새로운 스트림 요청에 의한 데이터 지연 문제를 피하기 위한 선행 버퍼링에 대한 연구

조경선[0] 원유집
한양대학교 전자통신전파공학과
{goodsun, yjwon}@ece.hanyang.ac.kr

Glitch-free Pre-buffering against New Stream Request

Kyung-Sun Cho[0] You-Jip Won
Division of Electrical and Computer Engineering, Hanyang University

요 약

멀티미디어 시스템에서는 미디어 데이터의 연속성을 보장하는 것이 중요한 문제이다. 90년대에 제안된 구역분할 디스크에서 연속성을 보장하면서 멀티미디어를 효과적으로 저장, 전송하기 위하여 새로운 스케쥴링 방식과 데이터 블록의 배치가 제안되었다. 이 방식은 구역을 순환하면서 데이터 블록을 배치시키고 SCAN 알고리즘으로 데이터를 읽어 들이는 방식이다. 이 경우 SCAN 알고리즘으로 데이터를 읽어 들이므로 이중 버퍼링(double buffering) 방법을 사용하게 된다. 이중 버퍼링의 데이터를 읽어 들이는 주기와 서비스 주기의 불일치성으로 인하여 새로운 스트림의 요청이 있을 때 기존의 서비스 스트림에 주기시간의 증가로 인한 데이터의 지연문제(jitter)가 발생한다. 본 논문에서는 구역분할 디스크를 이용하는 비디오 서버에서 새로운 요구의 도착으로 인하여 발생하는 데이터 지연 문제(jitter)를 해결하기 위하여 선행 버퍼링이란 기법을 제시한다.

1.서론

멀티미디어 서비스는 여러 형태의 미디어가 다중화되어 사용자에게 서비스되는 것을 말한다. 멀티미디어 중에서 큰 부분을 차지하는 것이 영상과 음성(정확하게는 오디오)이다. 멀티미디어 데이터는 연속되는 미디어의 일부분이므로 정해진 시간 내에 들어와 서비스되어야 한다. 즉 정해진 시간 내에 들어오지 못하면 데이터는 그 의미를 잃는다. 따라서 멀티미디어 데이터에서는 연속성이 중요한다. 연속성이란 데이터를 일정한 시간 간격으로 전송하여 인간이 자연스럽게 영상이나 음성의 변화를 느낄 수 있도록 하는 것이다. 따라서 데이터의 블록이 정해진 시간 내에 도착하지 않으면 멀티미디어 서비스에 문제가 생기고 영상의 경우 자연스러운 움직임이 깨지게 된다. 이런 데이터의 지연으로 인하여 생기는 화면이나 음성의 깨짐을 "jitter"라고 한다. 멀티미디어를 지원하는 멀티미디어 시스템(저장장치, 네트워크, 기타 등등)에서도 연속성이 중요하다. 데이터가 저장되어 있는 디스크에서부터 사용자가 보는 디스플레이까지 데이터의 모든 경로에 있는 장치들(저장장치, 네트워크, 기타 등등)은 연속성을 보장하기 위하여 일정한 속도이상의 전송 대역폭(bandwidth)을 가져야 한다. 연속성 보장은 시스템 중에서 가장 낮은 대역폭을 가진 장치들에 의하여 제공할 수 있는 시스템 대역폭이 결정된다.

90년대에 구역분할 디스크(zoned disk) 가 제안되었다. 디스크의 구역분할(zoning)의 개념은 크게 두 가지로 나누어 생각할 수 있다. 바깥쪽 실린더로 가면서 특성이 비슷한 실린더를 구역으로 묶는 것과 이런 구역 내의 실린더의 크기가 충분히 크다면 실린더를 더 많은 섹터로 나누어 실린더의 크기를 늘리는 것이다. 디스크는 기하학적인 형태가 원형이므로 바깥쪽 실린더로 갈수록 물리적인 면적이 늘어난다. 디스크의 회전속도는 일정하므로 바깥쪽 실린더는 그 선속도도 증가한다. 구역분할 디스크가 제안되기 전에는 각 실린더의 섹터의 수가 동일하였다. 따라서 바깥쪽 실린더는 더 빠른 선속도를 가지고 있으면서도 이를 이용하지 못하였다. 구역분할 디스크가 제안되면서 바깥쪽 구역의 실린더들이 더 많은 섹터를 가지면서 저장용량과 전송 대역폭의 장점을 사용할 수 있게 되었다. 디스크 전체적으로는 저장용량의 증가와 평균 전송 대역폭의 증가를 가져왔다.

멀티미디어 서비스를 제공하기 위해서는 일정한 전송속도를 보장해야 하므로 구역 분할 기술로 인한 각 구역의 전송속도의 차이는 멀티미디어를 지원하는 저장장치로는 적합하지 않으며 이를 지원하기 위해서는 특별한 방법을 사용해야 한다. 그렇지 않으면 가장 안쪽 구역의 전송속도로 저장장치 전체의 전송속도가 결정되어 전송 대역폭의 낭비를 가져온다. 따라서 데이터를 바깥쪽 구역에서 안쪽 구역으로 순환하는 방식으로 배치하고 데이터를 읽어 들이는 디스크 스케줄링을 SCAN 알고리즘을 사용하는 방법이 제시되었다 **[GKS96]**. 이를 이용하면 구역분할 디스크의 특성을 멀티미디어를 지원하는 저장장치에서도 사용할 수 있다.

데이터 블록의 순환적 배치 때문에 저장장치에서 읽어 들이는 데이터 블록의 순서와 사용자에게 서비스되는 순서가 다르다. 이중 버퍼링(double buffering) 을 사용하여 이 문제를 해결한다. 이중 버퍼링은 데이터를 읽어 들이고 데이터를 버퍼링 하였다가 다음 주기에 서비스하는 것이다. 이 주기의 불일치성 때문에 새로운 스트림의 요청이 발생하여 주기시간을 증가시키면 데이터의 지연(jitter)이 발생한다.

본 논문에서는 구역분할 디스크를 이용하는 비디오 서버에서 새로운 요구의 도착으로 인하여 발생하는 데이터 지연 문제(jitter)를 해결하는 첨단 기법을 버퍼링 측면에서 제안한다.

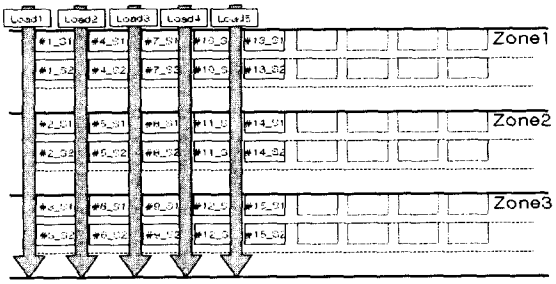
2.새로운 스트림 요청에 의한 데이터 지연 문제

2.1주기시간의 증가

구역분할 디스크는 각 구역의 특성에 따라 전송 대역폭이 다르다. 멀티미디어를 지원하는 저장장치는 일정한 전송 대역폭을 필요로 하므로 특별한 방법을 사용하지 않고는 구역분할 디스크를 사용할 수 없다. **[GKS96]**에서는 구역분할 디스크에서의 각 구역의 전송 대역폭의 차이로 인한 문제를 줄이기 위하여 각 구역을 순환하면서 데이터 블록을 배치시키고 이를 SCAN 스케쥴링을 사용하여 읽어 들인다. **[GKS96]**에서 제안한 방식은 SCAN 스케쥴링을 사용하므로 이중 버퍼링이 사용된다. 새로운 스트림의 요청이 들어오면 주기시간이 늘어나게 되는데 주기시간은 현재의 서비스하고 있는 스트림의 개수에 의해 결정되고 서비스하고 있는 스트림의 개수가 늘어나게 되면 주기시간도 증가한다. 이는 주기시간 내에서 서비스하고 있는 스트림이 늘어나면 동일한 주기시간 내에 더 많은 데이터를 필요로 하는데 저장장치에서의 연속성을 보장하기 위

하여 주기시간을 늘여야 하기 때문이다. 세부적으로는 데이터 블록을 읽어 들여 전송하는데 걸리는 시간(transfer time)과 데이터 블록을 찾는데 걸리는 탐색시간(seek time)이 늘어나게 된다. 이 같은 이유로 새로운 스트림 요청이 들어오면 주기시간이 증가하게 된다. 이중 버퍼링의 문제는 주기시간이 늘어날 때 주기의 불일치성으로 인하여 데이터 지연문제(jitter)를 발생시키는 것이다. 따라서 SCAN 스케줄링을 사용한 멀티미디어를 지원하는 구역분할 디스크에서는 새로운 스트림의 요청이 들어오면 주기시간의 증가로 인하여 데이터 지연문제를 일으킨다.

2.2구역 분할 디스크의 데이터 블록 배치의 특수성



[그림. 1] 블록 배치로 심화되는 데이터 지연문제에 대한 디스크 모델

SCAN 스케줄링을 사용하면 디스크에서 읽어 들여지는 데이터 블록의 순서와 디스플레이로 서비스되는 데이터 블록의 순서가 다르다. 이에 따른 문제는 앞에서 이야기한 이중 버퍼링 방법으로 해결할 수 있다. 새로운 데이터의 요청으로 인한 갑작스러운 주기시간의 증가가 일어나는 과도시간(transition time) 에는 저장장치에서 읽어 들여지는 데이터의 양이 부족하여 데이터의 지연을 발생시킨다. 이중 버퍼링을 사용하므로 다음 주기를 위해 디스크에서 읽어 들이는 데이터가 버퍼에 있다. 읽어 들여지는 데이터 블록의 순서가 순차적이라면 주기시간의 증가로 인한 데이터 지연 효과를 줄일 수 있다.

그러나 본 논문에서 가정한 데이터 블록의 배치 방법은 순환적 배치이다. 데이터 블록의 순환적인 배치는 새로운 스트림의 증가 시에 발생하는 데이터 지연문제를 더욱 크게 만든다. [그림. 1]은 3 개의 구역을 가진 디스크가 2 개의 스트림을 지원하는 디스크 모델이다. 주기시간이 각 스트림에 대하여 6 개의 블록을 읽도록 잡혀 있다면 버퍼로 들어오는 스트림 1 에 대한 블록의 순서는 #1_S1, #4_S1, #2_S1, #5_S1, #3_S1, #6_S1 이다. 이번에는 주기시간이 각 스트림에 대하여 15 개의 블록을 읽도록 잡혀 있다면 블록의 순서가 #1_S1, #4_S1, #7_S1, #10_S1, #13_S1, #2_S1, #5_S1, #8_S1, #11_S1, #14_S1, #3_S1, #6_S1, #9_S1, #12_S1, #15_S1 이 될 것이다. 버퍼링을 할 때에 들어오는 데이터 블록의 순서가 서비스 블록의 순서와 유사하다면 주기시간의 증가로 인한 데이터 지연 효과가 줄어들 수도 있지만 제안된 데이터 블록의 배치가 순환적이기 때문에 주기시간의 증가로 많은 데이터 블록을 읽어 들여야 한다면 데이터 지연 효과는 더욱 커지게 된다.

3.선행 버퍼링(Pre-buffering)을 이용한 데이터지연 방지기법

3.1선행 버퍼링

구역 분할 디스크가 멀티미디어를 지원하기 위하여 데이터 블록을 순환 배치하고 데이터를 SCAN 알고리즘으로 읽어 들이는 방법이 제안되었다. SCAN 스케줄링을 사용하므로 이중 버퍼링(double buffering)을 사용한다. 이중 버퍼링은 정상시간(stationary time) 에는 데이터 지연의 문제를 발생시키지 않는다. 그러나 새로운 스트림의 요청이 들어와 주기시간을 늘려야 하는 과도시간(transition time) 에는 데이터를 읽어 들이는 주기와 서비스 주기의 차이로 인하여 데이터 지연을 발생시킨다. 정상시간(stationary time) 은 버퍼링과 서비스가 주기적으로 수행되어 데이터의 지연이 발생하지 않는 시간을 이야기하고 과도시간(transition time)은 새로운 스트림의 요청이나 스트림의 정지같이 주기시간의 변동으로 버퍼링과 서비스가 주기적으로 수행되지 못하는 시간을 말한다.

선행 버퍼링의 개념은 주기시간의 갑작스러운 증가 시에 발생하는 데이터의 지연을 피하기 위하여 서비스를 시작하기 전에 과도시간에 발생하는 데이터의 부족분에 대한 버퍼링을 하여 과도시간에 발생하는 데이터의 지연을 막는 것이다. 저장장치로 사용하는 디스크의 사양과 데이터 블록의 배치가 결정되면 최대한 서비스할 수 있는 스트림의 개수와 주기시간과 주기시간에 필요한 데이터의 양을 알 수 있다. 따라서 서비스하는 스트림의 개수가 늘어날 때의 데이터 증가량을 계산할 수 있다. 들어오면 주기시간을 두 번째 스트림에 맞도록 증가시킨다. 주기시간이 증가할 때 발생하는 데이터의 지연으로 인한 문제는 선행 버퍼링 데이

멀티미디어 저장장치에서 연속성을 보장하기 위하여 필요한 두 가지 조건이 있다. 첫 번째는 일정주기에 대하여 서비스하는 데이터의 양이 저장장치에서 읽어 들이는 양보다 작아야 한다는 것이고 두 번째는 일정한 데이터에 대하여 서비스 주기가 저장장치의 읽어 들이는 주기보다는 길어야 한다는 것이다. 구역분할 디스크에서의 연속성 보장에 대한 식은 다음과 같다.

$$T \times r_{display} \le n_i \times b_i \quad \text{..... [식. 1]}$$

$$T \ge \sum_{i=1}^{s}\sum_{j=1}^{z}\frac{n_i \times b_i}{Z \times r_j} + \sum T_{seek} + \sum T_{latency} + T_{fullseek} \quad \text{..... [식. 2]}$$

[식. 1] 과 [식. 2] 를 연립하여 풀면 주기시간 T 와 이 주기시간 내에 필요한 데이터 블록의 개수 n_i 를 구할 수 있다.

$$T \ge \frac{\alpha}{1-\left(\frac{r_{display} \times S}{Z} \times \sum_{j=1}^{z}\frac{1}{r_j}\right)} \quad \text{..... [식. 3]}$$

$$n_i \ge \frac{T \times r_{display}}{b_i} \ge \left\{\frac{\alpha}{1-\left(\frac{r_{display} \times S}{Z} \times \sum_{j=1}^{z}\frac{1}{r_j}\right)} \times \frac{r_{display}}{b_i}\right\} \quad \text{..... [식. 4]}$$

[식. 3] 에서 α 는 데이터 전송시간을 제외한 디스크 헤드의 이동에 필요한 시간이다. [식. 4] 에서 구한 n_i 를 [식. 1] 에 대입하여 한 주기에 필요한 데이터의 양을 구할 수 있다.

$$Data_{onescan} = n_i \times b_i \ge \left\{\frac{\alpha}{1-\left(\frac{r_{display} \times S}{Z} \times \sum_{j=1}^{z}\frac{1}{r_j}\right)} \times r_{display}\right\} \quad \text{..... [식. 5]}$$

데이터를 읽어 들이는 방식이 이중 버퍼링이므로 실제의 버퍼의 크기는 [식. 5]의 2 배가 된다.

$$Buffer_{double} = 2 \times \left\{\frac{\alpha}{1-\left(\frac{r_{display} \times S}{Z} \times \sum_{j=1}^{z}\frac{1}{r_j}\right)} \times r_{display}\right\} \quad \text{..... [식. 6]}$$

주기시간의 증가로 인한 데이터 지연 문제를 피하기 위한 선행 버퍼의 크기는 주기시간의 증가량과 관계 있다. 왜냐하면 새로운 스트림의 요청이 들어와 주기시간을 갑자기 늘릴 때 늘어난 주기시간과 이전의 주기시간의 차이만큼의 데이터가 부족하여 문제(jitter)를 발생시키기 때문이다. 따라서 최대로 서비스 할 수 있는 스트림의 주기시간과 현재 스트림의 주기시간과의 차이만큼의 데이터를 서비스하기 전에 선행 버퍼링하면 된다.

$$Buffer_{pre} = \Delta T \times r_{display} \quad \text{..... [식. 7]}$$

[식. 7]은 선행 버퍼의 크기를 구하는 식이다. 따라서 각 스트림이 서비스하기 전에 버퍼링을 해야 하는 총 버퍼의 크기는 스트림의 선행 버퍼와 스트림이 서비스하기 위한 이중 버퍼를 합한 것이다.

$$Buffer_{total;i} = Buffer_{pre;i} + Buffer_{double;i} \quad \text{..... [식. 8]}$$

$$Buffer_{total;i} = \left\{(T_{\max} - T_i) \times r_{display}\right\} + \left\{2 \times T_i \times r_{display}\right\} = (T_{\max} + T_i) \times r_{display} \quad \text{..... [식. 9]}$$

[식. 9] 에 의하여 구한 스트림의 총 버퍼량은 서비스할 수 있는 최대의 스트림일 때의 각 스트림의 버퍼의 크기와 각 스트림이 서비스할 때의 버퍼 크기의 합으로 표시된다. 여기서는 버퍼의 크기가 이중 버퍼링을 이야기하는 것이 아니라 단지 한 주기시간에 필요한 데이터의 양을 이야기한다.

3.2선행 버퍼링의 일반적인 동작

저장장치가 지원하는 서비스 스트림이 없는 상태에서 첫 번째 스트림의 요청이 들어오면 [식. 9] 에 의하여 구한 버퍼량만큼 서비스하기 전에 버퍼링을 하게 된다. 이 때에는 서비스하고 있는 스트림이 없으므로 한 번에 데이터를 읽어 들일 수 있다. 선행 버퍼링을 하는 과도시간을 마치고 나면 주기시간을 주기로 데이터의 읽기와 서비스를 하게 된다.

첫 번째 스트림의 정상적인 서비스 중에 두 번째 스트림의 요청이 지연시간이 줄어들게 된다. 그러나 두 번째 그룹(스트림 #4, #5,

증가할 때 발생하는 데이터의 지연으로 인한 문제는 선행 버퍼링 데이터를 사용하여 피할 수 있다. 두 번째 스트림부터는 서비스하고 있는 스트림이 있으므로 한 번에 선행 버퍼링을 할 수 없다. 따라서 주기시간 내에서 자신에게 할당된 데이터를 읽는 시간에 읽어 들일 수 있는 데이터의 양만큼 여러 번에 걸쳐 선행 버퍼링을 하게 된다. 이 후에 들어오는 스트림들도 같은 방법으로 선행 버퍼링을 하여 주기시간의 증가로 인한 데이터 지연 문제를 피할 수 있다.

3.3스트림의 서비스 중지

새로운 스트림을 서비스하려고 할 때 그 스트림의 선행 버퍼의 크기는 자신 이후에 서비스할 수 있는 스트림의 개수와 관계있다. 데이터 지연이 새로운 스트림의 요청이 있을 때 발생하기 때문이다. 디스크가 서비스하고 있는 스트림이 적을 때에 요청을 한 스트림은 선행 버퍼링을 많이 할 것이고 디스크가 서비스하고 있는 스트림이 많을 때 들어온 스트림은 선행 버퍼링을 적게 할 것이다. 그러나 정상시간 중에서의 어느 순간의 모든 스트림이 가지는 선행 버퍼의 양은 모두 같다. 이는 모든 스트림에게 앞으로 일어나는 주기시간의 증가와 데이터 지연 효과는 같기 때문이다.

여러 개의 스트림을 서비스하고 있는 저장장치를 생각해 보자. 앞에서 이야기한 것같이 정상시간에서의 선행 버퍼량은 모두 같다. 즉 현재 서비스하고 있는 스트림은 앞으로 들어올 수 있는 스트림의 개수만큼의 선행 버퍼 데이터를 모두 똑같이 가지고 있는 것이다. 이런 상황에서 디스크로부터 데이터를 읽어 들이고 있는 스트림의 중지가 발생하면 디스크가 지원할 수 있는 스트림의 개수는 증가하지 않는다. 여기서의 서비스를 할 수 있는 스트림의 의미는 충분한 선행 버퍼를 가지고 있어 자신보다 후에 들어오는 스트림에 의하여 생기는 주기시간의 증가로 인한 데이터 지연 문제(jitter)를 피할 수 있는 것을 말한다. 왜냐하면 서비스를 하던 스트림이 중지를 하여도 디스크가 지원하고 있는 모든 스트림의 선행 버퍼링 데이터의 양은 스트림의 중지가 일어나기 전의 상태로 선행 버퍼링이 되어 있으므로 스트림의 중지가 발생하여 디스크가 지원하고 있는 스트림의 개수가 줄어들어도 앞으로 서비스할 수 있는 스트림의 개수는 증가하지 않는다. 따라서 어떤 서비스 스트림이 중지가 되면 디스크가 앞으로 서비스할 수 있는 스트림의 개수를 증가시키기 위하여 나머지 서비스 스트림의 선행 버퍼링 데이터를 증가시켜 주어야 한다. 이를 위하여 서비스 스트림의 중지가 발생하고 나면 재선행 버퍼링을 해야 한다.

3.4재선행 버퍼링

서비스 스트림이 중지되면 이 스트림을 서비스할 수 있는 스트림으로 만들기 위하여 나머지 서비스하고 있는 스트림들은 재선행 버퍼링을 하게 된다. 스트림의 중지가 발생하여도 주기시간을 줄이지 않고 중지된 스트림에게 할당되었던 데이터의 읽는 시간에 기존의 서비스 스트림들의 재선행 버퍼링을 한다. 재선행 버퍼링을 하는 과도주기가 끝나고 나면 주기시간을 줄이고 정상상태의 주기로 들어가게 된다.

재선행 버퍼링은 읽는 순서에 따라 두 가지 방법으로 나눌 수 있다. 모든 서비스 스트림의 재선행 버퍼링을 다같이 조금씩 하는 방식과 서비스 스트림에 우선순위를 주어 하나씩 재선행 버퍼링을 하는 방식이다. 재선행 버퍼링을 하는 과도주기 동안에 여러 개의 새로운 스트림의 요청이 들어와 완전히 선행 버퍼링이 되지 않은 버퍼 데이터보다 커지면 데이터의 지연이 발생하게 되는데 첫 번째 방식은 모든 서비스 스트림에 데이터의 지연이 발생하는 반면 두 번째 방식은 우선순위가 낮은 서비스 스트림에 더 많은 데이터 지연이 발생한다.

4.선행 버퍼링

4.1그룹기반 선행 버퍼링

디스크가 지원하는 스트림이 적을 때 서비스 요청한 스트림은 선행 버퍼의 크기가 크다. 이런 스트림은 많은 서비스 지연시간을 필요로 한다. 사용자가 적을 때에도 최대 스트림을 목표로 선행 버퍼링을 수행한다면 서버의 부하가 적음에도 불구하고 사용자에게는 긴 서비스 지연시간이 주어지게 된다. 그룹기반 선행 버퍼링은 항상 최대 스트림을 목표로 선행 버퍼링을 할 때의 긴 서비스 지연 시간을 줄이기 위하여 제안된 것이다. 그룹기반 선행 버퍼링은 기존의 선행 버퍼링과 유사하고 단지 두 개 내지 세 개의 스트림마다 선행 버퍼링을 수행하는 것이다. 예를 들어 모두 6 개의 스트림을 지원할 수 있는 저장장치에서 첫 번째 스트림에서부터 여섯 번째 스트림을 지원하기 위하여 선행 버퍼링을 하는 것이 아니라 3 개의 스트림마다 선행 버퍼링을 수행하는 것이다. 이렇게 하면 첫 번째 그룹(스트림 #1, #2, #3) 에 속한 스트림의 선행 버퍼링 #6)의 스트림이 들어오면 두 번째 그룹의 스트림이 선행 버퍼링을 해야 하고 또한 첫 번째 그룹의 스트림도 재선행 버퍼링을 해야 한다. 따라서 두 번째 그룹이 시작되는 스트림 #4 의 경우에는 긴 지연시간이 발생하게 된다. 그러나 디스크가 지원하는 스트림이 적을 때에 서비스를 시작하는 스트림에서 발생하는 긴 서비스 지연시간을 줄일 수 있다. 또한 저장장치의 평균 서비스 스트림의 개수가 최대가 아니라고 한다면 최대 스트림을 위한 버퍼와 지연시간을 소비할 필요가 없을 것이다.

4.2점증적 선행 버퍼링(Incremental pre -buffering)

점증적 선행 버퍼링(incremental pre -buffering)은 선행 버퍼링을 서비스를 하면서 동시에 수행하는 방법이다. 기존의 방법들은 버퍼를 최소화하기 위하여 주기시간을 연속성을 보장하는 최소의 범위로 잡아 주기에 필요한 데이터를 읽어오고 서비스하는 형태였다. 주기시간을 최소시간보다 크게 잡는다면 디스크에서 읽어 들이는 데이터 양과 디스플레이에서 소모하는 데이터 양 사이의 차이로 인하여 매 주기마다 버퍼에 데이터가 남을 것이고 이 데이터의 양은 주기가 지날수록 증가하게 된다. 점증적 선행 버퍼링은 이 누적되는 데이터를 선행 버퍼로 이용하는 것이다. 점증적 선행 버퍼링에서는 주기시간을 크게 잡아 선행 버퍼링을 수행하는 시간을 과도시간(transition time) 이라고 하고 주기시간을 최소값으로 줄여 누적 데이터 없이 주기적으로 버퍼링과 서비스를 하는 시간을 정상시간(stationary time) 이라고 한다. 이 방식은 선행 버퍼링을 서비스와 동시에 수행하므로 별도의 서비스 지연시간을 필요로 하지 않는다. 그러나 과도주기가 서비스를 하는 동안에 분포되어 있어 이 과도주기에 새로운 스트림 요청이 들어오면 기존의 스트림에 데이터의 지연을 발생시키거나 새로운 스트림 요청을 거절하여야 한다. 스트림 요청이 특정한 시간범위 내에 들어오게 되는 확률을 가지고 있다면 주기시간을 조정하여 최적의 과도주기를 가지게 할 수 있다. 주기시간이 길어지면 과도주기는 줄어들지만 버퍼의 크기가 증가하게 되고 주기시간이 줄어들면 과도시간은 늘어나지만 버퍼의 크기는 줄어들게 된다.

점증적 선행 버퍼링의 특별한 경우로 주기시간이 최대 서비스 스트림일 때의 시간일 경우가 있다. 이 경우에는 주기시간이 항상 고정되어 있고 적은 수의 스트림을 지원할 경우에는 디스크의 헤드가 읽어 들이고 나서 쉬게 된다. 즉 주기시간 중에서 자신의 할당시간(time slot) 만을 사용하고 나머지 시간은 관계하지 않는 것과 같다. 이런 경우에는 모든 스트림에 대하여 지연시간과 버퍼의 크기가 크게 변화하지 않고 일정하게 유지된다.

5.결론 및 앞으로의 연구 방향

구역 분할 디스크에서 효율을 증가시키기 위하여 **[GKS96]**에서 제시한 방법은 데이터 블록의 순환적 배치와 SCAN 스케줄링을 사용하는 방법이 였다. SCAN 스케줄링은 이중 버퍼링을 사용해야 한다. 이중 버퍼링은 디스크에서 데이터를 읽어 들이는 주기와 디스플레이가 서비스하는 주기가 다른 주기의 불일치성 때문에 새로운 스트림의 요청으로 주기시간이 증가하면 데이터의 지연을 발생시킨다.

본 논문에서는 새로운 스트림의 요청에 따른 데이터 지연 문제를 해결하기 위하여 선행 버퍼링을 제시하였다. 선행 버퍼링은 서비스 전에 데이터 지연을 막기 위하여 미리 버퍼링을 하는 기법이다. 기본적인 선행 버퍼링의 단점을 보완한 선행 버퍼링 기법은 그룹기반 버퍼링과 점증적 선행 버퍼링이 있다. 그룹기반 선행 버퍼링은 스트림을 여러 개의 그룹으로 나누어 선행 버퍼링을 수행하는 것이고 점증적 선행 버퍼링은 주기시간을 크게 잡아 서비스와 같이 선행 버퍼링을 수행하는 것이다.

그룹기반 선행 버퍼링과 점증적 선행 버퍼링의 수식적인 해석과 최적의 선행 버퍼링 방법에 대하여 계속 연구 중에 있다.

6.Reference

[GKS96] **S. Ghandeharizadeh, S. H. Kim, and C. Shahabi, "*Continuous Display of Video Objects Using Multi -Zone Disks*", Second International Baltic Workshop on DB and IS, June 1996.**

[KS93] **Deepak Kenchammana-Hosekote and Jaideep Srivastava. *Scheduling continuous media in a Video -On-Demand server*. Technical Report TR93-75, Dept. Of Computer Science, University of Minnesota, November 1993.**