dynamic Sweep Scheme Enabling Scheduling Period Expansions for Continuous Media Playback

Sung Chae Lim†

ABSTRACT

With fast advances in computing power and network technologies, online streaming services of continuous media (CM) have been popularly implemented on the Web. To implement such services, a variety of CM streams need to be processed efficiently, so that the Sweep scheme was proposed. This scheme has several advantages such as hiccup-free playbacks and seek-time optimization. In this scheme, however, the entire CM streams are scheduled with a single scheduling period, called a cycle. Since only one scheduling period is allowed in this scheme, a significant amount of disk time is usually wasted because of its inflexible disk schedules. To solve this, we propose a new dynamic Sweep scheme. For this, we devise an algorithm that is able to expand scheduling periods of serviced CM streams and propose a new admission control mechanism guaranteeing hiccup-free playbacks. To show performance gains, we execute various simulation experiments. From the experimental results, we can see that the proposed scheme outperforms the Sweep scheme in terms of disk utilization and scheduling flexibility.

Key Words : Real-time Disk Scheduling, Continuous Media, Multimedia, Streaming Service

1. 서 론

컴퓨터 통신망과 CPU 처리 속도의 빠른 발전과 더불어 연속매체(continuous media)를 온라인 상에서 실시간으로 재연해주는 서비스가 보편화된 상황에 있다. 여기서 연속매체라 함은 MPEG 코딩된 동영상이나 음악사운드와 같이 해당 매체를 재연함에 있어 데이터를 일정한 속도로 그리고 연속적으로 사용자의 컴퓨팅 장치가 소모할 수 있어야 하는 매체를 의미한다. 이런 연속매체는 고유의 재연속도(playback rate)를 가지며, 재연속도 r을 갖는 연속매체가 시점 s에 재연을 시작하였다면, 매 시점 t 마다 $r\times(t-s)$ 크기의 데이터를 소모할 수 있어야 재연시의 끊김현상을 막을 수 있다. 이런 특성의 연속매체를 여러 사용자에게 동시에 서비스하는 멀티미디어 시스템에서는 연속매체가 가지는 데이터 읽기의 요구사항을 지킬 수 있는 시스템의 자원관리 기법, 특

※ 이 논문은 2005년도 동덕여자대학교 연구비 지원에 의해 수행된 것임.
† 정 회 원 : 동덕여자대학교 컴퓨터전공 교수
논문접수 : 2005년 4월 28일, 심사완료 : 2005년 8월 2일

히 실시간 특성의 디스크 스케줄링 기법이 필요하다[1, 2].

연속매체의 온라인 서비스를 위해 초기 연구되었던 분야로 상용 VOD(Video-on Demand) 서비스 분야를 들 수 있다. 이 서비스에서는 VCR 화질 정도의 고품질 스트림(stream)을 제공하기 위해 서비스되는 비디오 스트림의 재연속도를 1.5 Mbps 정도로 둔다. 이런 고품질의 비디오 스트림을 유료 서비스로 제공하기 위해 필요한 다양한 기술적인 문제들이 다뤄졌다. 특히 인기(hot) 비디오를 메모리상에 일부 캐싱해 둘 수 있는 메모리 관리기법[3, 4, 5, 6], 데이터를 여러 개의 디스크에 분산 저장하여 성능 및 비용을 좋게 할 수 있는 기법[7, 8, 9, 10, 11], ATM(Asynchronous Transfer Mode) 기술에 기반한 효과적인 네트워크 구성에 관한 기법[12, 13, 14, 15] 등이 많이 연구되었다. 이런 VOD 시스템에서는 스트림의 재연속도가 크고 재연기간이 매우 길기 때문에 특정 하나의 디스크를 위한 스케줄링 기법에 대해서는 큰 관심이 없었다.

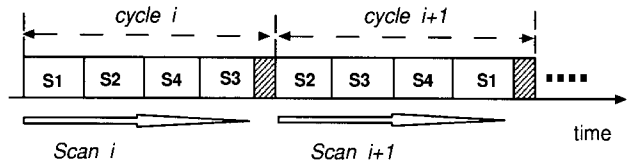
이와는 달리 본 논문에서는 다양한 재연속도를 가지는 여러 가지 종류의 연속매체 스트림을 온라인 서비스 하는 시스템에서의 스케줄링 문제를 다룬다. 논문에서 고려하는 스트림들은 상용 VOD 시스템의 스트림에 비해 작은 크기의 재연속도와 짧은 재연시간을 요구하는 스트림들로서, 현재 웹 서버를 통해 흔히 접할 수 있는 음악서비스나 비디오 클립 서비스 등이 하나의 예라 하겠다.

이런 연속매체 데이터는 비교적 작은 크기를 가지기 때문에, 하나의 디스크에 여러 개가 함께 존재할 수 있으며, 다수 사용자로부터의 재연요청을 동시 수용할 수 있어야 한다. 이런 경우 여러 개의 스트림들이 하나의 디스크 자원에 대해 서로 디스크 대역폭을 사용하려고 경쟁하기 때문에 적절한 스케줄링 기법이 없다면 서로간의 간섭으로 인한 끊김현상을 피하기 어렵다. 또한, 과도한 디스크 탐색시간이 발생하여 디스크 대역폭을 효과적으로 사용하지 못하는 문제도 함께 발생할 수 있다[16, 17].

이처럼 다양한 크기의 재연속도를 가지는 연속매체 스트림을 같은 디스크로부터 서비스하기 위한 스케줄링 기법으로는 Fixed-stretch, Sweep, GSS(Grouped Sweep Scheme), Sweep, SCAN-EDF, DM-SCAN 등의 연구가 있다[18, 19, 20, 21, 22, 23]. 이중에 Sweep 기법은 시스템에서 재연을 승인한 모든 스트림에 대해서 끊김현상을 완전히 없앨 수 있을 뿐만 아니라 디스크 SCAN 알고리즘의 장점인 디스크 탐색시간을 최소화함으로써 동일한 디스크 상에서 보다 많은 수의 연속매체 스트림을 서비스 할 수 있는 장점이 있다.

Sweep 기법에서는 사이클(cycle)이라고 하여 시스템에서 일정하게 잡은 시간간격에 맞춰 모든 연속매체 스트림의 읽기요청들이 처리되는 것이 특징이며, 이때 각 사이클 안에 처리해야 하는 읽기요청은 디스크 SCAN 알고리즘에 의해 처리된다. (그림 1)은 네 개의 연속매체 스트림 $S_j(1 \le j \le 4)$를 Sweep 기법으로 서비스 할 때 발생할 수 있는 스케줄링의 한 예이다.

(그림 1)에서 편의상 각 스트림들은 매 사이클 마다 하나의 데이터 읽기요청만을 발생시키는 것으로 가정했으며, 이런 읽기요청에 대해서 실제 사용된 디스크 시간은 그림에서 사각형으로 표현했다. 사이클 i에서는 $S_1 \rightarrow S_2 \rightarrow S_4 \rightarrow S_3$ 의 스트림 순서로 읽기요청이 처리되었으며, 다음 사이클 $j+1$에서는 $S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_1$의 순서로 서비스 되었다. Sweep 기법을 통해 서비스 하는 경우 스트림들의 읽기요청은 SCAN 방식에 따라 디스크를 이동시킬 때 디스크 헤드위치에 가까운 순서로 처리되기 때문에, 스트림의 데이터들 간의 위치변화에 따라 스트림이 서비스 되는 순서도 사이클에 따라 변한다.



▨: slack time in the cycle

(그림 1) Sweep 기법을 이용한 네 개 스트림의 서비스 예

재연속도 r을 가지는 임의의 스트림을 사이클 크기 T_{cycle}를 가지는 Sweep 기법에 따라 서비스 하려면 해당 스트림은 각 사이클 마다 최소 $r \times T_{cycle}$ 크기의 데이터를 읽을 수 있어야 하며, 사이클 i에서 읽은 데이터는 다음 사이클 $i+1$ 동안 소모된다. 또한 서비스하고 있는 스트림들의 모든 읽기요청을 한 번의 디스크 스캔을 통해 처리하기 때문에 (그림 1)에서 보이듯이 디스크 여유시간(slack time)은 각 사이클의 뒤쪽 부분에 생긴다.

좀더 Sweep 기법의 특성에 대해 설명하기 위해 임의의 사이클 i의 종료시점을 기호 $E_t(i)$로 표시해 보자. Sweep 기법에서는 사이클 i 동안에 처리되어야 할 모든 데이터 읽기요청은 시점 $E_t(i)$를 동일한 마감시간으로 가지며, 사이클 i 동안 처리될 읽기요청은 $E_t(i-1)$에 생성된다. 즉 Sweep 기법에서는 재연되고 있는 스트림들이 자신의 읽기요청을 매 사이클의 시작점에 요청하며 이런 읽기요청들은 해당 사이클이 끝나기 전에 항상 처리된다.

이처럼 디스크 읽기의 마감시간이 항시 사이클의 끝 시점에 나타나기에 재연시작 시점 또한 사이클의 끝 시점에서만 가능하다. 예를 들어 (그림 1)에서 스트림 S_1이 자신의 최초 데이터를 사이클 i의 앞쪽 시간에 읽어 들였어도 재연시작을 즉각적으로 할 수는 없다. 대신 시점 $E_t(i)$에서야 비로소 재연시작을 할 수 있으며, 이는 (그림 1)에서도 보이듯이 다음 사이클 $i+1$에서는 S_1의 데이터가 가장 뒤쪽 시점에 읽혀질 수도 있기 때문이다. 사이클 간에 일정하게 유지되지 않는 데이터 읽기시점 간격에도 불구하고 끊김현상이 생기지 않도록 하기 위해 재연시작 시점은 항상 사이클의 경계(boundary)에서만 허용된다. 이처럼 재연시작 시간이 사이클의 크기 T_{cycle}에 좌우되기 때문에 적당한 크기로 이 값을 정하는 것이 중요하다 [18].

Sweep 기법은 다양한 재연속도의 연속매체들을 동시에 서비스 할 때 끊김현상을 없앨 수 있는 기법으로 알려진 방법 중에서 가장 우수한 성능을 가진다[18, 22]. 물론 이 기법은 Fixed-stretch나 GSS에 비해 메모리를 좀더 많이 사용한다는 단점이 있기는 하지만 요즘 사용 가능한 메인 메모리 크기가 빠르게 커진 상황에서는 큰 단점으로 볼 수는 없다. 본 논문에서는 이런 Sweep 기법의 확장기법에 대해서 연구한다. 제안하는 방법은 기존의 Sweep 기법과는 달리 각 스트림들이 서로 다른 길이의 스케줄링 주기를 가질 수 있다는 장점을 가진다. 이와 함께 기존 Sweep 기법에서와 같이 재연승인된 모든 스트림들에 대해서 끊김현상을 막을 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 디스크 저장 구조에 대한 기본 사항 및 Sweep 기법의 데이터 프리팻칭에 대해서 기술한다. 이어서 3장에서는 제안하는 멀티미디어 시스템의 구조와 알고리즘에 대해 기술한다. 4장에서는 제안한 알고리즘의 특성에 대한 실험결과를 보이고, 5장에서 결론을 맺는다.

2. 연구관련 배경지식

2.1 사용된 디스크 모델

연속매체 데이터는 크기가 크고 연속적으로 읽혀지기 때문에 디스크 상에 물리적으로 클러스터링(clustering) 하여 저장하는 것이 일반적이다. 이에 따라 임의의 연속매체 데이터는 하나의 디스크 트랙을 단위로 해서 디스크 공간을 점유하며, 이 데이터를 구성하는 디스크 트랙들은 디스크 실린더 상에 자유롭게 위치할 수 있다. 데이터 클러스터링을 통해 연속매체 재연시에 발생할 수 있는 과도한 디스크 탐색지연 시간의 발생을 막아 디스크 대역폭의 이용효율을 높이는 것이 연속매체를 위한 일반적인 디스크 저장방식이다[10].

이런 데이터 클러스터링 하에서 각 연속매체 스트림은 한 개의 디스크 읽기요청을 생성하여 원하는 하나의 데이터 strip을 읽는다. 이때 하나의 데이터 strip은 동일한 디스크 트랙에 연속적으로 위치한 데이터를 의미하며, 논문에서는 스트림이 생성하는 데이터 읽기요청을 **C-request**라 칭하겠다. 각 데이터 strip은 (*Offset1*, *Offset2*, X) 형태의 데이터 **위치식별자**를 통해 지정된다. 여기서 *Offset1*은 트랙 내에서 데이터 strip의 시작위치, *Offset2*는 마지막 위치, 그리고 X는 해당 디스크 트랙 번호이다. 하나의 C-request에는 이 같은 하나의 데이터 위치식별자가 부여되며 마감시간도 함께 주어진다. 아무리 작은 크기의 데이터를 읽어 들인다 하여도 이 데이터가 두 개의 서로 다른 디스크 트랙에 걸쳐 존재한다면 두 개의 C-request가 요청되어야만 한다.

디스크 트랙 크기를 기본 단위로 데이터를 클러스터링 하는 경우, 대부분의 최신 디스크 드라이브가 가지고 있는 내부 캐쉬 메모리를 효율적으로 사용할 수 있다는 장점이 함께 따른다. 근래의 디스크 드라이브에는 수 MB 크기의 내부 캐쉬 메모리가 있어서 회전지연시간(rotational delay) 동안 디스크 헤드를 지나치는 디스크 섹터를 그냥 두지 않고 내부 캐쉬 공간에 저장시킬 수 있다. 이런 방식을 사용하여 전체 디스크 트랙을 읽어 들이는 경우 기존의 회전지연 시간이란 것을 무시할 정도로 작게 만들 수 있다[24].

이런 사실에 따라 본 논문에서는 하나의 C-request를 처리하기 위해 필요한 데이터 전송시간과 회전지연시간의 합을 한 번의 디스크 판 회전 시간과 같게 둔다. 따라서 η개의 C-request들을 처리하기 위해 필요한 전체 디스크 스캔 시간 $T_{scan}(\eta)$을 아래 식 (1)과 같이 표시할 수 있다. 아래 식에서 기호 $T_{revolution}$은 디스크 판이 한번 회전하는데 걸리는 시간을 나타내며, $SeekTime$ l만큼의 디스크 실린더 거리를 이동할 때 필요한 탐색지연 시간이다. 전체 이동거리 $\sum_{i=1}^{\eta+1}(l_i)$의 크기는 디스크의 전체 실린더 개수인 N_{cyl}의 값과 같다.

$$T_{scan}(\eta) = \eta \times T_{revolution} + \sum_{i=1}^{\eta+1} SeekTime(l_i),$$
$$\text{where} \quad \sum_{i=1}^{\eta+1} l_i = N_{cyl} \tag{1}$$

기존 연구에 따르면 디스크의 탐색지연 시간 $SeekTime(l)$은 아래와 같은 식으로 모델링 될 수 있다[24]. $SeekTime(l)$은 디스크 마다 주어지는 실린더 거리 L을 초과하는 거리를 이동하는 경우는 이동 거리에 비례해서 탐색지연시간이 증가하는 특성을 가지며, 식의 C_x는 디스크의 하드웨어 특성에 따른 값이다.

$$SeekTime(l) = \begin{cases} C_1 + C_2\sqrt{l}, & \text{if } l \le L, \\ C_3 + C_4 \times l, & otherwise. \end{cases}$$

위의 탐색지연시간 함수에 따라 한 번의 디스크 스캔으로 여러 개의 데이터 strip을 읽어 들일 때, 한 번의 디스크 스캔 기간 내에 발생하는 전체 탐색지연시간의 합은 읽어 들이는 데이터 strip들이 디스크 실린더 상에 동일한 거리를 두고 위치할 때 가장 커진다[25]. 이런 특성에 따라 위의 식 (1)의 $T_{scan}(\eta)$가 가질 수 있는 Upper-bound 시간, $T^{ub}_{scan}(\eta)$는 아래 식으로 모델링 될 수 있다.

$$T^{ub}_{scan}(\eta) =$$
$$\eta \times T_{revolutiion} + (\eta+1) \times SeekTime(\lceil N_{cyl}/(\eta+1) \rceil). \tag{2}$$

본 논문에서는 위의 식 (2)으로부터 나온 결과를 이용하여 스캔 방식으로 항상 얻을 수 있는 최소한의 디스크대역폭 크기를 계산한다. 아래 식 (3)은 식 (2)의 $T^{ub}_{scan}(\eta)$를 이용하여 하나의 사이클 시간 내에 언제나 처리 할 수 있는 C-request의 최대 개수를 구하는 식이다. 즉, 아래 식 (3)을 만족시키는 정수값 K를 구함으로써 Sweep 기법이 하나의

사이클 시간 동안 항상 처리 가능한 C-request의 최대 개수를 얻는다.

$$T^{ub}_{scan}(K) \leq T_{cycle} < T^{ub}_{scan}(K+1). \quad (3)$$

본 논문에서는 정수값 K를 **대역폭허용계수**라고 부르며, 각 스트림이 자신이 소모할 데이터를 읽기 위해 매 사이클마다 요청할 수 있는 C-request의 최대개수를 **대역폭부하값**이라고 한다. 이런 정의에 따르면, 디스크 트랙 크기를 S_{track}으로 표시하고 스트림 S_i의 재연속도가 r_i일 때 이 스트림의 대역폭 부하값은 $\lceil r_i \times T_{cycle}/S_{track} \rceil$으로 계산된다. *Sweep 기법의 승인제어에 의해서 서비스 되는 모든 스트림들의 대역폭 부하값의 합은 K를 넘지 않도록 통제된다.* 이 값을 넘기는 사용자의 재연요청이 있는 경우는 해당 재연요청을 FIFO 대기큐에 넣는다.

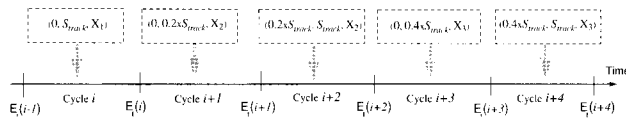
2.2 Sweep 기법에서의 데이터 프리팻칭

앞 장에서는 Sweep 기법의 디스크 모델 및 승인제어 기법에 대하여 설명하였다. 위와 같은 승인제어 기법에 의해 Sweep 기법이 스트림들을 서비스 할 때, 사이클 당 필요한 데이터 크기와 디스크 트랙 사이즈가 정확하게 일치하지 않기 마련이다. 이에 따라 작은 크기의 데이터라 하더라도 두 개의 디스크 트랙에 걸쳐 존재하는 경우 추가적인 탐색지연 시간이 요구됨으로써 예기치 못한 끊김현상을 발생시킬 수 있다. 이런 문제를 피하기 위해 Sweep 기법에서는 일반적으로 데이터 프리팻칭을 사용한다[18, 22].

이 방법에 대해서 알아보기 위해 사이클 당 $0.6\times S_{track}$의 데이터를 소모하는 스트림 S를 고려해 보자. 이 스트림의 재연속도는 정의에 의해 $0.6\times\frac{S_{track}}{T_{cycle}}$의 크기를 가지며, 이 스트림의 대역폭 부하값은 1이다. 또한 스트림 S가 사이클 i에서부터 데이터를 읽기 시작하고, S의 데이터는 $X_1, X_2, \ldots, X_n$의 디스크 트랙에 저장되어 있다고 가정하자.

이런 경우 사이클 i 동안 읽혀질 데이터 strip은 위치식별자 (0, $0.6\times S_{track}$, X_1)로 지정되며 이를 읽기 위해 S는 하나의 C-request를 시점 $E_t(i-1)$에 생성할 것이다. 이어서 다음 사이클 $i+1$동안에 읽어야 할 S의 데이터는 두 개의 디스크 트랙 X_1과 X_2에 걸쳐 위치하며, 이들은 각각 위치식별자 ($0.6\times S_{track}$, S_{track}, X_1)와 (0, $0.2\times S_{track}$, X_2)로 표시된다. 이처럼 두 개의 디스크 트랙에 저장된 데이터들을 읽기 위해서는 두 개의 C-request가 필요하다. 이 경우 재연승인 당시보다 많은 디스크 대역폭이 필요로 될 수 있으며(초기 주어지지 않았던 탐색지연 시간이 추가로 발생한다) 이는 Sweep 기법에서 허용되는 상황이 아니다.

이런 현상을 막기 위해 Sweep 기법에서는 프리팻칭을 사용한다. (*Offset*, *Offset2*, X)를 위치식별자로 갖는 임의의 데이터 strip을 읽을 때 트랙의 뒤 쪽에 남겨질 데이터의 크기를 *Remain*(= S_{track} - *Offset2*)이라 하고, 이 크기가 작다면 (*Offset2*, S_{track}, X)의 데이터를 미리 프리팻칭 한다. 좀 더 정확히 표현한다면 스트림이 소모하는 사이클 당 데이터 크기를 D라 할 때, D **mod** S_{track}의 크기보다 *Remain* 크기가 작다면 나머지 데이터인 (*Offset2*, S_{track}, X)를 함께 읽어 들인다. (그림 2)는 이 같은 방식의 프리팻칭을 이용하여 위에서 예로 든 스트림 S가 데이터를 읽어 들이는 모습을 보인다. 그림과 같이 적정량의 데이터를 프리팻칭 함으로써 각 사이클 마다 오직 한 개의 C-request만으로도 원하는 데이터를 읽어 들일 수 있다.



(그림 2) 프리팻칭을 이용한 스트림 S의 데이터 읽기 방식

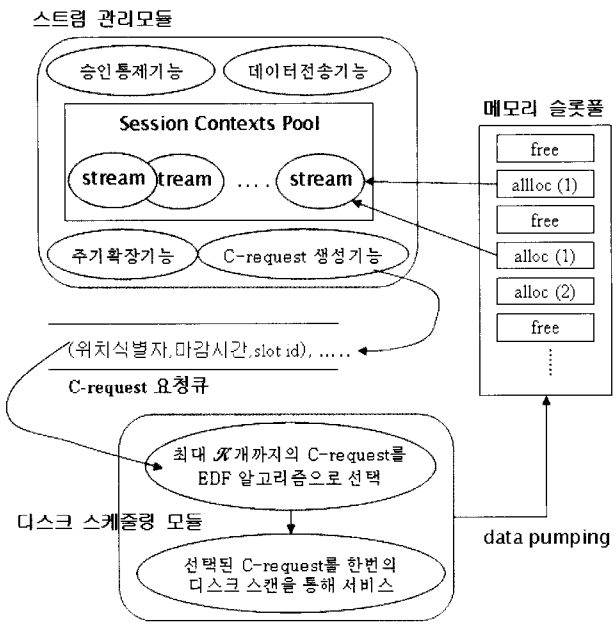
3. 제안하는 Dynamic Sweep 기법

3.1 시스템 구조

제안하는 멀티미디어 시스템은 크게 스트림 관리모듈과 디스크 스케줄링 모듈로 구성된다. 스트림 관리모듈은 승인제어 동작과 디스크로부터 읽은 데이터를 사용자에게 전송하는 역할을 담당한다. 이와 함께 재연이 승인된 사용자 요청에 대해 세션문맥(session context)을 관리하며, 각 세션문맥의 데이터 요구사항에 따라 C-request를 생성한다. 생성된 C-request는 디스크 스케줄링 모듈로 보내져 처리된다. 디스크 스케줄링 모듈에서는 요청큐로 들어오는 C-request들의 마감시간 정보를 참조하여 각 디스크 스캔 시간에 처리해야 할 C-request 들을 결정한다. 선택된 C-request들을 디스크 스캔방식으로 처리하며, 대응되는 데이터 strip들은 메모리로 읽혀진다.

(그림 3)은 제안하는 시스템의 대략적인 구조이다. 디스크 스케줄링 모듈은 널리 쓰리고 있는 EDF 알고리즘 방식에 따라 C-request 요청 큐에서 최대 K개까지의 C-request를 한 번의 S_{track} 스캔으로 처리한다. 스캔을 통해 읽혀진 데이터 strip들은 메인 메모리에 위치한 메모리 슬롯에 적재된다. 메모리 슬롯풀에 존재하는 메모리 슬롯은 리스트 형태로 관리되며, 슬롯 하나의 크기는 디스크 트랙의 크기 S_{track}과 동일하게 생성한다.

하나의 C-request을 생성할 때 스트림 관리모듈은 해당 데이터 strip을 읽어 들일 메모리 슬롯을 메모리 슬롯풀로부터 할당받는다. 할당 받은 슬롯의 식별자와 마감시간을 C-request에 함께 붙여 디스크 스케줄링 모듈로 보낸다. 이후 디스크 스케줄링 모듈은 이 슬롯 식별자를 이용하여 대응되는 슬롯에 데이터를 로딩하며, 이 데이터는 스트림 관리모듈에 의해 소모된다. 데이터가 완전히 소모된 슬롯은 스트림 관리모듈에 의해 다시 슬롯풀로 반환된다.



(그림 3) 제안된 시스템의 주요 모듈과 각 모듈의 기능 개요도

3.2 주기적 특성의 스케줄링 패턴

(그림 1)에서 보았듯이 각 사이클의 뒤쪽에는 여유분의 시간이 발생한다. 이런 여유분의 시간은 서비스 중인 스트림들이 K개의 C-request를 모두 사용할 때에도 늘 발생한다. 이는 승인제어에 사용할 대역폭허용계수가 worst-case의 디스크 사용 시간을 고려했기 때문이다. 사이클 내의 여유분 시간은 Sweep 기법에서는 idle 시간으로 버려진다[18]. 이와는 달리 본 논문에서는 이런 여유분의 시간을 이용하여 개별 스트림의 스케줄링 주기를 확장시킴으로써 점유하는 디스크 대역폭의 크기를 줄일 수 있다. 이를 통해 보다 많은 수의 스트림들을 동일한 디스크를 이용하여 서비스 할 수 있다.

이를 설명하기 위해 다시 스트림 S의 데이터 읽기 상황을 보이는 (그림 2)로 돌아가기로 한다. 앞서 가정했듯이 S는 최소 $0.6\times S_{track}$ 크기의 데이터를 매 사이클 마다 읽어 들여야 하며, 사이클 i에서 자신의 최초 데이터를 읽을 경우 이 스트림의 재연 시작 시점은 $E_t(i)$이다. 스트림 S는 기본적으로 각 사이클 마다 한 개의 디스크 트랙 전체를 읽어 들일 수 있는 디스크 대역폭을 점유하고 있기에 이를 모두 사용하여 사이클 $i+1$과 $i+2$에서도 전체 디스크 트랙을 읽어 들인다고 가정해보자. 그러고 나서 시점 $E_t(i+2)$에서의 데이터 캐쉬 상황을 살펴보기로 한다.

스트림 S는 이 시점까지 3개의 디스크 트랙을 읽었고 사이클 $i+1$과 $i+2$ 동안에 $0.6\times S_{track}$ 크기의 데이터를 각각 소모했으므로 시점 $E_t(i+2)$에 $1.8\times S_{track}$ 크기의 데이터를 사용하지 않은 채로 캐쉬 할 것이다. 이 데이터는 스트림 S가 3개의 사이클 동안 소모할 수 있는 정도의 데이터이다. 본 논문에서는 이런 시점에서 S의 스케줄링 주기를 최초 하나의 사이클 크기에서 3개의 사이클로 크기를 확장시킨다. 동시에 스트림 S는 매 3개 사이클 마다 2개의 C-request를 생성할 수 있도록 허가된다. 이 경우 스트림 S가 점유할 수 있는 디스크 대역폭의 크기는 $\frac{2S_{track}}{3T_{cycle}}$ 이므로 S의 재연속도 $0.6\times\frac{S_{track}}{T_{cycle}}$ 보다는 큰 값이다. 따라서 끊김 현상 없이 필요한 데이터를 읽어 들일 수 있다.

이처럼 논문에서는 사이클의 경계에서 스케줄링 주기를 확장할 수 있는 스트림이 생겼을 때 해당 스트림에게 새로운 **스케줄링 패턴**(scheduling pattern)을 부여하며, 이런 스케줄링 패턴은 스케줄링 주기의 크기를 나타내는 정수 p와 그 주기당 생성할 수 있는 C-request의 최대수 k로 표현된다. 이를 표시할 때 $<k, p>$의 기호를 사용한다.

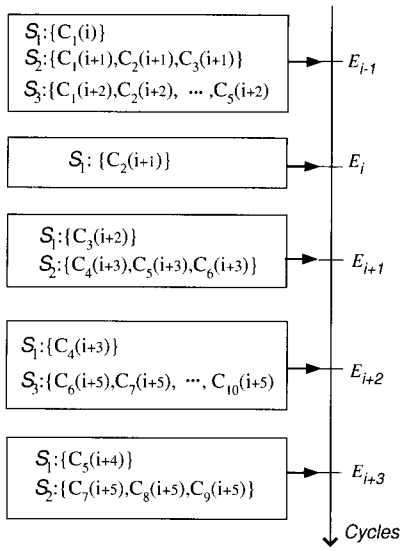
초기 재연승인 된 모든 스트림들은 $<k, 1>$ 형태의 스케줄링 패턴을 가지며 이후 여유분의 시간을 이용하여 스케줄링 주기를 확장할 경우 $<k', p'>$ $(p' > 1)$의 새로운 스케줄링 패턴을 가질 수 있다. 이때 언제나 $k'/p' < k$의 조건이 성립하도록 스케줄링 패턴을 확장하여 해당 스트림이 점유하는 디스크 대역폭의 크기를 줄인다. 이렇게 줄어든 디스크 대역폭을 다른 새로운 스트림의 서비스에 할당함으로써 서비스 가능한 전체 스트림의 숫자를 크게 한다.

논문에서는 임의의 디스크가 대역폭허용계수 K를 가질 때, 스케줄링 패턴 $<k, p>$를 갖는 스트림에 대해 실수값 $k/(p \times K)$을 **작업량비율값**(workload rate) 으로 정의한다. 이 값은 초기 재연승인 될 때는 항상 k/K의 값을 가질 것이다. 본 논문에서는 위와 같이 정의된 작업량 비율값에 기반을 두어 승인제어 및 디스크 스케줄링을 수행한다.

3.3 EDF 알고리즘에 기반한 스케줄링 기법

제안한 기법에서는 서비스하는 스트림 별 디스크 여유시간의 이용 형태나 각각의 재연속도 차이에 따라 서로 다른 스케줄링 패턴을 가지는 것이 보통이다. 이에 따라 디스크 스케줄링 모듈이 처리해야 하는 요청큐 내의 C-request들의 마감시간들도 서로 다르다. 이것은 기존 Sweep 기법에서는 요청큐 내의 모든 C-request들이 동일한 마감시간을 갖는 것과는 구별된다. 앞서 언급한 바와 같이 Sweep 기법에서는 큐에 있는 모든 C-request들이 동일한 마감시간을 갖기에 이들 모두를 선택하여 한 번의 디스크 스캔으로 처리하면 된다. 이 때 승인제어 기법에 의거하여 큐에 들어있는 C-request 의 개수는 K를 넘지 않는다.

이와는 다르게 제안된 방식에 따라 서비스되는 스트림들은 서로 다른 스케줄링 패턴에 맞춰 C-request를 생성하기 때문에 요청큐에 들어있는 C-request의 개수나 이들이 가지는 마감시간도 서로 다르다. 이를 알아보기 위해 세 개의 스트림 S_1, S_2, S_3를 생각해 보자. 그리고 이들 스트림들의 스케줄링 패턴을 각각 <1, 1>, <3, 2> 그리고 <5, 3>이라 가정하자. 스트림들이 우연히 사이클 i에서 자신들의 C-request를 최초로 생성했다고 했을 때 이들이 생성하는 C-request의 형태는 (그림 4)(a)와 같은 모양을 가진다. 그



(a) Periodic issuing of C-requests

Time	Enqueued C-requests	Selected C-requests
at $E_t(i-1)$	S_1:{$C_1(i)$} S_2:{$C_1(i+1),C_2(i+1),C_3(i+1)$} S_3:{$C_1(i+2),C_2(i+2)$, ...,$C_5(i+2)$}	S_1:{$C_1(i)$} S_2:{$C_1(i+1),C_2(i+1),C_3(i+1)$} S_3:{$C_1(i+2)$}
at $E_t(i)$	S_1:{$C_2(i+1)$} S_2:{} S_3:{$C_2(i+2),C_3(i+2)$, ...,$C_5(i+2)$}	S_1:{$C_1(i+1)$} S_2:{} S_3:{$C_2(i+2),C_3(i+2)$, ...,$C_5(i+2)$}
at $E_t(i+1)$	S_1:{$C_3(i+2)$} S_2:{$C_4(i+3),C_5(i+3),C_6(i+3)$} S_3:{}	S_1:{$C_1(i+2)$} S_2:{$C_4(i+3),C_5(i+3),C_6(i+3)$} S_3:{}
at $E_t(i+2)$	S_1:{$C_4(i+3)$} S_2:{} S_3:{$C_6(i+5),C_7(i+5)$, ...,$C_{10}(i+5)$}	S_1:{$C_4(i+3)$} S_2:{} S_3:{$C_6(i+5),C_7(i+5)$, ...,$C_9(i+5)$}
at $E_t(i+3)$	S_1:{$C_5(i+4)$} S_2:{$C_7(i+5),C_8(i+5),C_9(i+5)$} S_3:{$C_{10}(i+5)$}	S_1:{$C_5(i+4)$} S_2:{$C_7(i+5),C_8(i+5),C_9(i+5)$} S_3:{$C_{10}(i+5)$}

(b) A feasible schedule for (a)

(그림 4) 스트림 3개의 C-request 생성 및 EDF 알고리즘에 의한 처리방법

림에서 기호 $C_j(i)$는 해당 스트림으로부터 j번째 생성된 C-request를 나타내며, 이것의 마감시간은 $E_t(i)$임을 의미한다. (그림 4)(a)에서 보이듯이 세 개의 스트림들이 생성되는 C-request 전체 개수도 사이클에 따라 변화가 있다.

(그림 4)(a)과 같이 스트림들이 생성한 C-request들은 요청큐에 쌓이게 되며 이들을 처리하는 것은 디스크 스케줄링 모듈이다. 제안한 방식에서는 디스크 스케줄링 모듈이 대역폭허용계수 K 범위 내에서 C-request 를 선택하여 디스크 스캔으로 처리하는 방식을 취한다. 이 때 큐에 K 보다 많은 C-request들이 존재할 수도 있기에 디스크 스케줄링 모듈은 EDF 알고리즘에 따라 마감시간이 가까운 C-request를 우선적으로 선택하여 처리한다[26]. 이런 선택의 예는 (그림 4)(b)에서 보인다. 편의상 (그림 4)(b)에서 $K=5$로 했으며, 그림의 스케줄은 모든 C-request의 마감시간을 만족시키는 스케줄이다.

(그림 4)와 같이 주기적인 C-request 생성과 EDF 알고리즘에 의해 C-request까 처리될 때, 다음과 같은 조건이 만족되어야 C-request의 마감시간을 지킬 수 있다. 아래와 같은 조건은 다음 절에 기술될 승인제어 기법 및 스케줄링 주기 확장 기법에 의해 지켜진다.

임의의 시점 $E_t(j)$에 C-request 대기큐에 $E_t(i)$를 마감시간으로 하는 C-request들의 개수를 $W_j(i)$로 표시해 보자. C-request 중 마감시간을 지킬 수 없는 경우는, $W_j(i)$의 정의에 따라 다음의 두 경우 중 하나이다. 즉, (i) $W_j(j+1)>K$ 이거나, (ii) $W_j(i)>0$ for any $i\le j$. 이런 사실에 따라 아래의 마감시간 만족조건을 얻을 수 있으며, 제안하는 기법은 자연수 j에 대해 아래 식을 항상 만족시킴으로써 끊김현상 없이 연속매체를 재연한다. 아래 조건식에서 기호 $P_{\max}$는 $P_{\max}=Max(p_i)$로, 모든 스트림의 스케줄링 패턴 $<k_i,\ p_i>$에 대해서 가장 큰 p_i 값이다.

[마감시간 만족조건]

$W_j(j+1) <= K$ and $W_j(j-P_{\max}) = W_j(j-P_{\max}+1) = \ldots = W_j(j) = 0$

3.4 제안하는 알고리즘

본 절에서는 앞서 기술한 EDF 알고리즘과 스케줄링 패턴에 기반한 Dynamic Sweep 기법에(이하 D-Sweep) 대해 기술한다. 먼저 승인제어에 사용되는 알고리즘을 설명한다. 아래에 보이는 세 개의 단계는 재연승인을 요청한 스트림에 대해서 재연승인이 가능한 지를 결정하기 위한 것이다. 제안된 승인제어 기법은 스트림의 작업량 비율값에 기초하여 동작한다. 작업량 비율값이란 3.2절에서 정의되었듯이 스케줄링 패턴<k, p>을 가지는 스트림에 대해서 $k/(p\times K)$의 값으로 주어진다. D-Sweep 기법에서는 서비스 되고 있는 전체 스트림 집합의 작업량 비율값의 합이 1이 넘지 않도록 통제함으로써 C-request의 마감시간을 지키지 못해 오는 끊김현상을 막는다.

(1) 재연속도 r를 S에 대해 정수 k를 다음의 식으로 구한다. $k = \lceil (r\times T_{cycle})/S_{track} \rceil$.

(2) 현재 서비스되고 있는 스트림 집합에 대한 작업량 비율값의 합인 W_c를 다음의 식으로 구한다. $W_c = \sum_{j=1}^{|S|}(k_j/(p_j\times K))$. 왼쪽의 식에서 $<k_j,\ p_j>$는 서비스 되고 있는 스트림 S_j들의 스케줄링 패턴을 나타낸다.

(3) 앞에서 구한 k와 W_c를 이용하여 승인가능 여부를 체크한다. 즉, 만약 $W_c + k/K$의 값이 1보다 작거나 같다면 스트림 S를 승인한다. 만약 1보다 크다면 다른

스트림이 재연을 끝내고 디스크 대역폭을 반환할 때까지 S를 대기큐에 넣는다.

위의 재연승인 조건을 만족시킨 스트림은 재연초기의 스케줄링 패턴 <k, 1>를 가지고 서비스를 시작한다. 재연승인된 스트림에 대한 데이터 전송 및 서비스 상태의 추적을 위해 이에 대응되는 세션문맥이 생성되며 이를 스트림 관리모듈이 관리한다. 이후 스트림은 여유분의 디스크 시간을 이용하여 자신의 스케줄링 주기를 최초 하나의 사이클 길이에서 더 큰 크기로 확장할 수 있다. 다음 (그림 5)의 알고리즘은 이런 주기확장을 허용하는 D-Sweep 기법의 세부 알고리즘이다. 알고리즘은 스트림 관리모듈에 의해 수행되며, 알고리즘에 따라 생성된 C-request들은 디스크 스케줄링 모듈로 보내져 처리된다.

(그림 5)의 알고리즘은 임의의 사이클 i 동안에 스트림 관리모듈에 의해 수행되는 절차를 표현한 것이다. 이 기간 중 $E_t(i)-\varepsilon$이 되기까지 재연승인 작업을 수행하며 ε의 기간 동안 라인 (8)-(17)의 작업을 수행한다. 이 작업은 현재의 CPU 속도에 의하면 1 ms 안에 처리될 수 있는 정도의 작업이다.

1. 현재시간을 *CurrtTime*으로, 서버에서 관리되는 세션문맥 집합을 S_{CM}으로 표시한다.
2. **While** [*CurrentTime* < $E_t(i)$ − ε] **Do**
3. 사용자의 재연요청들을 저장하고 있는 FIFO 큐로부터 하나의 재연요청을 가져온다. 추출된 사용자 요청을 R로 표시하자.
4. 앞서 기술한 재연승인 단계에 따라 R이 승인 가능한지를 체크 한다. 만약 승인이 가능하지 않다면 R을 다시 FIFO 대기큐의 앞 쪽에 넣고 시점 $E_t(i)-\varepsilon$까지 대기한다.
5. 앞의 단계에서 R이 재연승인 되었다면 대응되는 세션문맥을 하나 생성하여 이를 *Obj*로 표시하자.
6. 세션문맥 *Obj*를 다음과 같이 초기화 한 후에 S_{CM}으로 삽입한다. *Obj.k* = *k*, *Obj.p* = 1 and *Obj.Deadline* = $E_t(i)$. // *k*는 대역폭부하값
7. **EndWhile** // Loop for admission control
8. **For** [each *Obj* $\in$ S_{CM}] **Do** // C-request 생성
9. **If** [*Obj.deadline* = $E_t(i)$] Then
10. *Obj*의 캐쉬 데이터 크기와 현재의 스케줄링 패턴을 참조하여 스케줄링 주기의 확장이 가능한지를 체크한다.
11. 만약 확장가능 하다면 *Obj.p*와 *Obj.k*의 값을 변화된 값으로 수정한다. 그렇지 않다면 다음 단계로 그냥 넘어간다.
12. *Obj.Deadline*의 값을 $E_t(i+Obj.p)$로 갱신한다.
13. 생성할 C-request 의 개수를 구하기 위해 다음의 식을 만족시키는 가장 작은 정수값 N을 구한다. *Obj.CacheSize* + N x S_{track} ≥ 2 x *Obj.PerCycleData* x *Obj.p*.
14. N개의 메모리 슬롯을 할당 받아 같은 수의 C-request들을 생성한다. 이 때 생성되는 C-request의 마감시간은 $E_t(i+Obj.p)$이다. 각각 하나의 메모리 슬롯을 할당받으며 읽혀질 데이터 strip은 하나의 디스크 트랙을 단위로 한다.
15. **Else** 아직 스케줄링 주기가 끝나지 않은 스트림이기에 C-request를 생성하지 않는다.
16. **EndFor** // End of C-request issues
17. 새롭게 생성된 C-request를 디스크 스케줄링 모듈로 보내 처리되도록 한다.

(그림 5) 사이클 i 동안의 스트림 관리모듈의 동작 알고리즘

라인 (10)-(11)에서는 서비스 하는 스트림 중에서 스케줄링 주기를 확장할 수 있는 것을 체크하고, 확장 가능한 것에 대해서는 *Obj.k*와 *Obj.p*를 수정하여 스케줄링 패턴을 변경시킨다. 알고리즘에서는 다음과 같은 조건식 (4)를 만족하는 두 정수 k'와 p'가 존재하는 가에 따라 스케줄링 패턴을 확장할 수 있는 지가 결정된다. 이 때 정수 p'는 *Obj.p* 보다 크고 $P_{\max}$ 보다는 작거나 같은 값이어야 한다. 여기서 $P_{\max}$는 스케줄링 주기의 확장 가능한 최대값이다.

$$p' \text{ x } Obj.PerCycleSize \le Obj.CacheSize \text{ and } k'/p' < Obj.k/Obj.p \quad (4)$$

위의 식 (4)에서 *Obj.PerCycleSize*는 해당 스트림이 하나의 사이클 동안 소모하는 데이터의 크기를 의미하고, *Obj.CacheSize*는 이런 속도로 시점 $E_t(i)$때 까지 데이터를 소모할 때 메모리에 캐쉬 되어 남아 있을 데이터의 크기를 나타낸다. 식 (4)를 만족시키는 정수 쌍이 존재한다면 해당 스트림은 스케줄링 주기를 확장할 수 있는 스트림이며, 라인 (11)에서 *Obj.k* = k'로, *Obj.p* = p'로 갱신된다. 즉, <k', p'>로 스케줄링 패턴이 확장된다. p'를 p+1에서 $P_{\max}$로 크게 하면서 조건식을 만족하는 k'를 구한다. 그리고 가장 작은 k'/p' 값을 가진 쌍을 선택하게 된다.

라인 (12)에서는 새로운 스케줄링 주기의 시작을 위해 스트림의 데드라인이 재조정되며, 이런 재조정에 의해 *Obj.p* 개의 사이클이 지난 후에 다시 새로운 C-request를 생성할 수 있다. 라인 (13)에서는 실제 생성될 C-request의 개수가 구해지는데, 원래의 Sweep 기법 특성에 따라 스케줄링 주기 동안 사용될 데이터에 대해 최대 2배 크기의 데이터가 캐쉬될 수 있음을 가정하여 계산된다[18].

4. 모의실험을 통한 성능비교

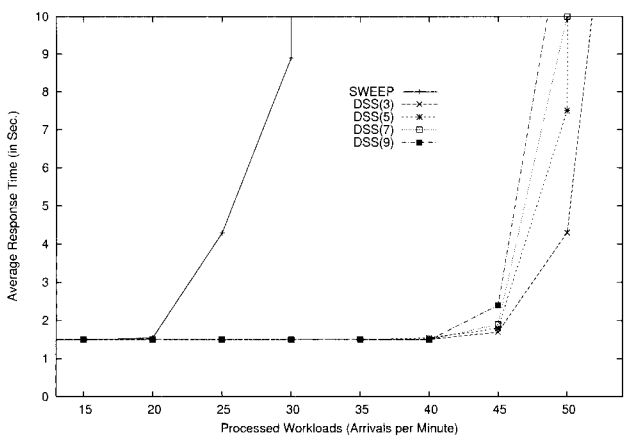
본 장에서는 모의실험을 통해 제안된 D-Sweep 기법의 특성을 Sweep 기법과 비교한다. 모의실험에 사용할 디스크

〈표 1〉 디스크 사양: IBM Ultrastar 73LZX

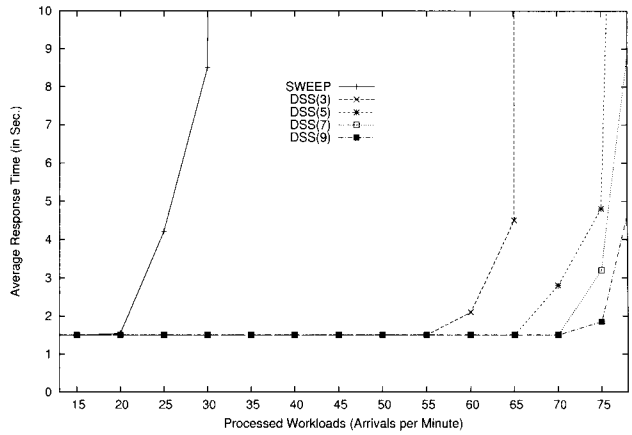
표시 사항	데이터 값
데이터 섹터 크기	512 bytes
전체 실린더 개수	20,847
트랙당 섹터수	390
$SeekTime(l)$	$1.85+0.07\sqrt{l}ms, If d<383$ $5.47+2.5*10^{-4}d$ ms, otherwise
디스크 용량	36.7 GBytes
디스크판 회전속도	10,000 RPM

드라이브에 대한 하드웨어 사양은 <표 1>에서 보인다. 모델링한 하드디스크 드라이브는 IBM SCSI 디스크 드라이브이다 [27]. 모의실험에서는 사용자의 연속매체 재연요청이 포아송 분포에 따라 시스템에 도착한다고 가정한다. 재연되는 연속매체의 재연속도는 [250 Kbps, 700 Kbps] 구간에서 균등(uniform)하게 분포한다. 이런 재연속도는 현재 웹 서버를 통해 인터넷에서 서비스 되고 있는 연속매체 스트림들을 대부분 포함한다고 볼 수 있다. 또 재연되는 연속매체의 재연 지속시간은 3분에서 5분 사이에서 균등하게 분포시켰다.

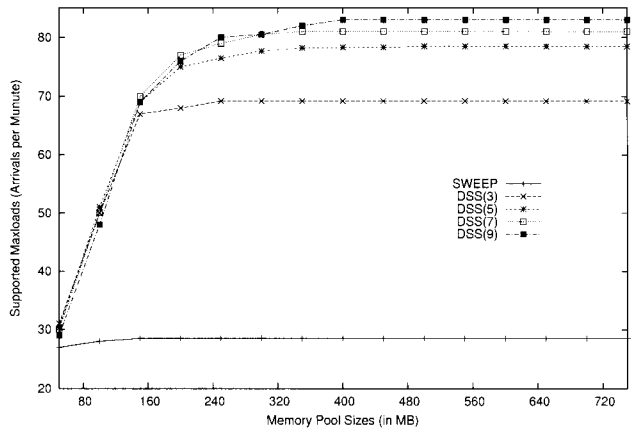
(그림 6)에서 SWEEP으로 표시된 그래프는 기존 Sweep 기법에 대한 성능을 보이다. 반면 DSS(x)로 표시된 것은 $P_{\max}=x$로 두었을 때의 D-Sweep 기법의 성능을 보인다. 즉, DSS(5)란 $P_{\max}$ 값을 5로 둔 것이며, 따라서 스케줄링 패턴의 확장을 통해 최대 5개 사이클 길이의 스케줄링 주기를 가질 수 있음을 의미한다. 실험에서 SWEEP과 DSS 모두 시스템에 들어오는 사용자의 재연요청이 적을 때는 1.5초 정도의 평균응답시간을 가진다. 여기서 응답시간이란 사용자의 재연 요청이 들어온 시점을 t, 이 요청이 승인되어 최초 하나의 사이클 동안 사용될 데이터를 읽은 사이클을 j라 했을 때, 시간간격 $E_t(j)-t$ 의 값과 같다. 그래프의 수직축은 이런 응답시간의 평균값을 보이고 있으며, 이를 평균응답시간이라 부른다. 그림에서 SWEEP은 분당 30개의 재연요청을 채 처리하기 전에 시스템의 과부하로 인해 평균응답시간이 폭주하고 있음을 보인다. 이에 반해 제안된 DSS는



(그림 6) 100 MB 크기의 슬롯풀에 대한 평균응답시간 추이



(그림 7) 400 MB 크기의 슬롯풀이 주어졌을 때의 평균응답시간 추이

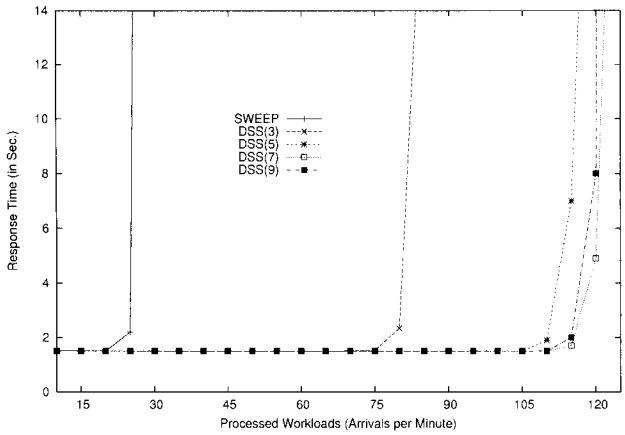


(그림 8) 슬롯풀 크기 변화에 따른 수용 가능한 최대 사용자 요청량

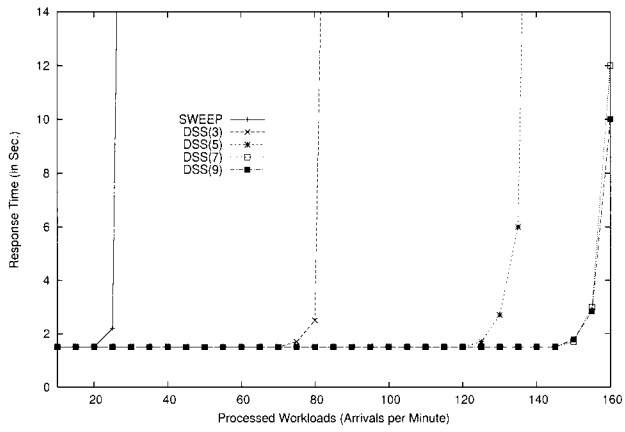
분당 45개의 재연요청이 들어오는 상태에서도 평균응답시간이 2 초 이하로 유지되고 있다.

(그림 6)에서 보인 성능상의 우위는 (그림 7)에서 더욱 분명해진다. (그림 7)은 (그림 6)의 실험을 조금 변화시킨 것으로, 사용할 수 있는 메모리 슬롯풀의 크기를 400 MB로 크게 했을 때의 변화 모습을 보인다. (그림 6)에서와 같이 슬롯풀의 크기를 상당히 제약했을 경우 DSS(9)와 DSS(7)의 성능이 DSS(3)에 비해 오히려 좋지 않았으나, (그림 7)에서는 그렇지 않은 결과를 보인다. 이것은 이 두 방식의 경우 $P_{\max}$ 값이 크기 때문에 다른 방식들에 비해 상대적으로 큰 버퍼 메모리를 요구하기 때문이다. 400 MB정도의 메모리를 사용하는 경우 이 두 방식이 DSS(3)에 비해 15%-25% 정도 더 좋은 성능을 보이고 있다. 현재와 같이 2GB 정도의 메모리 크기의 서버 메인메모리를 사용하는 것이 일반적인 상황에서 400 MB-500 MB 정도의 메모리를 사용하여 스트리밍 서비스를 운영하는 것은 다른 응용프로그램의 성능에 큰 영향을 주지 않은 적정한 범위라 할 수 있겠다.

(그림 8)은 사용할 수 있는 메모리 슬롯풀의 크기 변화에 따라서, 각 기법이 지원할 수 있는 최대 사용자 요청량을 보인다. DSS의 경우는 메모리 크기 변화에 따라 지원 가능



(그림 9) 멀티디스크를 이용한 평균응답시간(슬롯풀 크기=600 MB)



(그림 10) 멀티디스크를 이용한 평균응답시간(슬롯풀 크기=800 MB)

한 사용자 부하량의 개선은 거의 없다. 이에 비해 제안한 방식은 300 MB 정도의 메모리 사용만으로 매우 뚜렷한 성능 향상을 보이고 있음을 알 수 있다. 그리고 250 MB-400 MB 정도의 크기보다 큰 메모리에 대해서는 DSS 역시 성능 향상이 되고 있지 않다. 이는 아무리 많은 메모리를 사용할 수 있다고 해도 결국 지원 가능한 스트림의 수는 디스크의 대역폭에 한정되기 때문이다. 즉, 사용 가능한 디스크 대역폭 한도에서만 성능개선이 가능한 것이다.

(그림 9)는 앞의 앞선 실험들에서의 디스크 대역폭의 제한을 풀었을 때의 성능을 보인다. 멀티미디어 시스템의 경우 동일 서버에 장착 가능한 SCSI 디스크의 수는 4개 이상인 경우가 많고, RAID-5 방식에 의해 대역폭의 확장 및 디스크 무결성을 높이는 방식이 많이 사용된다. 이런 특성을 반영하기 위해 (그림 9)에서는 4개의 디스크를 RAID-5로 구성하여 설치한 경우에 대한 실험결과를 보였다.

(그림 9)에서 보이듯이 SWEEP에서는 특별한 성능개선을 볼 수 없다. 이것은 일정하게 정해진 사이클 길이로 인해 발생하는 디스크 대역폭 단편화 현상이 심해지기 때문이다. 즉, 재연승인된 각 스트림이 자신이 사용할 대역폭보다 과도한 디스크 대역폭을 점유하게 되기 때문인데, 이런 경우 사이클의 길이를 길게 해줘야 한다. 이럴 경우 늘어나는 사이클 길이에 비례해서 재연응답시간 역시 늘어나게 되므로 사이클 길이를 늘이는 방식은 적절한 문제 해결 방법으로 보기 어렵다[18, 22]. 이와는 달리 DSS(5), DSS(7), DSS(9)는 처리 가능한 사용자 요청량이 그림 7에 비해 많게는 70% 정도 향상됨을 알 수 있다.

마지막으로 (그림 10)은 앞의 (그림 9)에서 할당했던 메모리 크기를 800 MB로 좀더 크게 했을 때의 실험결과이다. 그림에서 알 수 있듯이 DSS(7)와 DSS(9)의 성능이 상대적으로 크게 향상되고 있으며, 처리 가능한 사용자 요청량이 (그림 9)에 비해 30% 이상 좋아지고 있다. (그림 9)에서는 DSS(9)의 성능이 DSS(7)에 비해서 좋지 않았으나 (그림 10)에서는 DSS(9)가 가장 좋은 성능을 보인다. 메모리의 크기를 더 크게 하는 경우 이 둘 간의 성능차는 좀 더 커지지만, 그 이상의 슬롯풀 사용은 실제 환경에서는 일반적이지 않다고 판단하여 800 MB 정도의 크기로 한정해 실험하였다.

5. 결 론

본 논문에서는 연속매체의 스트리밍 서비스에 적합한 Sweep 기법의 확장인 동적 Sweep 기법을 위한 디스크 스케줄링 기법 및 승인제어 알고리즘을 제안하였다. 제안된 기법은 특징은 기존 Sweep 기법이 가지고 있는 끊김현상 없이 연속매체 스트림을 서비스 할 수 있다는 장점 및 디스크의 탐색시간 지연을 최소화 함으로써 디스크 이용 효율을 크게 높일 수 있다는 장점을 제공한다. 이와 함께 실제 사용된 디스크 시간과 미리 확보된 디스크 대역폭과의 차에서 오는 디스크 여유시간을 이용하여 이미 재연승인이 된 스트림에 대해서 스케줄링 주기를 손쉽게 확장할 수 있는 알고리즘을 제시하였다. 이를 통해 동시 서비스 가능한 스트림의 수를 기존 Sweep 기법에 비해 크게 늘일 수 있었으며, 이때에도 사용자의 재연요청에 대한 응답시간을 보다 좋게 유지할 수 있다는 성능상의 장점을 가진다. 실험을 통해 제안된 동적 Sweep 기법이 요즘 사용되는 시스템에서 지원 가능한 크기의 메인 메모리 사용만을 요구하며, 멀티 디스크를 이용한 연속매체 재연 시스템에도 매우 효과적으로 적용될 수 있음을 보였다.

참 고 문 헌

[1] William I. Grosky. "Multimedia Information Systems," *IEEE Multimedia*, Vol.1, No.1, pp.12-24, 1994.

[2] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. "A Fast File System for UNIX," *ACM Trans. on Computer Systems*, Vol.2, No.3, pp.181-197, 1984.

[3] D. James Gemmell, Harrick M. Vin, Dilip D. Kandlur, and P. Venkat Rangan. "Multimedia Storage Servers: A Tutorial and Survey," *IEEE Computer*, Vol.28, No.5, pp.40-49, 1995.

[4] R. Wijayaratne and N. Reddy. "Integrated QoS Management for Disk I/O," In *Proc. of the IEEE Multimedia*

Systems, pp.487-492, June, 1999.

[5] Roger L. Haskin. "Tiger Shark-a Scalable File System for Multimedia," *IBM Journal*, Vol.42, No.2, pp.185-197, 1998.

[6] Doron Rotem and J. Leon Zhao. "Buffer Management for Video Database Systems," In *Proc. of the IEEE Intl. Conference on Data Engineering*, pp.439-447, 1995.

[7] J. Aerts, J. Korst, and S. Egner. Random Duplicate Storage Strategies for Load Balancing in Multimedia Servers. Technical report, NL-MS 20.314, 2000.

[8] S. Ghandeharizadeh and R. Muntz. "Design and Implementation of Scalable Continuous Media Servers," *Parallel Computing Journal*, pp.91-122, 1998.

[9] Huang-Jen Chen and Thomas D.C. Little. "Storage allocation policies for time-dependent multimedia data," *IEEE Trans. on Knowledge and Data Engineering*, Vol.8, No.5, pp.855-864, 1996.

[10] O. Ertug, M. Kallahalla, and P. J. Varman. "Real-Time Parallel Disk Scheduling for VBR Video Servers," In *Proc. of the Fifth Intl. Conference on Computer Science and Informatics*, February, 2000.

[11] Jose R. Snatos and Richard Muntz. "Performance Analysis of the RIO Multimedia Storage Systems with Heterogeneous Disk Configurations," In *Proc. of the ACM Intl. Conference on Multimedia*, pp.303-308, 1998.

[12] A. Ermedahl, H. Hansson, and M. Sjödin. "Response-Time Guarantees for ATM Networks," In *Proc. of the 18'th IEEE Real-Time Systems Symposium*, pp.274-284, 1997.

[13] M. Welsh, A. Basu, and T. Von Eicken. "ATM and Fast Ethernet Network Interfaces for User-level Communication," In *Proc. of the Third Intl. Symposium on High Performance Computer Architecture*, pp.332-342, February, 1996.

[14] S. McCanne et al. "Toward a Common Infrastructure for Multimedia-Networking Middleware," In *Proc. of NOSS-DAV*, pp.39-49, 1997.

[15] Lixin Gao and Don Towsley. "Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast," In *Proc. of the Intl. Conf. on Multimedia Computing and .....*, pp.117-121, 1999.

[16] R. K. Abbott and H. Garcia-Molina. "Scheduling I/O Requests with Deadlines: A Performance Evaluation," In *Proc. of the Real-Time Systems Symposium*, pp.113-125, 1990.

[17] P. Venkat Rangan and Harrick M. Vin. "Efficient Storage Techniques for Digital Continuous Multimedia," *IEEE Trans. on Knowledge and Data Engineering*, Vol.5, No.4, pp.567-573, 1993.

[18] Edward Y. Chang and Hector Garcia-Molina. "Effective Memory Use in a Media Server," In *Proc. of the Intl. Conference on Very Large Databases*, pp.496-505, 1997.

[19] D. Kandlur M. Chen and P. Yu. "Optimization of Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Systems," In *Proc. of the ACM Multimedia*, pp.235-242, 1993.

[20] P.S. Yu, M.S. Chen, and D.D. Kandlur. "Grouped Sweeping Scheduling for DASD-based Multimedia Storage Management," *ACM Multimedia Systems*, Vol.1, No.2, pp.99-109, 1993.

[21] A.L.N. Reddy and J.C. Wyllie. "I/O Issues in a Multimedia System," *IEEE Computer*, Vol.27, No.3, pp.69-74, 1994.

[22] Sungchae Lim and MyoungHo Kim. "Real-time Disk Scanning for Timely Retrieval of Continuous Media Objects," *Information and Software Technology*, Vol.45, No.9, pp.547-558, June, 2003.

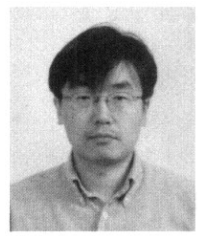
[23] Ray-I Chang, Wei-Kuan Shih, and Ruei-Chuan Chang. "Real-Time Disk Scheduling for Multimedia Applications with Deadline-Modification-Scan Scheme," *Real-Time Systems*, Vol.19, No.2, pp.149-168, 2000.

[24] C. Ruemmler and J. Wilkes. "An Introduction to Disk Modeling," *IEEE Computer*, Vol.27, No.3, pp.17-28, March, 1994.

[25] Yen-Jen Qyang. "A Tight Upper Bound of the Lumped Disk Seek Time for the SCAN Disk Scheduling Policy," *Information Processing Letters*, Vol.54, No.6, pp.323-329, 1997.

[26] C.L. Liu and J.W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, Vol.20, No.1, pp.46-61, 1973.

[27] IBM. IBM Disk Drive Specifications. *Http://www.storage.ibm.com*, 2002.



임 성 채

e-mail : sclim@dongduk.ac.kr

1992년 서울대학교 컴퓨터공학과(학사)

1994년 한국과학기술원 대학원 전산학과 (이학석사)

2004년 한국과학기술원 대학원 전산학과 (공학박사)

1999년 충남대학교 전산학과 강사

2000년~2000년 서울정보시스템 기술부 부장

2000년~2005년 2월 코리아와이즈닛 연구소 수석연구원 웹검색엔진 및 Enterprise 검색엔진 개발

현 재 동덕여자대학교 컴퓨터전공 교수

관심분야 : Web IR, 멀티미디어 시스템, 메인메모리 DBMS, 고성능 색인기법