# 주문형 비디오 시스템에서의 동적 버퍼 할당 기법
## (A Dynamic Buffer Allocation Scheme in Video-on-Demand Systems)

이 상 호 † 문 양 세 † 황 규 영 †† 조 완 섭 †††
(Sang-Ho Lee) (Yang-Sae Moon) (Kyu-Young Whang) (Wan-Sup Cho)

**요 약** 주문형 비디오 시스템에서 초기대기시간과 메모리 요구량의 최소화는 중요하다. 초기대기시간의 최소화는 빠른 응답 시간의 서비스를 제공할 수 있게 하고, 메모리 요구량의 최소화는 동일한 메모리량으로 더 많은 동시 사용자 요청을 서비스할 수 있게 한다. 주문형 비디오 시스템에서는 사용자 요청에게 할당되는 버퍼의 크기가 증가함에 따라 초기대기시간과 메모리 요구량이 증가하므로 사용자 요청에게 할당되는 버퍼의 크기를 최소화해야 한다. 그러나, 기존의 정적 버퍼 할당 기법은 시스템이 완전 부하된 상태에 있다는 가정하에서 버퍼 크기를 결정하여 시스템이 불완전 부하인 경우에는 사용자 요청에게 필요 이상으로 큰 버퍼를 할당한다. 본 논문에서는 시스템의 완전 부하 상태 뿐만 아니라 불완전 부하 상태에서도 사용자 요청에게 최소 크기의 버퍼를 할당하는 동적 버퍼 할당 기법을 제안한다. 동적 버퍼 할당 기법은 서비스 중인 사용자 요청 수와 이들 요청들을 서비스하는 동안에 도착하는 사용자 요청 수에 기반하여 버퍼 크기를 동적으로 결정한다. 또한, 분석과 시뮬레이션을 통하여 동적 버퍼 할당 기법이 초기대기시간과 지원 가능한 동시 사용자 요청 수에 있어서 정적 버퍼 할당 기법에 비해 크게 우수함을 보인다. 시뮬레이션 결과, 동적 버퍼 할당 기법이 정적 버퍼 할당 기법에 비해 평균 초기대기시간을 29% ~ 65% 줄이고, 다수의 디스크들로 구성된 시스템에서는 서비스한 평균 동시 사용자 요청 수를 48% ~ 68% 증가시킨 것으로 나타났다. 이와 같은 결과는 동적 버퍼 할당 기법이 주문형 비디오 시스템의 성능과 용량을 크게 향상시킴을 보여주는 것이다.

***Abstract*** In video-on-demand (VOD) systems it is important to minimize initial latency and memory requirements. The minimization of initial latency enables the system to provide services with short response time, and the minimization of memory requirements enables the system to service more concurrent user requests with the same amount of memory. In VOD systems, since initial latency and memory requirement increase according to the increment of buffer size allocated to user requests, the buffer size allocated to user requests must be minimized. The existing static buffer allocation scheme, however, determines the buffer size based on the assumption that the system is in the fully loaded state. Thus, when the system is in a partially loaded state, the scheme allocates user requests unnecessarily large buffers. This paper proposes a dynamic buffer allocation scheme that allocates user requests the minimum buffer size in the fully loaded state as well as a partially loaded state. This scheme dynamically determines the buffer size based on the number of user requests in service and the number of user requests arriving while servicing current requests. In addition, through analyses and simulations, this paper validates that the dynamic buffer allocation outperforms the static buffer allocation in initial latency and the number of concurrent user requests that can be supported. Our simulation results show that, in proportion to the static buffer allocation scheme, the dynamic buffer allocation scheme reduces the average initial latency by 29% ~ 65%, and, in a system having several disks, increases the average number of concurrent user requests by 48% ~ 68%. Our results show that the dynamic buffer allocation scheme significantly improves the performance and reduce the capacity requirements of VOD systems.

• 본 연구는 청단정보기술연구센터를 통하여 한국과학재단의 지원을 받았음
† 비 회 원 : 한국과학기술원 전자전산학과 전산학전공
sangho@mozart.kaist.ac.kr
ysmoon@mozart.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 전산학전공 교수
kywhang@cs.kaist.ac.kr
††† 정 회 원 : 충북대학교 경영정보학과 교수
wscho@trut.chungbuk.ac.kr
논문접수 : 2000년 1월 31일
심사완료 : 2001년 7월 19일

## 1. 서 론

최근의 통신 기술과 비디오 데이타의 압축 및 디지털화 기술의 발전으로 네트워크를 통한 비디오 데이타의 송수신이 가능하게 되었다. 이러한 기술은 주문형 비디오, 온라인 지침서(on-line tutorial), 그리고 비디오 게임 등과 같은 응용에 널리 사용되고 있다.

주문형 비디오 시스템(video-on-demand system, 간단히 *VOD* **시스템**이라 한다)은 사용자 요청(user request)에 따라 비디오 데이타를 사용자에게 제공하는 시스템이다. 비디오 데이타의 특성으로는 데이타의 양이 크다는 대용량성과 사용자에게 데이타가 연속적으로 제공되어야 한다는 시간적 연속성이 있다. VOD 시스템은 비디오 데이타의 대용량성으로 인해 여러 개의 디스크를 사용하여 데이타를 저장할 수 있다. 또한, 시스템은 대용량의 비디오 데이타 전체를 메모리에 유지할 수 없기 때문에 데이타를 블록 단위로 관리하는 버퍼 관리가 필요하다. VOD 시스템의 버퍼 관리에서는 비디오 데이타의 시간적 연속성을 보장하기 위하여 사용자 요청이 버퍼에 있는 데이타 블록을 모두 사용하기 전에 다음 데이타 블록을 버퍼로 읽어 들여야 한다.

VOD 시스템의 버퍼 관리에서는 초기대기시간(initial latency)과 메모리 요구량을 최소화하는 것이 중요하다. **초기대기시간**은 사용자 요청이 시스템에 도착한 시점부터 시스템이 사용자 요청에게 비디오 데이타를 제공하기 시작하는 시점까지의 시간 간격이다[2]. 이러한 초기대기시간을 최소화하면, 시스템은 사용자 요청에게 일반 서비스 뿐만 아니라, VCR 기능을 빠른 응답 시간으로 제공하여 서비스의 질을 향상시킬 수 있다. VCR 기능은 빨리 보내기(fast forward)나 빨리 뒤로 보내기(fast backward)와 같은 기능을 말하며, 이들은 대부분의 VOD 시스템에서 새로운 사용자 요청으로 처리한다[2, 13, 14, 15]. 또한, 버퍼 관리에서의 메모리 요구량이 작을 수록 동일한 메모리량으로 더 많은 사용자 요청을 동시에 지원할 수 있다.

VOD 시스템의 버퍼 관리에 관한 기존 연구에서는 메모리 요구량과 초기대기시간을 줄이기 위해 여러 버퍼 스케줄링 방식이 제안되었다[1, 2, 5, 15, 17]. 버퍼 스케줄링 방식은 사용자 요청에게 할당된 버퍼에 데이타를 채우는 순서를 결정한다. 기존의 버퍼 스케줄링 방식들은 각 사용자 요청에게 버퍼를 할당하기 위해 **정적 버퍼 할당 기법**(*static buffer allocation scheme*)을 사용하였다. 정적 버퍼 할당 기법은 시스템이 지원 가능한 최대수의 사용자 요청을 서비스하는 완전 부하(full load) 상태에서 각 사용자 요청이 필요로 하는 최소 버퍼 크기를 구하고, 이를 시스템의 부하에 관계없이 모든 사용자 요청에게 동일하게 할당하는 방법이다. 따라서, 이 기법은 시스템이 불완전 부하 상태에 있는 경우에도 불필요하게 큰 버퍼를 할당하여 메모리를 비효율적으로 사용하게 되며, 결과적으로는 시스템의 평균 초기대기시간과 메모리 요구량을 증가시킨다[1, 2, 4].

To와 Hamidzadeh[19]는 최근에 정적 버퍼 할당 기법의 비효율적인 메모리 사용을 개선하는 기법을 제안하였다. 이 기법은 시스템이 불완전 부하 상태인 경우에 사용하지 않는 메모리를 서비스 중인 사용자 요청들에게 할당하여 시스템 메모리를 모두 활용할 수 있도록 한다. 이 기법은 서비스 중인 사용자 요청들에게 더 많은 메모리를 할당하므로 이들 요청들이 서비스를 받아야 하는 시간을 더 지연시킬 수 있고, 이 시간동안 새로 도착한 사용자 요청을 서비스할 수 있기 때문에 사용자 요청의 초기대기시간을 줄일 수 있다[19]. 그러나, 이 기법도 정적 버퍼 할당 기법을 기반으로 하여 초기 버퍼 크기를 구하므로 정적 버퍼 할당 기법에서와 같이 불필요하게 큰 버퍼를 할당하는 문제점이 있다.

본 논문에서는 시스템의 완전 부하 상태 뿐만 아니라 불완전 부하 상태에서도 사용자 요청에게 최소 크기의 버퍼를 할당하는 **동적 버퍼 할당 기법**(*dynamic buffer allocation scheme*)을 제안한다. 동적 버퍼 할당 기법은 서비스 중인 사용자 요청 수와 이들 요청들을 서비스하는 동안에 도착하는 사용자 요청 수를 고려하여 각 사용자 요청에게 할당할 버퍼의 최소 크기를 동적으로 결정함으로써 불완전 부하 상태에서 필요이상으로 큰 버퍼를 할당하는 정적 버퍼 할당 기법의 문제점을 해결한다. 그리고, 이를 통하여 평균 초기대기시간과 지원 가능한 평균 동시 사용자 요청 수를 정적 버퍼 할당 기법에 비해 크게 개선할 수 있다. 또한, 동적 버퍼 할당 기법은 버퍼 스케줄링 방식에 독립적인 것이기 때문에 기존의 모든 버퍼 스케줄링 방식에 적용할 수 있다. 본 논문에서는 이를 보이기 위해 제안한 동적 버퍼 할당 기법을 대표적인 버퍼 스케줄링 방식인 라운드 로빈 방식, 스윕 방식, 그리고 GSS 방식에 적용한다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 VOD 시스템의 모델에 관한 관련 연구를 설명한다. 제 3 장에서는 본 논문에서 제안하는 동적 버퍼 할당 기법의 버퍼 할당 알고리즘과 버퍼 크기 결정 방법, 그리고 서비스 주기내에 도착할 사용자 요청 수를 결정하는 방법에 관해서 기술한다. 제 4 장에서는 동적 버퍼 할당 기법의 메모리 요구량을 분석한다. 제 5 장에서는 정적 버퍼 할

당 기법과 동적 버퍼 할당 기법의 초기대기시간과 지원 가능한 동시 사용자 요청 수를 비교 평가한다. 마지막으로, 제 6 장에서 결론을 맺는다.

## 2. 관련 연구

본 장에서는 관련 연구로서 VOD 시스템의 모델, 버퍼 관리에서 사용하는 기존 버퍼 스케줄링 방식, 그리고 정적 버퍼 할당 기법에 관해서 기술한다.

### 2.1 주문형 비디오 시스템 모델

VOD 시스템의 기본 아키텍춰는 그림 1과 같다. 그림에서와 같이 VOD 시스템은 비디오 데이타를 저장한 디스크, 각 사용자 요청에게 할당된 버퍼, 그리고 디스크로부터 데이타를 읽어 버퍼를 채우는 서버로 구성된다. VOD 시스템에서 서버가 디스크로부터 데이타를 읽어 버퍼를 채우는 행위를 **서비스**라 하고, 서비스 중인 모든 버퍼에 비디오 데이타를 한번씩 채우는 시간 간격을 **서비스 주기**라 한다. 그리고, 버퍼 할당 시점으로부터 서비스 주기내에 도착하는 사용자 요청을 **추가 요청**이라 정의한다. 예를 들어, 그림 2에서와 같이 버퍼 할당 시점 $t_1$으로부터 서비스 주기 $T$내에 도착하는 사용자 요청 $R_1$, $R_2$, 그리고 $R_3$가 추가 요청이다. 또한, 각 사용자 요청이 버퍼로부터 비디오 데이타를 가져가는 속도를 **비디오 데이타 소비 속도**라 하고, 디스크의 탐색 시간과 회전 지연 시간을 **디스크 지연 시간**(*disk latency*)[3]이라 한다.

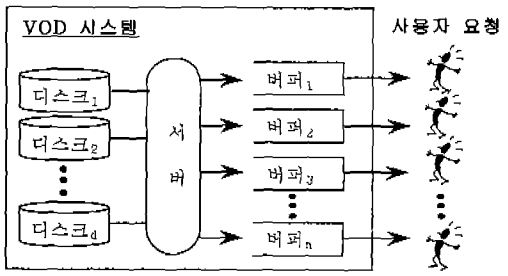
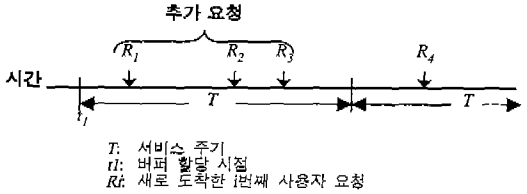



그림 1 VOD 시스템의 기본 아키텍춰




T: 서비스 주기
$t_1$: 버퍼 할당 시점
$R_i$: 새로 도착한 i번째 사용자 요청

그림 2 추가 요청

VOD 시스템의 서버는 시스템에 도착한 각 사용자 요청에 하나의 버퍼를 할당하고, 할당된 버퍼에 비디오 데이타를 주기적으로 채움으로써 사용자에게 연속적으로 비디오 데이타를 제공한다. 서버가 버퍼들에게 데이타를 채우는 순서는 버퍼 스케줄링 방식에 의해서 결정되며, 대표적인 버퍼 스케줄링 방식에는 라운드 로빈(round-robin) 방식, 스윕(sweep) 방식, 그리고 GSS (grouped sweeping scheduling) 방식이 있다. 라운드 로빈 방식은 각 버퍼를 할당된 순서로 서비스하는 방식이며[1, 2, 3], 스윕 방식은 비디오 데이타가 디스크상에 위치한 순서로 서비스를 하여 디스크의 탐색 시간(seek time)을 최소화하는 방식이다[1, 2, 3]. 그리고, GSS 방식은 버퍼들을 여러 그룹으로 나누어 각 그룹내의 버퍼들은 스윕 방식으로 서비스하고, 각 그룹들은 라운드 로빈 방식으로 서비스하는 방식이다[5].

각 사용자 요청에게 할당된 버퍼들은 시스템의 메모리 요구량을 줄이기 위해서 메모리를 공유한다. 즉, 사용자 요청들은 버퍼에 있는 데이타를 사용한 후에는 바로 해당 메모리를 반환하고, 서버는 반환된 메모리를 사용하여 다른 사용자 요청에게 버퍼를 할당한다[1, 22]. 메모리는 페이지 단위로 할당하고 반환하며, 이는 메모리 공유로 인해 발생할 수 있는 메모리 단편화(fragmentation) 현상을 제거한다. 그러나, 본 논문에서는 설명의 편의를 위하여 메모리가 페이지 단위가 아닌 가변길이 단위로 할당되고 반환된다고 가정한다[1]. 일반적으로, 마지막 메모리 페이지의 활용도는 100%이하이므로 이 가정하에서 유도된 결과는 실제와 다를 수 있으나, 메모리 페이지가 버퍼 크기에 비해 훨씬 작기 때문에 이 가정으로 인한 차이는 무시할 수 있다[1].

본 논문에서는 설명의 편의를 위해서 모든 사용자 요청의 비디오 데이타 소비 속도가 동일하다고 가정한다[1]. 그리고, 디스크 지연 시간을 줄이기 위해서 비디오 데이타는 디스크에 연속적으로 저장된다고 가정한다[1][2, 19]. 따라서, 하나의 버퍼를 서비스하는 데에는 한번의 디스크 지연만이 발생한다.

본 논문에서 사용하는 변수들은 표 1과 같다. 표 1에

1) 이 가정을 만족시키기 위해서 chunk라는 데이타 배치 기법이 제안되었다[2]. 일반적으로, 비디오 데이타 전체를 디스크에 연속적으로 저장할 수 없기 때문에 여러 페이지로 구성된 블록에 데이타를 저장한다. 이 경우에 버퍼 크기가 가변적이면 하나의 버퍼가 사용하는 데이타가 이웃한 두 블록에 걸쳐 저장될 수 있다. 이 문제를 해결하기 위해 chunk에서는 이웃한 블록들이 최대 버퍼 크기 만큼의 비디오 데이타가 중첩되도록 저장하여, 하나의 버퍼가 사용하는 데이타는 항상 하나의 블록에서 읽을 수 있도록 한다.

서 지원 가능한 최대 동시 사용자 요청 수 $N$은 비디오 데이타 소비 속도 $CR$과 디스크의 전송 속도 $TR$에 의해서 결정된다. 디스크가 시간적 연속성을 보장하면서 $N$개의 사용자 요청들을 서비스하기 위해서는 $TR$이 $N$개의 사용자 요청이 데이타를 소비하는 속도 $N \times CR$보다는 크거나 같아야 한다. 그러나, 디스크가 사용자 요청을 서비스할 때에 항상 디스크 지연이 발생하기 때문에, $TR$과 $N \times CR$이 같으면 디스크는 사용자 요청들에게 시간적 연속성을 보장할 수 없다. 따라서, $TR$은 $N \times CR$보다 커야 하며, 수식(1)을 만족한다. 그리고, $N$은 최대치이므로 수식 (1)을 만족하는 가장 큰 정수가 된다.

$$N < \frac{TR}{CR} \tag{1}$$

표 1 변수 리스트

| 변수명 | 설 명 |
|---|---|
| $TR$ | 디스크의 전송 속도 (bits/sec) |
| $CR$ | 비디오 데이터 소비 속도 (bits/sec) |
| $DL$ | 디스크 지연 시간 |
| $DL^{RR}$ | 라운드 로빈 방식에서의 디스크 지연 시간 |
| $DL^{Sweep}$ | 스윕 방식에서의 디스크 지연 시간 |
| $DL^{GSS}$ | GSS 방식에서의 디스크 지연 시간 |
| $T$ | 서비스 주기 |
| $BS$ | 버퍼 크기 |
| $BS^{RR}$ | 라운드 로빈 방식에서의 버퍼 크기 |
| $BS^{Sweep}$ | 스윕 방식에서의 버퍼 크기 |
| $BS^{GSS}$ | GSS 방식에서의 버퍼 크기 |
| $N$ | 지원 가능한 최대 동시 사용자 요청 수 |
| $N$ | 서비스 중인 사용자 요청 수 |
| $k$ | 추가 요청 수 |

### 2.2 버퍼 스케줄링 방식

본 절에서는 동적 버퍼 할당 기법을 적용할 대표적 버퍼 스케줄링 방식인 라운드 로빈 방식, 스윕 방식, 그리고 GSS 방식의 특성과 각 방식별로 수행된 기존 연구에 관해서 논한다. 여기에서 논하는 각 버퍼 스케줄링 방식의 특성은 초기대기시간과 디스크 지연 시간이다. 초기대기시간은 본 논문에서 연구하는 버퍼 할당 기법의 성능 평가 기준이고, 디스크 지연 시간은 버퍼 크기를 결정하는 중요한 요소이다. 각 사용자 요청에게 할당된 버퍼는 다음 서비스 받을 때까지 사용자 요청이 사용할 데이타를 유지해야 한다. 따라서, 버퍼 크기를 결정하기 위해서는 다음 서비스 받을 때까지의 시간 간격인 서비스 주기를 계산하여야 하며, 이를 위해서는 각 버퍼들을 서비스할 때에 발생하는 디스크 지연 시간을 추정하는 것이 필요하다. 이 추정치가 실제보다 작으면, 작은 크기의 버퍼가 할당되기 때문에 서비스 도중에 버퍼가 비게 된다. 그러므로, VOD 시스템에서는 최악의 경우의 디스크 지연 시간을 사용하여 버퍼 크기를 결정하며, 본 절에서도 각 버퍼 스케줄링 방식별로 최악의 경우에서의 디스크 지연 시간에 관해서 논한다.

2.2.1 라운드 로빈 방식

라운드 로빈 방식에서는 각 버퍼들이 할당된 순으로 서비스 순서를 정한다. 따라서, 이 방식의 디스크 지연 시간은 이전에 서비스된 버퍼가 사용한 데이타와 현재 서비스 중인 버퍼가 사용하는 데이타 사이의 거리를 탐색(seek)하는 디스크 탐색 시간과 회전 지연 시간(rotational delay)의 합이 된다. 이에 따라 최악의 경우의 디스크 지연 시간인 $DL^{RR}$은 디스크의 실린더 수 만큼을 탐색하는 시간과 최대 회전 지연 시간의 합이 된다. 따라서, $x$개의 실린더를 탐색하는데 소요되는 디스크 탐색 시간의 함수를 $(x)$라고 하고, 최대 회전 지연 시간을 , 그리고 디스크의 전체 실린더 수를 $Cyln$라고 하면, $DL^{RR}$은 $(Cyln)+[1]$가 된다. 라운드 로빈 방식은 디스크상의 데이타 위치를 고려하지 않기 때문에 스윕 방식이나 GSS 방식에 비해 디스크 지연 시간이 훨씬 크고, 이로 인해 버퍼 크기도 커져서 더 많은 시스템 메모리를 필요로 한다.

Chang과 Garcia-Molina[1]는 버퍼들의 메모리 공유를 최대화하기 위해서는 각 버퍼들의 서비스 시간을 동일하게 해야 함을 증명하고, 이를 라운드 로빈 방식에 적용하여 Fixed-Stretch 방식이라는 버퍼 스케줄링 기법을 제안하였다. 또한, 초기대기시간을 줄이기 위해서 Fixed-Stretch 방식에 기반을 둔 BubbleUp[2]이라는 버퍼 스케줄링 기법을 제안하였다. Fixed-Stretch 방식이 고정된 순서하에서 버퍼들을 서비스하는 것에 반해, BubbleUp은 버퍼들을 서비스하는 순서를 동적으로 조정하여 새로운 사용자 요청이 도착하면, 현재 수행 중인 서비스가 완료된 후에 바로 서비스할 수 있도록 하였다. 본 논문에서는 동적 버퍼 할당 기법을 적용할 라운드 로빈 방식으로 BubbleUp을 사용한다. BubbleUp의 최악의 초기대기시간 $IT^{RR}$은 수식(2)와 같이 서비스 중인 버퍼를 서비스하는 시간 $DL^{RR}+\frac{BS^{RR}}{TR}$과 새로 도착한 사용자 요청을 서비스하는데 소요되는 디스크 지연 시간 $DL^{RR}$을 합한 시간이다[2].

$$IT^{RR} = 2 \times DL^{RR} + \frac{BS^{RR}}{TR} \tag{2}$$

To와 Hamidzadeh[19]의 기법도 라운드 로빈 방식의 한 종류로서, 이 기법은 서비스 중인 사용자 요청들에게 메모리를 추가로 더 할당하여 현재 서비스 중인 버퍼를

서비스한 후에 새로 도착한 사용자 요청을 가능한 빨리 서비스하여 초기대기시간을 줄인다. 이에 반해, Bubble Up은 서비스 중인 사용자 요청들에게 메모리를 추가로 할당하지 않고도 새로 도착한 사용자 요청을 현재 서비스 중인 버퍼를 서비스한 후에 바로 서비스할 수 있기 때문에 BubbleUp은 To와 Hamidzadeh의 기법에 비해서 초기대기시간이 더 짧다. 이는 현재 서비스 중인 버퍼를 서비스하는 시간이 BubbleUp이 To와 Hamidzadeh의 기법에 비해서 짧기 때문이다.

2.2.2 스윕 방식

스윕 방식[1]은 디스크의 탐색 시간을 줄이기 위해 데이타가 디스크상에 위치한 순서로 버퍼들을 정렬하고, 정렬한 순서에 따라 버퍼들을 서비스하는 방식이다. 따라서, 스윕 방식의 디스크 지연 시간은 서비스 중인 버퍼들이 사용하는 데이타가 존재하는 디스크상의 위치에 따라 다르다. 디스크의 탐색 시간은 탐색할 디스크 실린더 수에 대한 오목 함수(concave function)[7]이기 때문에, 이 방식의 최악의 경우의 디스크 지연 시간은 서비스 중인 $n$개의 버퍼들이 사용하는 데이타들이 동일한 간격으로 떨어져 있는 경우이다[1]. 따라서, 스윕 방식으로 $n$개의 버퍼를 서비스하는데 소요되는 최악의 경우의 디스크 지연 시간은 $n\times(\gamma(Cyln/n)+\theta)$[1]가 된다. 그리고, 설명의 편의를 위하여 하나의 버퍼를 서비스하는데 소요되는 최악의 경우의 디스크 지연 시간 $DL^{Sweep}$을 $(\gamma(Cyln/n)+\theta)$로 정의한다.[2)]

Chang과 Garcia-Molina[1]는 스윕 방식의 메모리 요구량을 줄이기 위해서 버퍼들의 메모리 공유를 증가시킨 스윕*라는 버퍼 스케줄링 방식을 제안하였다. 스윕 방식은 버퍼들의 서비스 순서만을 관리하고 서비스 시간은 관리하지 않기 때문에, 버퍼들이 사용하는 모든 데이타가 디스크상에 인접하게 위치한 경우에는 실제 디스크 지연 시간이 예상 디스크 지연 시간보다 짧아져서 버퍼들에 대한 서비스가 짧은 시간 내에 완료될 수 있다. 이렇게 되면 사용자 요청들이 버퍼의 데이타를 소비할 시간적 여유가 없어진다. 결국, 앞서 서비스 받은 버퍼들이 메모리를 반환하기 이전에 다음 버퍼들이 서비스를 받게 되므로 버퍼들은 메모리를 거의 공유하지 못하고, 이로 인해 메모리 요구량이 증가하게 된다. 이에 반해, 스윕* 방식은 서비스 주기 내에서 마지막에 서비스되는 버퍼를 가능한 늦게 서비스하여 다른 버퍼들이 반환한 메모리를 사용할 수 있도록 한다. 이와 같이 스윕* 방식은 서비스 시간을 일부 조정하여 스윕 방식에 비해서 메모리 공유를 증가시키고, 이를 통하여 메모리 요구량을 줄인다.

스윕* 방식에서는 새로 도착한 사용자 요청을 도착한 시점에서의 서비스 주기 내에서 서비스하지 않는다. 이는 기존 버퍼들을 서비스하는 도중에 새로 도착한 사용자 요청을 서비스하면, 디스크 탐색을 최적화할 수 없기 때문이다. 또한, 스윕* 방식은 버퍼들이 사용하는 데이타의 디스크상에서의 위치에 따라 서비스 순서를 결정하기 때문에, 새로 도착한 사용자 요청이 서비스 주기의 맨 마지막에 서비스될 수 있다. 따라서, 최악의 경우에는 사용자 요청이 서비스 주기의 시작 시점에 도착하고, 다음 서비스 주기의 맨 마지막에 서비스된다. 이때의 초기대기시간 $IT^{Sweep}$은 수식(3)과 같이 $n$개의 서비스 중인 버퍼들을 서비스하는 시간, $n$개의 버퍼들을 다음 서비스 주기내에서 서비스하는 시간, 그리고 새로 도착한 사용자 요청의 버퍼를 서비스하는 시간을 합한 시간이 된다[2].

$$IT^{Sweep}=2\times n\times\left(DL^{Sweep}+\frac{BS^{Weep}}{TR}\right)+DL^{Sweep}+\frac{BS^{Sweep}}{TR} \quad (3)$$

2.2.3 GSS 방식

GSS 방식은 메모리 요구량을 줄이기 위해서 스윕 방식과 라운드 로빈 방식을 접목시킨 버퍼 스케줄링 방식이다[5]. GSS 방식은 $n$개의 사용자 요청을 $G$개의 그룹으로 구성하고, 각 그룹내에 있는 $n/G(=g)$개의 버퍼들은 스윕 방식으로 서비스하며, 각 그룹은 라운드 로빈 방식으로 서비스한다. 따라서, $G=1$인 경우는 스윕 방식이 되고, $G=n$인 경우는 라운드 로빈 방식이 된다. GSS 방식은 메모리 요구량이 가장 적은 경우의 $G$ 값을 구하고[5], 이를 사용하여 버퍼들을 서비스한다. 이 방식에서 $g$개의 버퍼로 구성된 하나의 버퍼 그룹을 서비스하는 경우에 발생하는 최악의 디스크 지연 시간은 스윕 방식과 유사하게 $g\times(\gamma(Cyln/g)+\theta)$로 유도할 수 있고[1], 한 버퍼를 서비스하는데 소요되는 최악의 경우의 디스크 지연 시간인 $DL^{GSS}$은 $(\gamma(Cyln/g)+\theta)$가 된다.

Chang과 Garcia-Molina는 GSS 방식에서 버퍼들의 메모리 공유를 증가시키기 위해 그룹간에는 Fixed-Stretch 방식으로 서비스하고, 그룹내의 버퍼들은 스윕*로 서비스하는 GSS* 방식[1]을 제안하였다. 또한, 이들은 GSS* 방식의 초기대기시간을 줄이기 위해서 그룹들

2) 디스크 지연 시간은 서비스 주기를 계산하기 위해서 사용되기 때문에, 항상 서비스 주기내에 서비스 중인 모든 버퍼들의 디스크 지연 시간을 합한 값을 사용한다. 따라서, 본 논문에서와 같이 $DL^{Sweep}$을 정의하여도 서비스 중인 모든 버퍼들의 디스크 지연 시간을 합한 값에는 변화가 없고, 본 논문에서 유도하는 결과에도 영향을 미치지 않는다. 본 논문에서는 단지 기존의 여러 버퍼 스케줄링 방식을 일관되게 설명하기 위해서 $DL^{Sweep}$을 이와 같이 정의한다.

을 Fixed-Stretch 방식 대신에 BubbleUp[2]으로 서비스하는 방식으로 GSS* 방식[4]을 확장하였다. 본 논문에서는 이와 같이 확장한 GSS* 방식을 동적 버퍼 할당 기법에 적용한다. 이 방식의 최악의 초기대기시간 $IT^{GSS}$는 수식(4)와 같이 현재 서비스 중인 그룹의 서비스 시간과 새로 도착한 사용자 요청이 포함된 다음 그룹의 서비스 시간을 합한 것이다[4].

$$IT^{GSS} = 2 \times g \times \left( DL^{GSS} + \frac{BS^{GSS}}{TR} \right) \quad (4)$$

지금까지 각 버퍼 스케줄링 방식의 초기대기시간, 디스크 지연 시간, 그리고 기존 연구에 관해서 살펴보았다. 수식(2)(3)(4)에서 보는 바와 같이 초기대기시간은 버퍼 스케줄링 방식에 관계없이 버퍼 크기 $BS$에 따라 선형적으로 증가함을 볼 수 있다. 즉, 각 수식의 $DL$, $TR$, 그리고 $g$가 상수이므로 초기대기시간은 오직 버퍼 크기에 따라 결정된다. 따라서, 각 사용자에게 할당되는 버퍼의 크기가 증가하면 시스템의 메모리 요구량 뿐만 아니라, 초기대기시간도 증가한다. 본 연구에서는 버퍼 크기를 최소화하여 메모리 요구량과 초기대기시간을 최소화하고자 한다.

### 2.3 정적 버퍼 할당 기법

정적 버퍼 할당 기법은 완전 부하 상태에서의 최소 버퍼 크기를 구하고, 이를 시스템의 부하 상태에 관계없이 모든 사용자 요청에게 동일하게 할당한다. 따라서, 이 기법은 버퍼 할당이 간단하다는 장점이 있는 반면, 시스템이 불완전 부하 상태에 있는 경우에는 필요 이상으로 큰 버퍼를 할당한다는 단점이 있다.

완전 부하 상태에서의 최소 버퍼 크기는 새로 도착하는 사용자 요청들을 고려할 필요 없이 오직 서비스 중인 사용자 요청들만을 고려하여 구한다. 이는 완전 부하 상태가 된 시스템은 더 이상의 새로운 사용자 요청을 서비스할 수 없기 때문이다. 완전 부하 상태에서 버퍼 크기가 만족해야 할 조건은 다음 조건1과 조건2이다.

**조건1** : 버퍼 크기는 사용자 요청이 서비스 주기동안에 소비하는 데이타량보다 크거나 같아야 한다.

**조건2** : 시스템은 서비스 주기 내에 서비스 중인 모든 사용자 요청들을 한번씩 서비스 할 수 있어야 한다.

조건1은 사용자 요청들에게 비디오 데이타의 시간적 연속성을 보장해 주기 위한 필요 조건이다. 조건1이 만족되지 않으면, 서비스 중인 버퍼가 빌 수 있다. 조건2는 너무 큰 버퍼의 할당으로 인해, 하나의 버퍼를 서비스하는 시간이 너무 많이 걸려 서비스 주기 내에 모든 버퍼들을 서비스할 수 없게 되는 것을 방지하기 위해서 필요하다. 조건1과 조건2를 만족하는 완전 부하 상태에서의 최소 버퍼 크기 $BS(n)$은 수식(5)와 같으며, 이는 참고문헌[1]에 증명되어 있다.

$$BS(n) = \frac{n \times CR \times DL \times TR}{TR - n \times CR} \quad (5)$$

## 3. 동적 버퍼 할당 기법

본 절에서는 현재 서비스 중인 사용자 요청 수와 추가 요청 수에 따라 버퍼 크기를 결정하는 동적 버퍼 할당 기법을 제안한다. 제 3.1절에서는 동적 버퍼 할당 기법의 버퍼 할당 알고리즘에 관해서 설명하고, 제 3.2 절에서는 사용자 요청에게 할당하는 버퍼의 크기를 결정하는 수식을 제시한다. 그리고, 제 3.3 절에서는 버퍼 크기의 결정에 필요한 추가 요청 수를 결정하는 방법을 설명한다.

### 3.1 버퍼 할당 알고리즘

VOD 시스템에서 서버는 각 사용자 요청을 서비스할 때마다 버퍼를 새로 할당한다. 그리고, 동적 버퍼 할당 기법은 최소 크기의 버퍼를 할당하기 위해 실행시간 정보인 '서비스 중인 사용자 요청 수'를 알아야 한다. 따라서, 버퍼 할당 알고리즘은 서비스 중인 사용자 요청 수를 유지하고, 이를 사용하여 각 사용자 요청에게 동적으로 최소 크기의 버퍼를 할당한다. 버퍼 할당 알고리즘은 아래와 같다.

1. 사용하는 버퍼 스케줄링 방식에 따라 서비스할 사용자 요청 $U$를 선택한다.
2. $U$가 서비스가 완료된 사용자 요청이면, 서비스 중인 사용자 요청 수를 하나 감소시키고 스텝1을 수행한다.
3. $U$가 새로 도착한 사용자 요청이면, 서비스 중인 사용자 요청 수를 하나 증가시키고 스텝4를 수행한다.
4. ($U$가 서비스 완료되지 않는 기존 사용자 요청이거나 새로 도착한 사용자 요청이면,) 제3.2절의 정리1에 의하여 동적으로 서비스 중인 사용자 요청 수와 추가 요청 수에 따라 버퍼 크기를 결정하여 $U$에게 버퍼를 할당한다.

위 알고리즘의 스텝1, 2, 3에서는 서비스 중인 사용자 요청 수를 관리하며, 스텝4에서는 동적으로 최소 버퍼 크기를 결정하고, 이를 각 사용자 요청에게 할당한다. 최소 버퍼 크기의 결정 방법은 제3.2절에서 자세히 설명한다. 그리고, 버퍼 크기 결정에서 사용되는 추가 요청 수는 버퍼를 할당받는 시점부터 서비스 주기 내에 새로 도착할 사용자 요청 수이기 때문에 버퍼 할당 시점에서는 정확한 값을 알 수 없다. 따라서, 이는 추정치를 사용해야 하며, 그 방법에 관해서는 제3.3절에서 자세히 논한다.

### 3.2 버퍼 크기 결정

본 절에서는 동적 버퍼 할당 기법에서 버퍼 크기를 결

정하기 위해 사용하는 수식을 유도한다. 버퍼 크기는 제2.3절의 조건1과 조건2 외에 새로 도착할 사용자 요청들을 서비스하기 위해 필요한 조건3을 추가로 만족하여야 한다.

**조건3** : 시스템은 추가 요청들을 현재의 서비스 주기 내에 한번씩 서비스 할 수 있어야 한다.

조건3에서 사용되는 추가 요청 수는 추정치로서, 이를 **예상 추가 요청 수**라 정의하고, 실제로 발생한 추가 요청 수는 **실제 추가 요청 수**라 정의한다. 그리고, 예상 추가 요청 수가 실제 추가 요청 수보다 작은 경우를 **추정 실패**라 정의하고, 그렇지 않은 경우를 **추정 성공**이라 정의한다. 동적 버퍼 할당 기법은 시스템에 새로운 요청(추가 요청)이 도착할 때마다 실제 추가 요청 수가 예상 추가 요청 수보다 크지 않은 지를 검사한다. 이때, 추정 실패가 발생하면, 해당 추가 요청의 서비스를 다음 서비스 주기로 연기한다. 다음 서비스 주기에서는 이전 서비스 주기에서 연기된 추가 요청들과 새로 도착한 추가 요청들을 시스템에 도착한 순서에 따라 다시 이 검사를 수행한다. 즉, 시스템은 서비스가 연기된 요청들과 새로 도착한 추가 요청들을 구분하지 않고 시스템에 도착한 순서에 따라 요청들을 서비스한다. 그리고, 추정실패가 발생하면, 아직 서비스되지 않은 모든 요청들의 서비스를 다음 서비스 주기로 연기한다.

동적 버퍼 할당 기법은 제2.3절의 조건1과 조건2 그리고 위의 조건3을 만족하는 최소 크기의 버퍼를 각 사용자 요청에게 할당한다. 이들 조건을 만족하는 최소 버퍼 크기는 $n$개의 사용자 요청을 서비스 중인 서버가 서비스 주기 내에 $k$개까지의 새로운 사용자 요청을 추가로 서비스하여도 버퍼가 비지 않게 되는 가장 작은 버퍼 크기이다. 시스템이 완전 부하 상태인 경우의 최소 버퍼 크기는 제2.3절의 수식(5)와 같으며, 불완전 부하 상태인 경우의 최소 버퍼 크기는 서비스 주기 내에 최대 $k+n$개까지의 사용자 요청을 서비스할 수 있도록 하는 가장 작은 버퍼 크기로서 이는 정리1과 같다.

**정리1**: 서비스 주기 $T$이내에 도착할 추가 요청 수가 $k$이고, 현재 서비스 중인 사용자 요청 수가 $n$이라 하면, 최소 버퍼 크기 $BS_k(n)$은 수식(6)과 같다.

$$BS_k(n)=\begin{cases} DL\times CR\times\left[\left(\frac{CR}{TR}\right)^{e}\times\prod_{i=1}^{e-1}(n+i\times k)\times\frac{N^2\times TR}{TR-N\times CR}+\right. \\ \left.\sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^{i}\times\prod_{j=1}^{i+1}(n+j\times k)\right\}+\left(\frac{CR}{TR}\right)^{e-1}\times N\times\prod_{j=1}^{e-1}(n+j\times k)\right] & ,n<N \\ DL\times\frac{N\times CR\times TR}{TR-N\times CR} & ,n=N \end{cases} \quad (6)$$

단, 여기서 $e=\left\lceil\frac{N-n}{k}\right\rceil$ 이다.

**증명:** 부록A 참조.

정리1에서 $n < N$인 경우는 불완전 부하 상태에서의 최소 버퍼 크기이고, $n=N$인 경우는 완전 부하 상태에서의 최소 버퍼 크기이다. 위의 정리를 사용하여 동적 버퍼 할당 기법이 각 버퍼 스케줄링 방식에 따라 할당하는 버퍼의 크기는 따름정리1과 같다.

표 2 버퍼 스케줄링 방식에 따른 버퍼 크기

| 버퍼 스케줄링 방식 | $n$개의 사용자 요청을 지원하기 위한 버퍼 크기($BS_k(n)$) | |
|---|---|---|
| | $n<N$ | $n=N$ |
| 라운드 로빈 | $(\gamma(Cyln)+\theta)\times CR\times\left[\left(\frac{CR}{TR}\right)^{e}\times\prod_{i=1}^{e-1}(n+i\times k)\times\frac{N^2\times TR}{TR-N\times CR}+\sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^{i}\times\prod_{j=1}^{i+1}(n+j\times k)\right\}+\left(\frac{CR}{TR}\right)^{e-1}\times N\times\prod_{j=1}^{e-1}(n+j\times k)\right]$ | $(\gamma(Cyln)+\theta)\times\frac{N\times CR\times TR}{TR-N\times CR}$ |
| 스윕* | $(\gamma(Cyln/n)+\theta)\times CR\times\left[\left(\frac{CR}{TR}\right)^{e}\times\prod_{i=1}^{e-1}(n+i\times k)\times\frac{N^2\times TR}{TR-N\times CR}+\sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^{i}\times\prod_{j=1}^{i+1}(n+j\times k)\right\}+\left(\frac{CR}{TR}\right)^{e-1}\times N\times\prod_{j=1}^{e-1}(n+j\times k)\right]$ | $(\gamma(Cyln/n)+\theta)\times\frac{N\times CR\times TR}{TR-N\times CR}$ |
| GSS* | $(\gamma(Cyln/g)+\theta)\times CR\times\left[\left(\frac{CR}{TR}\right)^{e}\times\prod_{i=1}^{e-1}(n+i\times k)\times\frac{N^2\times TR}{TR-N\times CR}+\sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^{i}\times\prod_{j=1}^{i+1}(n+j\times k)\right\}+\left(\frac{CR}{TR}\right)^{e-1}\times N\times\prod_{j=1}^{e-1}(n+j\times k)\right]$ | $(\gamma(Cyln/g)+\theta)\times\frac{N\times CR\times TR}{TR-N\times CR}$ |

**따름정리1:** 버퍼 스케줄링 방식에 따른 동적 버퍼 할당 기법의 버퍼 크기는 표 2와 같다.

**증명:** 제2.2절에서 언급한 각 버퍼 스케줄링 방식의 디스크 지연 시간을 수식 (6)의 $DL$에 대치하면 표2의 버퍼 크기 결정 수식을 얻는다.

동적 버퍼 할당 기법이 버퍼 크기 결정을 위해 사용하는 따름정리1의 수식 계산은 서버가 버퍼를 할당할 때마다 수행되어야 하기 때문에 많은 CPU 시간이 소요될 수 있다. 이 문제는 모든 가능한 $n$과 $k$의 값에 대해서 수식 계산을 미리 수행하여 계산된 값들을 시스템에 저장하고, 실제 버퍼 할당시에 이미 계산된 값을 사용함으로써 해결할 수 있다. 이 경우에 $n$과 $k$의 최대값은 $N$이므로 메모리 오버헤드는 $O(N^2)$이나, $N$은 작은 값(=79)이기 때문에 메모리 오버헤드는 무시할 수 있다.

### 3.3 추가 요청 수

동적 버퍼 할당 기법에서 서비스 중인 사용자 요청 수는 시스템에서 관리하는 반면에, 추가 요청 수는 예상 추가 요청 수를 사용한다. 그런데, 예상 추가 요청 수를 버퍼 할당 시점마다 결정하게 되면, 예상 추가 요청 수가 바로 앞 버퍼 할당 시점의 예상 추가 요청 수에 비해서 증가하거나 감소할 수 있다. 그런데, 예상 추가 요청 수가 증가하면, 할당되는 버퍼의 크기가 커져 이들 버퍼들을 서비스하는데 시간이 많이 걸리고, 이로 인해 다른 버퍼들을 서비스할 시간이 늦어져 이들 버퍼들이 비게 된다. 따라서, 본 논문에서는 이 문제점을 해결하고, 버퍼 크기 결정을 단순화하며, 버퍼 할당 시점마다 예상 추가 요청 수를 결정하는 오버헤드를 피하기 위해서 시스템의 초기화 단계에서 예상 추가 요청 수를 결정하고, 이를 모든 할당 시점에서 일관되게 사용한다.

본 절에서는 여러 기존 연구에서와 같이 사용자 요청은 포아송 과정에 따라 도착한다고 가정한다[15, 19, 20]. 제3.3.1절에서 예상 추가 요청 수의 결정 방법에 관해서 설명하고, 제3.3.2절에서는 제3.3.1절에서 사용한 포아송 과정의 파라미터인 평균 도착률의 결정 방법에 관해서 설명한다.

3.3.1 예상 추가 요청 수의 결정

본 절에서는 사용자 요청의 평균 도착률을 $\lambda$로 가정하고, 예상 추가 요청 수를 결정하는 방법을 설명한다. 동적 버퍼 할당 기법에서 예상 추가 요청 수가 너무 작으면 추정 실패가 많이 발생하여 초기대기시간이 길어지고, 이 값이 너무 크면 버퍼 크기가 커져 메모리 요구량이 증가한다. 따라서, 본 논문에서는 높은 추정 성공 확률을 보장하면서 가능한 작은 값을 예상 추가 요청 수로 결정한다. 즉, 추정 성공 확률이 $P_k$ 이상이 되는 추가 요청 수의 범위를 구하고, 이의 최소값을 예상 추가 요청 수로 결정한다. 이를 정리하면 보조정리1과 같으며, 여기에서 확률값 $P_k$는 시스템 관리자에 의해 주어진다고 가정한다.

**보조정리1:** 동적 버퍼 할당 기법의 예상 추가 요청 수는 수식(7)의 확률 함수 $f(\lambda, k, T)$가 $P_k$이상이 되는 $k$중에서 최소값이다. 여기에서 $T$는 서비스 주기의 최대치 ($n=N$인 경우) 이다.

$$f(\lambda, k, T) = \sum_{i=0}^{k} P(\lambda, T, i) \tag{7}$$

$$P(\lambda, T, i) = \frac{e^{\lambda T}(\lambda T)^i}{(i!)} \tag{8}$$

**증명:** 평균 도착률 $\lambda$하에서 $T$시간 동안에 도착하는 사용자 요청 수가 $k$보다 작거나 같을 확률 함수가 $f(\lambda, k, T)$라 하자. 그러면, 추정 성공 확률이 $P_k$ 이상이 되는 예상 추가 요청 수의 범위는 $f(\lambda, k, T) \geq P_k$을 만족하는 모든 $k$가 된다. 여기에서 $f(\lambda, k, T)$는 $T$시간 동안에 도착하는 사용자 요청 수 $i$가 $0 \leq i \leq k$를 만족할 확률 함수이므로 수식(7)과 같이 나타낼 수 있고, 수식에서 $P(\lambda, T, i)$는 $T$시간동안에 $i$개의 사용자 요청이 도착할 확률 함수이다. $P(\lambda, T, i)$는 대기행렬이론에서 수식(8)과 같이 제시되어 있다[21]. 결론적으로, 동적 버퍼 할당 기법의 예상 추가 요청 수는 수식(7)의 확률 함수 $f(\lambda, k, T)$가 $P_k$ 이상이 되는 $k$중에서 최소값이 된다. □

3.3.2 평균 도착률의 결정

제 3.3.1 절에서 논한 예상 추가 요청 수는 최대 서비스 주기 $T$와 평균 도착률 $\lambda$를 파라미터로 하여 결정한다. 그런데, 최대 서비스 주기 $T$는 일정하므로 예상 추가 요청 수는 평균 도착률 $\lambda$에 의해 결정된다. 이러한 $\lambda$는 시스템의 과거 사용 기록을 이용하는 방법(방법1)과 시스템에서 지원할 수 있는 최대 동시 사용자 요청 수 $N$을 이용하는 방법(방법2)으로 결정할 수 있다. 방법1은 동적으로 변하는 사용자 요청의 도착 빈도에 기반하여 평균 도착률을 결정하기 때문에 $\lambda$는 동적으로 변하게 되고, 이로 인해 일관되게 사용할 예상 추가 요청 수를 결정하기 어렵다. 반면에, 방법2는 항상 일정한 시스템의 최대 동시 사용자 요청 수를 고려하여 $\lambda$를 결정하므로 이러한 문제점을 피할 수 있다. 따라서, 본 논문에서는 방법2을 사용하여 평균 도착률 $\lambda$를 결정하고, 이를 예상 추가 요청 수의 결정에 사용한다.

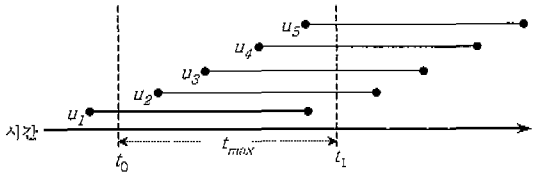
방법2에서는 $\lambda$를 VOD 시스템이 $N$개 이상의 사용자 요청들은 동시에 서비스할 수 없음을 이용하여 결정한다. 평균 도착률 $\mu$ 하에서 서비스해야 할 동시 사용자 요청 수가 $N_\mu$인 경우에, $N_\mu \leq N$일 확률 $P(N_\mu \leq N)$가

$P_\lambda$ 이상이 되는 $\mu$의 범위를 구하고, 이의 최대값을 본 논문의 평균 도착률 $\lambda$로 결정한다. 이를 정리하면 보조정리2와 같으며, 여기에서 확률값 $P_\lambda$는 시스템 관리자에 의해 주어진다고 가정한다.

**보조정리2:** 동적 버퍼 할당 기법에서의 평균 도착률 $\lambda$는 수식(7)의 확률 함수 $f(\lambda, N, t_{max})$가 $P_\lambda$ 이상이 되는 평균 도착률 $\mu$ 중에서 최대값이다. 여기에서 $t_{max}$는 디스크에 저장된 비디오 데이타의 상영 시간들 중에 최대값이다.

**증명:** 확률 함수 $P(N_\mu \le N)$은 $N_\mu$가 $0 \le N_\mu \le N$일 확률 함수이다. 이 확률 함수를 유도하기 위해서는 $N_\mu$을 구하는 것이 필요하다. $N_\mu$는 평균 도착률 뿐만 아니라, 각 사용자 요청이 비디오를 관람하는 시간에 영향을 받기 때문에, 본 논문에서는 최악의 경우를 고려하여 각 사용자 요청의 비디오 관람 시간을 최대로 하여 $N_\mu$를 구한다. 즉, 각 사용자 요청의 비디오 관람 시간을 $t_{max}$로 한다. 이와 같이 모든 사용자 요청이 $t_{max}$시간동안 비디오를 관람하면 임의의 시간 $t_1$에서의 $N_\mu$는 $t_0(=t_1-t_{max})$시간부터 $t_1$시간 사이에 도착한 사용자 요청 수와 같다.

그림 3은 비디오 관람 시간이 $t_{max}$인 사용자 요청들의 비디오 관람 기간을 나타내고 있다. 이 그림에서 $t_1$시점에서의 $N_\mu$는 $t_0$와 $t_1$사이에 도착한 사용자 요청 수이고, 이들 사용자 요청은 $\mu_2$, $\mu_3$, $\mu_4$, 그리고 $\mu_5$이다. 따라서, $N_\mu$는 $t_{max}$시간 동안에 도착하는 사용자 요청 수가 되며, $P(N_\mu \le N)$은 $t_{max}$시간 동안에 도착하는 사용자 요청 수 $i$가 $0 \le i \le N_\mu$일 확률 함수가 되어 수식(7)을 사용하여 $f(\mu, N, t_{max})$로 표현된다. 결론적으로, 평균 도착률 $\lambda$는 $f(\mu, N, t_{max}) \le P_\lambda$을 만족하는 $\mu$ 중에서 최대값이 된다. □




$ui$: 사용자 요청 $i$의 비디오 관람 기간

그림 3 비디오 관람 시간이 $t_{max}$인 경우에 각 사용자 요청의 비디오 관람 기간

## 4. 동적 버퍼 할당 기법의 메모리 요구량 분석

본 장에서는 각 버퍼 스케쥴링 방식별로 동적 버퍼 할당 기법의 최소 메모리 요구량을 분석한다. 메모리 요구량은 버퍼 할당 기법의 성능 평가 기준 중에 하나인 지원 가능한 동시 사용자 요청 수를 결정하는 중요한 요소이다.

라운드 로빈 방식은 BubbleUp을 적용하여 각 버퍼들을 일정한 시간 간격을 두고, 주기적으로 서비스한다. 그리고, 모든 사용자 요청은 동일한 소비 속도로 비디오 데이타를 소비한다. 따라서, 각 버퍼의 메모리 요구 함수는 주기 함수(periodic function)가 되고[4], 라운드 로빈 방식의 최소 메모리 요구량은 이 주기 함수가 최대치를 가지는 시점에서의 메모리 요구량으로서 정리2에 의해서 구할 수 있다.

**정리2:** 동적 버퍼 할당 기법을 적용한 라운드 로빈 방식하에서 서비스 중인 사용자 요청 수가 $n$이고, 추가 요청 수가 $k$인 경우의 최소 메모리 요구량 $Mem_{min}^{RR}(k, n)$은 다음과 같다.

$$Mem_{min}^{RR}(k,n) = n \times BS_k^{RR}(n) - BS_k^{RR}(n) \times \frac{n \times (n-1)}{2 \times (k+n)} + n \times CR \times DL^{RR}$$

**증명:** 부록B 참조. □

스윕* 방식에서 메모리가 가장 많이 요구되는 시점은 $n$개의 버퍼중에서 $n$-1번째 버퍼를 서비스하기 위해서 버퍼를 할당하는 시점이다[4]. 따라서, 이 시점에서 요구되는 메모리 요구량이 스윕* 방식을 지원하기 위한 최소 메모리 요구량으로서 정리3에 의해서 구할 수 있다.

**정리3:** 동적 버퍼 할당 기법을 적용한 스윕* 방식하에서 서비스 중인 사용자 요청 수가 $n$이고, 추가 요청 수가 $k$인 경우의 최소 메모리 요구량 $Mem_{min}^{Sweep}(k, n)$은 다음과 같다.

$$Mem_{min}^{Sweep}(k,n) = \begin{cases} (n-1) \times BS_k^{Sweep}(n) + \left( \frac{n \times T}{k+n} - \frac{(n-2) \times BS_k^{Sweep}(n)}{TR} \right) \times CR \times n & , n > 1 \\ BS_k^{Sweep}(1) + \left( \frac{BS_k^{Sweep}(1)}{TR} + DL^{Sweep} \right) \times CR & , n = 1 \end{cases}$$

**증명:** 부록C 참조. □

GSS* 방식에서 각 그룹들은 BubbleUp에 의해서 서비스되기 때문에, 일정한 시간 간격을 유지하면서 주기적으로 서비스 된다[4]. 그러므로, 각 그룹의 메모리 요구 함수는 주기 함수가 되고[4], GSS* 방식의 최소 메모리 요구량은 이 주기 함수가 최대치를 가지는 시점에서의 메모리 요구량이 된다. 정리4는 $G$가 1보다 큰 경우의 최소 메모리 요구량이고, $G$가 1인 경우는 스윕* 방식으로 서비스되므로 정리3과 동일하다.

**정리4:** 동적 버퍼 할당 기법을 적용한 GSS* 방식하에서 서비스 중인 사용자 요청 수가 $n$, 추가 요청 수가 $k$, 그리고 그룹내의 버퍼 수가 $g$인 경우의 최소 메모리

요구량 $Mem_{min}^{GSS}(g, k, n)$은 다음과 같다. 수식에서 $G$는 그룹의 수로서 $n/g$이고, $g'$은 $n - n/g$ $g$이다.

$$Mem_{min}^{GSS}(g,k,n)=$$

$$\begin{cases}(G-1)\times\left\{g\times BS_g^{GSS}(n)-\left(\frac{n\times T}{k+n}+\frac{(g-2)\times BS_k^{GSS}(n)}{TR}-\frac{g\times T\times(G+2)}{2\times(k+n)}\right)\times CR\times g\right\}+ \\ (g-1)\times BS_k^{GSS}(n)+\left(\frac{T\times g}{k+n}-\frac{(g-2)\times BS_k^{GSS}(n)}{TR}\right)\times CR\times g \\ \quad ,G=\frac{n}{g} \\ (G-2)\times\left\{g\times BS_1^{GSS}(n)-\left(\frac{n\times T}{k+n}+\frac{(g-2)\times BS_k^{GSS}(n)}{TR}-\frac{g\times T\times(G+1)}{2\times(k+n)}\right)\times CR\times g\right\}+ \\ BS_k^{GSS}(n)\times(g+g'-1)+CR\times\left\{\left(\frac{T\times g}{k+n}-\frac{(g-2)\times BS_k^{GSS}(n)}{TR}\right)\times g-\frac{(g-2)\times g\times BS_1^{GSS}(n)}{TR}\right\} \\ \quad ,G>\frac{n}{g},1\le g'<g\end{cases}$$

단, 여기서 $G>1$이다

**증명:** 부록D 참조. □

## 5. 성능평가

본 장에서는 동적 버퍼 할당 기법과 정적 버퍼 할당 기법의 성능을 평가한다. 본 장에서는 분석과 시뮬레이션을 통해서 각 버퍼 할당 기법의 초기대기시간과 지원 가능한 동시 사용자 요청 수를 평가한다. 제 5.1 절에서는 성능 평가를 수행하는 환경에 관해서 설명한다. 그리고 제 5.2 절에서는 초기대기시간을 평가하고, 제 5.3 절에서는 시스템이 지원하는 동시 사용자 요청 수를 평가한다.

### 5.1 성능 평가 환경

성능 평가는 표3과 같은 사양의 Seagate Barracuda 9LP 디스크[4, 16]를 사용하는 VOD 시스템에서 수행한다. 그리고, 디스크에 저장된 비디오 데이타는 평균 전송률이 1.5Mbps인 MPEG-1으로 인코딩된 120분짜리 비디오라고 가정한다. 디스크 헤드가 $x$개의 실린더 거리를 탐색하는 탐색 시간 함수 $\gamma(x)$는 참고문헌[4, 7]에서 제안한 모델을 따라 수식 (9)와 같이 설정한다. 수식에서 $\alpha 1$, $\alpha 2$, $\beta 1$, 그리고 $\beta 2$의 값은 표 3과 같다.

표 3 Seagate Barracuda 9LP 디스크의 사양

| Parameter Name | Value |
|---|---|
| Disk Capacity | 9.19 Gbytes |
| Min. Transfer Rate (TR) | 120 Mbps |
| RPM | 7,200 |
| Max. Rotational Latency Time | 8.33 ms |
| Max. Seek Time(read) | 13.4 ms |
| 1 | 0.54 ms |
| 1 | 0.26 ms |
| 2 | 5 ms |
| 2 | 0.0014 ms |
| $N$ | 79 |

시뮬레이션에서 사용자 요청의 평균 도착률 $\lambda$는 30분 단위로 변화하며, 이 변화는 서비스를 시작한 후에 9시간이 지난 시점을 정점으로 하는 Zipf 분포를 따른다고 가정한다[18]. Zipf 분포는 파라미터로 0과 1사이의 값인 $\theta$를 가지며, $\theta$가 1이면 균일 분포가 되고, 0이면 편중(bias) 분포가 된다[18]. 시뮬레이션은 $\theta$가 0.0, 0.5, 1.0인 경우에 대해서 수행된다. 그리고, 사용자 요청들의 비디오 관람 패턴을 시뮬레이션하기 위해서 사용자 요청들의 비디오 관람 시간은 0과 120분사이의 균일분포를 따른다고 가정한다[15]. 그림4는 Zipf 분포의 $\theta$ 값에 따라 시스템의 동시 사용자 요청 수를 나타내고 있다. 그림 4에서는 $\theta$가 0.0이거나 0.5이면, 7시와 13시사이의 평균 도착률이 높기 때문에 사용자 요청이 이 시간대에 편중되어 도착하고, $\theta$가 1.0이면 사용자 요청이 균일하게 도착함을 보이고 있다. VOD 시스템에서는 서비스 중인 사용자 요청 수가 $N$과 같게 되면 새로 도착한 사용자 요청은 시스템의 승인 제어(admission control)에 의해서 거절된다. 따라서, $\theta$가 0.0이거나 0.5인 경우에 7시와 13시 사이에 도착하는 많은 사용자 요청이 거절된다.

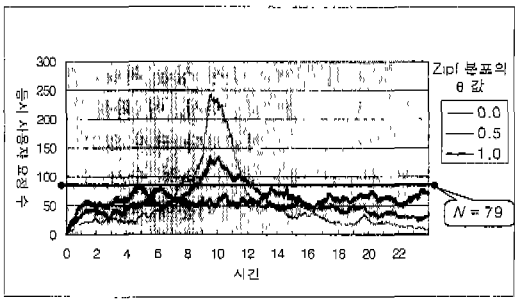
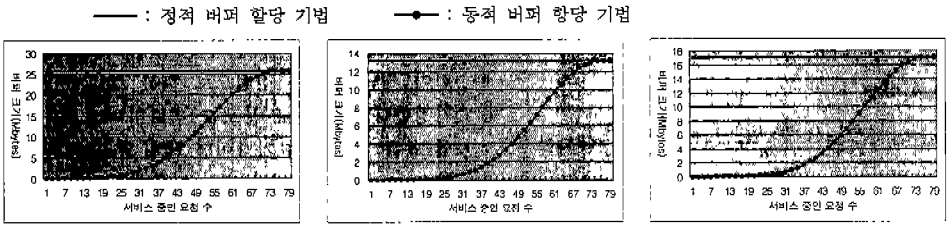



그림 4 사용자 요청의 평균 도착률 $\lambda$의 분포(Zipf 분포의 $\theta$ 값)별로 시스템이 서비스해야 할 동시 사용자 요청 수

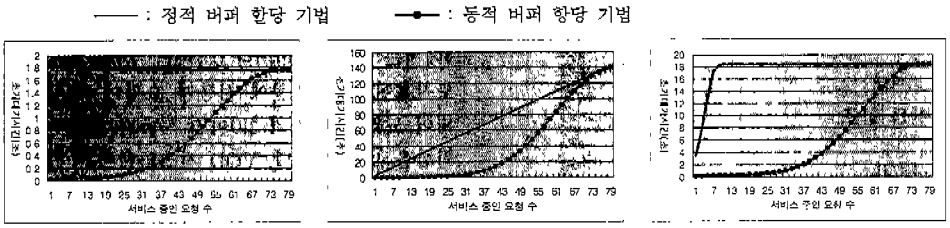
성능 평가는 대표적 버퍼 스케줄링 방식인 라운드 로빈 방식, 스윕*, 그리고 GSS* 방식에 대해서 수행한다. 그리고, GSS* 방식에서 한 그룹내의 버퍼 수는 메모리 요구량이 가장 적은 경우의 버퍼 수로 결정하므로[5], 본 논문에서도 정적 버퍼 할당 기법과 동적 버퍼 할당 기법의 메모리 요구량이 가장 적은 경우의 버퍼 수인 8을 한 그룹내의 버퍼 수로 사용한다.

동적 버퍼 할당 기법의 예상 추가 요청 수를 결정하기 위해서는 확률 $P_k$와 $P_\lambda$를 결정해야 한다. 본 논문의 성능 평가에서는 $P_k$와 $P_\lambda$가 0.9999인 경우, 즉, 예상 추정




(a) 라운드 로빈 방식 (b) 스윕* 방식 (c) GSS* 방식

그림 5 정적과 동적 버퍼 할당 기법의 버퍼 크기




(a) 라운드 로빈 방식 (b) 스윕* 방식 (c) GSS* 방식

그림 6 분석을 통한 정적과 동적 버퍼 할당 기법의 최악의 초기대기시간

성공 확률이 99.98%(=99.99%×99.99%)인 경우를 사용한다. 이는 추정 실패가 1주일에 1번 정도 발생하는 예상 추정 성공 확률이다. 예상 추정 성공 확률은 예상 추가 요청 수가 추정 성공할 확률로서, 이 확률은 결정된 평균 도착률이 실제 평균 도착률보다 크지 않고 이 평균 도착률 하에서 결정된 예상 추가 요청 수가 실제 추가 요청 수보다 크지 않을 확률이기 때문에, 각 경우에 대한 확률 값 $P_\lambda$와 $P_k$의 곱으로 표현된다. 표 4는 예상 추정 성공 확률이 99.98%인 경우에 각 버퍼 스케줄링 방식과 $\theta$의 값별로 실제 추정 성공률을 나타낸다. 실제 추정 성공률은 시뮬레이션을 통해 구한 추정 성공률이다. 표 4에서 대부분의 경우에는 예상 추정 성공 확률이 실제 추정 성공률에 유사함을 보이고 있으나, $\theta$가 1.0인 경우의 라운드 로빈 방식에서는 약간의 차이를 보인다. 이는 라운드 로빈 방식의 서비스 주기가 다른 스케줄링 방식에 비해 긴 디스크 지연 시간으로 인해 길고, 그림4에서 보는 바와 같이 $\theta$가 1.0인 경우에는 다른 $\theta$ 값에 비해서 시간에 관계없이 일관되게 많은 사용자 요청이 도착하여 추정 실패가 발생할 확률이 높기 때문이다.

성능 평가에서 각 버퍼 할당 기법이 버퍼 스케줄링 방식에 따라 각 사용자 요청에게 할당할 버퍼의 크기는 그림 5와 같다. 정적 버퍼 할당 기법은 수식(5)를 사용하고, 동적 버퍼 할당 기법은 수식(6)을 사용하여 버퍼 크기를 결정한다. 그림에서 정적 버퍼 할당 기법은 시스템의 완

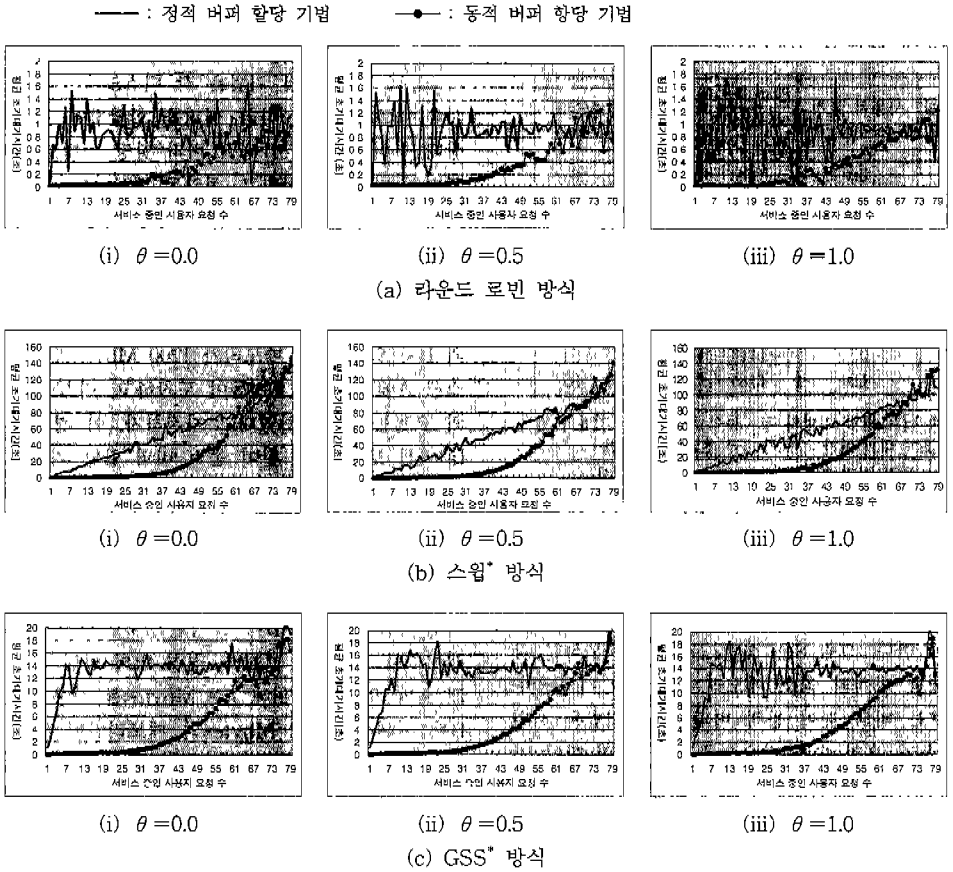
표 4 버퍼 스케줄링 방식별로 예상 추가 요청 수와 그리고 실제 추정 성공률($P_k$, $P_\lambda$=0.9999, 예상 추정 성공 확률 = 99.98%)

| 버퍼 스케줄링 방식 | 예상 추가 요청 수/ 실제 추정 성공률 | 사용자 요청의 평균 도착률 분포($\theta$) | | |
|---|---|---|---|---|
| | | 0.0 | 0.5 | 1.0 |
| 라운드 로빈 | 예상 추가 요청 수 | 6 | | |
| | 실제 추정 성공률(%) | 99.78 | 99.91 | 99.31 |
| 스윕* | 예상 추가 요청 수 | 5 | | |
| | 실제 추정 성공률(%) | 99.97 | 1.0 | 99.92 |
| GSS* | 예상 추가 요청 수 | 5 | | |
| | 실제 추정 성공률(%) | 99.94 | 99.97 | 99.65 |

전 부하 상태만을 고려하여 버퍼 크기를 결정하기 때문에 버퍼 크기가 일정하게 나타난다. 반면에, 동적 버퍼 할당 기법은 서비스 중인 사용자 요청 수와 추가 요청 수를 고려하여 버퍼 크기를 결정하기 때문에 버퍼 크기가 서비스 중인 사용자 수에 따라 동적으로 변함을 알 수 있다.

### 5.2 초기대기시간

본 절에서는 먼저 분석을 통하여 각 버퍼 할당 기법의 최악의 초기대기시간을 평가한다. 그리고, 시뮬레이션을 통해 각 버퍼 할당 기법의 평균 초기대기시간을 평가한다. 그림 6은 각 버퍼 스케줄링 방식별로 최악의 초기대기시간을 나타내는 수식(2), (3), (4)에 각 버퍼 할당 기법의 버퍼 크기를 대입한 것으로서, 각 버퍼 스




(i) $\theta=0.0$ (ii) $\theta=0.5$ (iii) $\theta=1.0$

(a) 라운드 로빈 방식

(i) $\theta=0.0$ (ii) $\theta=0.5$ (iii) $\theta=1.0$

(b) 스윕* 방식

(i) $\theta=0.0$ (ii) $\theta=0.5$ (iii) $\theta=1.0$

(c) GSS* 방식

그림 7 시뮬레이션을 통한 정적과 동적 버퍼 할당 기법의 평균 초기대기시간

케줄링 방식별로 각 버퍼 할당 기법의 최악의 초기대기시간을 나타낸다. 그림에서와 같이 동적 버퍼 할당 기법의 초기대기시간은 서비스 중인 사용자 요청 수가 작을수록 정적 버퍼 할당 기법보다 작음을 알 수 있다. 이는 동적 버퍼 할당 기법이 서비스 중인 사용자 요청 수가 작을수록 정적 버퍼 할당 기법보다 더 작은 버퍼를 할당하기 때문이다.

그림 7은 24시간 동안의 시뮬레이션을 수행한 후에 각 사용자 요청의 초기대기시간을 서비스 중인 사용자 요청 수별로 평균한 것이다. 그림 7에서 진동하는 것을 제외한 그래프의 경향(trend)은 그림 6의 분석 결과와 유사함을 알 수 있다. 그림에서 보는 바와 같이 동적 버퍼 할당 기법은 버퍼 스케줄링 방식이나 서비스 중인 사용자 요청 수에 관계없이 거의 대부분의 경우에 정적 버퍼 할당 기법보다 더 작은 초기대기시간이 소요되었다. 그림 7이 그림 6에 비해 초기대기시간이 더 작은 이유는 그림 7은 평균 초기대기시간을 나타내고, 그림6은 최악의 초기대기시간을 나타내기 때문이다. 그리고, 그림7이 그림 6과 달리 진동하는 것은 초기대기시간이 사용자 요청의 도착 시점에 영향을 받기 때문이다. 즉, 그림6의 분석에서는 사용자 요청의 도착 시점을 최악의 경우로 한 반면에, 그림 7의 시뮬레이션에서는 동적으로 변하기 때문이다.

표 5는 각 버퍼 스케줄링 방식과 사용자 요청의 평균 도착률 분포별로 그림 7의 시뮬레이션에서 각 버퍼 할당 기법이 서비스한 사용자 요청들의 초기대기시간을 서비스 중인 사용자 요청 수에 관계 없이 전체적으로 평균한 값을 나타낸다. 표에서 보는 바와 같이 동적 버퍼 할당 기법은 정적 버퍼 할당 기법에 비해서 평균 초기대기시간을 라운드 로빈 방식에서는 31% ~ 65%, 스윕* 방식에서는 29% ~ 36%, 그리고 GSS* 방식에서는 36% ~ 63%를 감소시킴을 볼 수 있다.

표 5 정적과 동적 버퍼 할당 기법이 서비스한 사용자 요청들의 평균 초기대기시간

| 버퍼 스케줄링 방식 | 사용자 요청 도착 분포($\theta$) | 평균 초기대기시간(초) | | 향상도* (%) |
|---|---|---|---|---|
| | | 정적 버퍼 할당 기법 | 동적 버퍼 할당 기법 | |
| 라운드 로빈 | 0.0 | 0.91 | 0.32 | 65 |
| | 0.5 | 0.94 | 0.43 | 54 |
| | 1.0 | 0.91 | 0.62 | 31 |
| 스윕* | 0.0 | 73.14 | 51.83 | 29 |
| | 0.5 | 79.70 | 51.00 | 36 |
| | 1.0 | 77.51 | 51.64 | 33 |
| GSS* | 0.0 | 13.92 | 5.09 | 63 |
| | 0.5 | 14.13 | 6.50 | 54 |
| | 1.0 | 13.85 | 8.84 | 36 |

*향상도= (정적 버퍼 할당 기법의 평균 초기 대기 시간 − 동적 버퍼 할당 기법의 평균 초기 대기 시간) / 정적 버퍼 할당 기법의 평균 초기 대기 시간

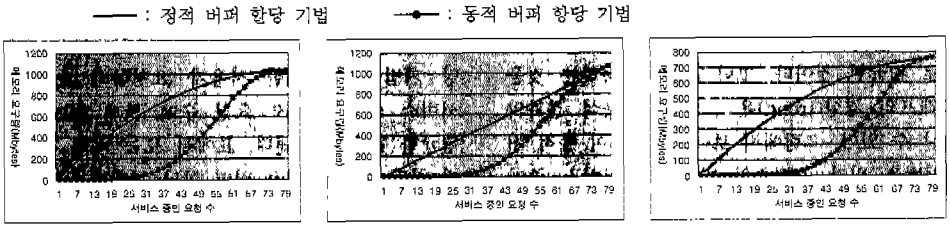
### 5.3 동시 사용자 요청 수

VOD 시스템에서 동일한 메모리량으로 더 많은 사용자 요청들을 동시에 서비스하기 위해서는 메모리 요구량을 줄여야 한다. 그림 8은 정리2, 3, 4에서 분석한 각 버퍼 스케줄링 방식의 최소 메모리 요구량을 나타낸다. 그림에서 보는 바와 같이 동적 버퍼 할당 기법은 서비스 중인 사용자 요청 수가 작을수록 정적 버퍼 할당 기법에 비해 더 적은 양의 메모리를 요구한다. 이와 같은 특성은 하나의 디스크를 사용하는 VOD 시스템에서는 동시 사용자 요청 수를 증가시키지 못하나, 여러 개의 디스크를 사용하는 VOD 시스템에서는 동시 사용자 요청 수를 증가시킬 수 있다.

여러 개의 디스크를 사용하는 VOD 시스템에서는 비디오 인기도에 따라 특정 디스크에 부하가 편중되는 현상이 발생한다[18]. 이로 인해, 특정 디스크는 완전 부하 상태가 되는 반면에, 대다수의 디스크들은 불완전 부하 상태가 된다. 그런데, 정적 버퍼 할당 기법은 완전 부하 상태만을 고려하여 버퍼를 할당하기 때문에 불완전 부하 상태인 디스크들에서는 필요이상으로 큰 버퍼를 할당하여 전체적으로 메모리를 낭비하게 된다. 그러나, 동적 버퍼 할당 기법은 각 디스크에서 서비스 중인 사용자 요청 수에 따라 버퍼를 할당하여 불완전 부하 상태인 디스크들에서 정적 버퍼 할당 기법보다 더 작은 크기의 버퍼를 할당하고, 절약된 메모리를 완전 부하 상태인 디스크에서 활용하여 더 많은 사용자 요청들을 서비스한다.

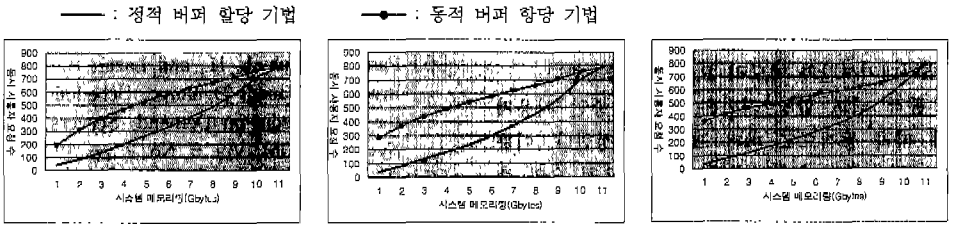
동시 사용자 요청 수의 증가를 구체적인 예를 통해 다시 설명하면 다음과 같다. VOD 시스템에 디스크 A와 B가 있고 가용 메모리가 1200 Mbytes인 경우에, 디스크 A에 41개의 요청들이 도착하고, 디스크 B에 79개의 요청들이 도착하였다고 가정하자. 이 경우에 그림 8 (a)에서 보는 바와 같이 동적 버퍼 할당 기법은 디스크 A에 도착한 모든 요청들을 서비스하기 위해서 170 Mbytes가 필요한 반면, 정적 버퍼 할당 기법은 788 Mbytes가 필요하다. 따라서, 동적 버퍼 할당 기법은 디스크 A에서 요구되는 메모리를 할당한 후 남은 1030 Mbytes로 디스크 B에 도착한 79개의 요청들을 모두 서비스할 수 있다. 그러나, 정적 버퍼 할당 기법은 디스크 A의 요청들을 위해 788 Mbytes를 사용하여 디스크 B를 위해서는 412 Mbytes의 메모리만이 남기 때문에 오직 18개의 요청만을 서비스할 수 있다. 결과적으로, 동적 버퍼 할당 기법은 120개의 요청을 서비스할 수 있는 반면에 정적 버퍼 할당 기법은 오직 59개의 요청들만을 서비스할 수 있다.

그림 9는 10개의 Seagate Barracuda 9LP 디스크로 구성된 VOD 시스템에서 서비스할 수 있는 동시 사용자 요청 수의 분석 결과를 나타낸다. 분석은 라운드 로빈 방식에 대해서 수행하였으며, 다른 버퍼 스케줄링 방식에서도 유사한 결과를 얻었다. 그림은 사용자 요청들이 Zipf 분포에 따라 각 디스크에 도착한다고 가정하고, 가 0.0, 0.5, 그리고 1.0인 경우에 대해서 정리2, 3, 4를 사용하여 동시 사용자 요청 수(즉, 각 수식의 $n$)를 계산한 것이다.



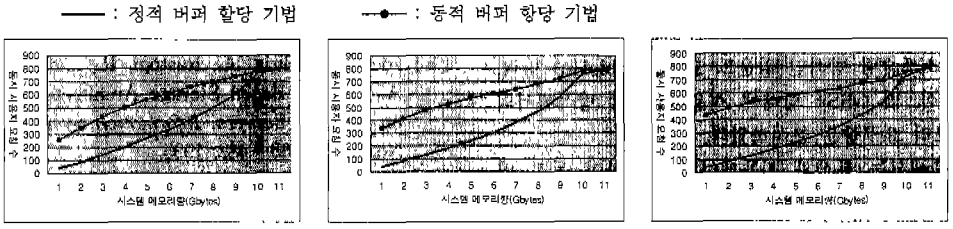

(a) 라운드 로빈 방식 (b) 스윕* 방식 (c) GSS* 방식

그림 8 분석을 통한 정적과 동적 버퍼 할당 기법의 최소 메모리 요구량




(a) $\theta$ =0.0 (b) $\theta$ =0.5 (c) $\theta$ =1.0

그림 9 Zipf 분포를 따르는 디스크 부하 분포하에서 라운드 로빈 방식으로 서비스 가능한 동시 사용자 요청 수




(a) $\theta$ =0.0 (b) $\theta$ =0.5 (c) $\theta$ =1.0

그림 10 시뮬레이션에서 라운드 로빈 방식으로 서비스한 동시 사용자 요청 수

참고로, 비디오의 인기도는 가 0.271인 Zipf 분포를 따른다고 알려져 있다[18]. 그림에서 보는 바와 같이 동적 버퍼 할당 기법은 디스크 부하 분포에 관계없이 정적 버퍼 할당 기법에 비해서 더 많은 수의 사용자 요청을 동시에 서비스함을 볼 수 있다. 이는 동적 버퍼 할당 기법이 메모리를 효율적으로 사용하기 때문이다. 그리고 시스템 메모리량이 11 Gbyte인 경우에는 각 버퍼 할당 기법이 동일한 동시 사용자 요청 수를 서비스함을 볼 수 있는데, 이는 메모리량이 충분하여 동시 사용자 요청 수가 오직 디스크 성능의 한계에 의해서 결정되기 때문이다.

그림 10은 그림 9와 같이 10개의 Seagate Barracuda 9LP 디스크를 사용하는 VOD 시스템이 라운드 로빈 방식으로 서비스하는 경우에 대해서 시뮬레이션을 수행하고, 각 디스크 부하 분포별로 동시 사용자 요청 수를 측정한 결과이다. 그림 10에서 보는 바와 같이 시뮬레이션의 결과는 그림 9의 분석 결과와 유사하게 동적 버퍼 할당 기법이 정적 버퍼 할당 기법에 비해 동시에 더 많은 사용자 요청을 지원함을 알 수 있다.

표 6은 디스크 부하 분포별로 그림 10의 동시 사용자 요청 수를 시스템 메모리량에 관계 없이 평균한 것이다. 표에서 보는 바와 같이 동적 버퍼 할당 기법은 정적 버퍼 할당 기법에 비해 평균 동시 사용자 요청 수를 48% ~ 68% 증가시켰다.

표 6 시뮬레이션에서 라운드 로빈 방식으로 서비스한 평균 동시 사용자 요청 수

| 디스크의 부하 분포($\theta$) | 서비스한 평균 동시 사용자 요청 수 | | 향상도* (%) |
|---|---|---|---|
| | 정적 버퍼 할당 기법 | 동적 버퍼 할당 기법 | |
| 0.0 | 377.4 | 559.7 | 48 |
| 0.5 | 358.4 | 573.6 | 60 |
| 1.0 | 343.3 | 577.8 | 68 |

$$*\text{향상도} = \frac{\text{동적 버퍼 할당 기법의 평균 동시 사용자 요청 수} - \text{정적 버퍼 할당 기법의 평균 동시 사용자 요청 수}}{\text{정적 버퍼 할당 기법의 평균 동시 사용자 요청 수}}$$

## 6. 결 론

본 논문에서는 VOD 시스템의 메모리 요구량과 초기 대기시간을 줄이는 동적 버퍼 할당 기법을 제안하였다. 기존의 정적 버퍼 할당 기법은 시스템이 완전 부하된 상태만을 고려하여 버퍼 크기를 결정하고 할당함으로써, 시스템이 불완전 부하 상태인 경우에는 필요 이상으로 큰 버퍼를 사용자 요청에게 할당한다. 반면, 동적 버퍼 할당 기법은 시스템의 완전 부하 상태뿐만 아니라, 불완전 부하 상태까지를 고려하여 사용자 요청에게 최소 크기의 버퍼를 할당한다. 이를 통해, 동적 버퍼 할당 기법은 평균 초기대기시간을 줄여 사용자에게 더 빠른 응답 시간의 서비스를 제공하고, 메모리 요구량을 줄여 더 많

은 사용자 요청들을 동시에 서비스한다. VOD 시스템에서는 사용자 요청에게 할당되는 버퍼의 크기가 감소함에 따라 초기대기시간과 메모리 요구량이 감소한다.

제안한 동적 버퍼 할당 기법은 서비스 중인 사용자 요청 수에 따라 동적으로 버퍼 크기를 결정한다. 이를 위해, 본 논문에서는 서비스 중인 사용자 요청 수와 추가 요청 수에 기반하여 동적 버퍼 할당 기법에서 사용할 버퍼 크기 결정 수식을 유도하였고, 수식에서 사용하는 추가 요청 수를 결정하는 방법을 제안하였다. 또한, 동적 버퍼 할당 기법이 버퍼 스케줄링 방식에 독립적이기 때문에 기존의 버퍼 스케줄링 방식에 모두 적용할 수 있음을 보이기 위해서 동적 버퍼 할당 기법을 대표적 버퍼 스케줄링 방식인 라운드 로빈(BubbleUp) 방식, 스윕* 방식, 및 GSS* 방식에 적용하고, 각 방식에서의 최소 메모리 요구량을 분석하였다.

본 논문에서는 분석과 시뮬레이션을 통하여 동적 버퍼 할당 기법이 초기대기시간과 지원 가능한 동시 사용자 요청 수에 있어서 정적 버퍼 할당 기법에 비해 크게 우수함을 확인하였다. 시뮬레이션 결과는 동적 버퍼 할당 기법이 정적 버퍼 할당 기법에 비해 평균 초기대기시간을 29% ~ 65% 줄이고, 다수의 디스크들로 구성된 시스템에서 서비스한 평균 동시 사용자 요청 수를 48% ~ 68% 증가시킨 것으로 나타났다. 이와 같은 결과는 동적 버퍼 할당 기법이 VOD 시스템의 성능과 용량을 크게 향상시킴을 보여주는 것이다.

## 참 고 문 헌


[1] Chang, E. and Garcia-Molina, H., Effective Memory Use in a Media Server, In *Proc. 23rd Int'l Conf. on Very Large Data Bases*, pp. 496-505, Athens, Greece, 1997.

[2] Chang, E. and Garcia-Molina, H., BubbleUp: Low Latency Fast-Scan for Media Servers, In *Proc. 5th ACM Int'l Conf. on Multimedia*, pp. 87-98, Seattle, WA, 1997.

[3] Chang, E. and Garcia-Molina, H., Cost-Based Media Server Design, In *Proc. 8th Int'l Workshop on Research Issues in Data Engineering*, pp. 76-83, Orlando, FL, 1998.

[4] Chang, E. and Garcia-Molina, H., Accounting for Memory Use, Cost, Throughput, and Latency in the Design of a Media Server, Technical Report SIDL-WP-1998-0096, Stanford University, 1998 (Available at http://www-db.stanford.edu/pub/papers/jvld98.ps).

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

[6] Pan, H., Ngoh, L. H., and Lazar, A. A., A Buffer-Inventory-Based Dynamic Scheduling Algorithm for Multimedia-on-Demand Servers, *Multimedia Systems*, ACM, Vol. 6, No. 2, pp. 125-136, 1998.

[7] Ruemmler, C. and Wikes, J., An Introduction to Disk Drive Modeling, *IEEE Computer*, Vol. 27, No. 3, pp. 17-28, 1994.

[8] zden, B., Biliris, A., Rastogi, R., and Silberschatz, A., A Low-Cost Storage Server for Movie on Demand Databases, In *Proc. 20th Int'l Conf. on Very Large Data Bases*, pp. 594-605, Santiago, Chile, 1994.

[9] zden, B., Biliris, A., Rastogi, R., and Silberschatz, A., A Disk-Based Storage Architecture for Movie on Demand Servers, *Information Systems*, Vol. 20, No. 6, pp. 465--482, 1995.

[10] Shim, S., Vedavyasa, H., and Du, D. H. C., An Effective Data Placement Scheme to Serve Popular Video On-Demand, In *Proc. Pacific Workshop on Distributed Multimedia Systems*, pp. 153-162, Hong Kong, 1996.

[11] Tobagi, F. A., Pang, J., Baird, R., and Gang, M., Streaming RAID™ A Disk Array Management System for Video Files, In *Proc. 1st ACM Int'l Conf. on Multimedia*, pp. 393-400, Anaheim, CA, 1993.

[12] Reddy, A. L. N. and Wyllie, J. C., I/O Issues in a Multimedia System, *IEEE Computer*, Vol. 2, No. 1, pp. 69-74, 1994.

[13] Berson, S., Ghandeharizadeh, S., Muntz, R., and Ju, X., Staggered Striping in Multimedia Information Systems, In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 79-90, Minneapolis, MN, 1994.

[14] Dey-Sircar, J. K., Salehi, J. D., Kurose, J. F., and Towsley, D., Providing VCR Capabilities in Large-Scale Video Servers, In *Proc. 2nd ACM Int'l Conf. on Multimedia*, pp. 25-32, San Francisco, CA, 1994.

[15] Dan, A., Sitaram, D., and Shahabuddin, P., Scheduling Policies for an On-Demand Video Server with Batching, In *Proc. 2nd ACM Int'l Conf. on Multimedia*, pp. 15-23, San Francisco, CA, 1994.

[16] Seagate, Inc., Seagate Barracuda 9LP Family Product Specification, 1998 (Available at URL: http://www.seagate.com).

[17] Goluchik, L., Lui, J. C. S., and Muntz, R. R., Adaptive Piggybacking: A Novel Techniques for Data Sharing in Video-on-Demand Storage

Servers, *Multimedia Systems*, ACM, Vol. 4, No. 3, pp. 140-155, 1996.
[18] Wolf, J. L., Yu, P. S., and Shachnai, H., Disk Load Balancing for Video-on-Demand Systems, *Multimedia Systems*, ACM, Vol. 5, No. 6, pp. 358-370, 1997.
[19] To, T.-P. J. and Hamidzadeh, B., Dynamic Real-Time Scheduling Strategies for Interactive Continuous Media Servers, *Multimedia Systems*, ACM, Vol. 7, No. 2, pp. 91-106, 1999.
[20] Dan, A. Kienzle, M., and Sitaram, D., A Dynamic Policy of Segment Replication for Load-Balancing in Video-on-Demand Servers, *Multimedia Systems*, ACM, Vol. 3, No. 3, pp. 93-103, 1995.
[21] Kao, E.P.C., *An Introduction to Stochastic Processes*, Duxbury Press, 1997.
[22] Makaroff, D. J. and Ng, R. T., Schemes for Implementing Buffer Sharing in Continuous-Media Systems, *Information Systems*, Vol. 20, No. 6, pp. 445-465, 1995.

## 부록 A. 정리 1의 증명

추가 요청 수가 $k$인 경우에는 서비스 주기 $T$ 내에 $k$개의 사용자 요청이 새로 도착하므로 서비스 중인 사용자 요청 수는 $n$, $n+1$, , $n+k$로 변하고, 이에 따라 버퍼 크기도 $BS_k(n)$, $BS_k(n+1)$, ..., $BS_k(n+k)$로 변한다. 조건 1, 조건 2, 그리고 조건 3은 이 모든 경우의 버퍼 크기에 대해서 만족되어야 한다. 조건 1을 만족하기 위해서는 수식(10)과 같이 가장 작은 버퍼 크기인 $BS_k(n)$이 서비스 주기동안에 사용자 요청이 소비하는 데이타량인 $T\times CR$보다 크거나 같아야 한다. 그리고, 조건 2와 조건 3을 만족시키기 위해서는 수식(11)과 같이 $n+k$개의 가장 큰 버퍼(버퍼 크기가 $BS_k(n+k)$인 버퍼)를 서비스 주기내에 서비스할 수 있어야 한다. 수식(11)에서 $\frac{BS_k(n+k)}{TR}+DL$은 서버가 $BS_k(n+k)$ 크기의 버퍼 하나를 서비스하는데 걸리는 시간이다. 수식(10)과 수식(11)을 전개하면, 수식(12)와 같은 점화 부등식을 얻을 수 있다.

$$BS_k(n)\geq T\times CR \tag{10}$$

$$T\geq(n+k)\times\left(\frac{BS_k(n+k)}{TR}+DR\right) \tag{11}$$

$$BS_k(n)\geq(n+k)\times\left(\frac{BS_k(n+k)}{TR}+DR\right)\times CR \tag{12}$$

VOD 시스템은 최대 $N$개의 사용자 요청을 동시에 서비스할 수 있기 때문에 서비스 주기내에 서비스해야 할 사용자 요청 수는 $N$이하이다. 그러므로, $BS_k(N)$은 완전 부하 상태에서의 최소 버퍼 크기이며, 이는 제 2.3절에서 유도한 수식(5)의 $BS(N)$과 같이 수식 (13)이 된다. 그리고, 불완전 부하 상태에서의 최소 버퍼 크기인 $BS_k(n)$은 점화 부등식인 수식(12)를 전개하여 얻을 수 있다. 수식(12)는 수식(14)로 전개된다. 수식(14)에서 $n+e\times k$는 $N$보다 크거나 같은데, 서비스 주기내에 동시에 서비스할 수 있는 사용자 요청 수는 $N$이하이기 때문에 $n+e\times k$는 $N$으로 대치된다. 이로 인해 수식(14)는 수식(15)가 되고, 수식(15)는 수식(13)으로 $BS_k(N)$이 대치되어 수식(16)이 된다.

$$BS_k(N)=DL\times\frac{N\times CR\times TR}{TR-N\times CR} \tag{13}$$

$$\begin{aligned}
BS_k(n)&\geq(n+k)\times\left(\frac{BS_k(n+k)}{TR}+DL\right)\times CR \qquad ,n<N\\
&=\frac{CR}{TR}\times(n+k)\times BS_k(n+k)+(n+k)\times DL\times CR\\
&\geq\frac{CR}{TR}\times(n+k)\times\left\{\frac{CR}{TR}\times(n+2\times k)\times BS_k(n+2\times k)+(n+2\times k)\times DL\times CR\right\}+(n+k)\times DL\times CR\\
&=\left(\frac{CR}{TR}\right)^2\times(n+k)\times(n+2\times k)\times BS_k(n+2\times k)+(n+k)\times DL\times CR\times\left(\frac{CR}{TR}\times(n+2\times k)+1\right)\\
&\geq\\
&\vdots\\
&\geq\left(\frac{CR}{TR}\right)^e\times\prod_{i=1}^{e}(n+i\times k)\times BS_k(n+e\times k)+DL\times CR\times\sum_{i=0}^{e-1}\left\{\left(\frac{CR}{TR}\right)^i\times\prod_{j=1}^{i}(n+j\times k)\right\} \qquad (14)
\end{aligned}$$

단, 여기서 $e=\left\lceil\frac{N-n}{k}\right\rceil$ 이다.

$$\begin{aligned}
&=\left(\frac{CR}{TR}\right)^e\times\prod_{i=1}^{e}(n+i\times k)\times N\times BS_k(N)+\\
&DL\times CR\times\left[\sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^i\times\prod_{j=1}^{i}(n+j\times k)\right\}+\left(\frac{CR}{TR}\right)^{e-1}\times N\times\prod_{j=1}^{e-1}(n+j\times k)\right] \qquad (15)
\end{aligned}$$

결론적으로, VOD 시스템에서 불완전 부하 상태 및 완전 부하 상태에서의 최소 버퍼 크기 $BS_k(n)$은 수식(16)과 같다.

$$BS_k(n)=\begin{cases}
DL\times CR\times\left[\left(\frac{CR}{TR}\right)^e\times\prod_{i=1}^{e}(n+i\times k)\times\frac{N^2\times TR}{TR-N\times CR}+\right.\\
\left.\sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^i\times\prod_{j=1}^{i}(n+j\times k)\right\}+\left(\frac{CR}{TR}\right)^{e-1}\times N\times\prod_{j=1}^{e-1}(n+j\times k)\right] & ,n<N\\
DL\times\frac{N\times CR\times TR}{TR-N\times CR} & ,n=N
\end{cases} \tag{16}$$

단,여기서 $e=\left\lceil\frac{N-n}{k}\right\rceil$ 이다.

## 부록 B. 정리 2의 증명

라운드 로빈 방식에서의 메모리 요구량은 각 버퍼가 요구하는 메모리량의 합이기 때문에 먼저, 서비스 중인 사용자 요청 수가 $n$이고, 추가 요청 수가 $k$인 경우에 $i$번째 서비스되는 버퍼 $B_i$가 $t$ 시점에서 요구하는 메모리량 $R_i^{RR}(t,k,n)$을 구한다.

라운드 로빈 방식은 BubbleUp을 적용하여 각 버퍼들을 서비스하기 때문에 각 버퍼들은 일정한 시간적인 거

리 $\frac{T}{t+n}$를 두고, 서버로부터 주기적으로 메모리를 할당 받는다. 그리고, 사용자 요청들은 $CR$의 속도로 버퍼의 데이타를 사용한 후에 해당 메모리를 반환한다. 따라서, 버퍼 $B_i$가 마지막으로 서비스 받은 시점이 $_i$인 경우에 $R_i^{RR}(t,k,n)$은 서버로부터 $\tau_i$ 시점에 할당받은 메모리량($BS_k^{RR}(n)$)에서 사용자 요청이 $t-\tau_i$시간 동안 $CR$의 속도로 반환한 메모리량($CR\times(t-\tau_i)$)을 뺀 메모리량이다. 그리고, 서버는 버퍼를 서비스할 때에 항상 디스크 지연이 발생하기 때문에 버퍼에는 디스크 지연 시간동안에 사용자 요청이 사용할 데이타를 유지하기 위해서 $CR\times DL^{RR}$ 크기의 메모리가 추가로 요구되고[4], 이로 인해 $R_i^{RR}(t,k,n)$은 수식(17)이 된다.

$$R_i^{RR}(t,k,n) = BS_k^{RR}(n) - CR\times(t-\tau_i) + CR\times DL^{RR} \quad (17)$$

단, 여기서 $1 \le i \le n$ 인 정수이다.

라운드 로빈 방식의 최소 메모리 요구량 $Mem_{min}^{RR}(k,n)$은 수식(18)과 같이 전체 메모리 요구량의 최대치이다. 전체 메모리 요구량이 최대가 되는 시점은 $t$가 $\frac{T}{k+n}$의 배수일 시점인데[4], 이는 이 시점에서만 서버가 버퍼에 메모리를 할당하고, 할당하는 메모리량도 모든 버퍼에 동일하기 때문이다.[3] 수식(19)은 수식(18)에 $t$의 값으로 $\frac{T}{k+n}$의 배수인 $\frac{(n-1)\times T}{k+n}$를 대입하고, 이에 따라 수식(17)의 $\tau_i$ 값으로 $\frac{(i-1)\times T}{k+n}$을 대입하여 전개한 것으로서 라운드 로빈 방식에서의 최소 메모리 요구량 $Mem_{min}^{RR}(k,n)$을 나타낸다.

$$Mem_{min}^{RR}(k,n) = Max_t \sum_{i=1}^{n} R_i^{RR}(t,k,n) \quad (18)$$

$$= n\times BS_k^{RR}(n) - BS_k^{RR}(n)\times\frac{n\times(n-1)}{2\times(k+n)} + n\times CR\times DL^{RR} \quad (19)$$

## 부록 C. 정리 3의 증명

동적 버퍼 할당 기법을 적용한 스윕* 방식의 최소 메모리 요구량은 참고 문헌[1]의 정리 2로 부터 유도할 수 있다. 참고 문헌[1]의 정리 2는 스윕* 방식으로 $n$개의 사용자 요청을 서비스하는 경우에 요구되는 최소 메모리량이 $(n-1)\times BS(n) + \left(T - \frac{(n-2)\times BS(n)}{TR}\right)\times CR\times n$임을 나타낸다. 여기에서 $T$는 $n$개의 사용자 요청을 서비스하는 시간이고, 동적 버퍼 할당 기법에서의 $T$는 $k+n$개의 사용자 요청을 서비스하는 시간이기 때문에 동적 버퍼 할당 기법하에서 이 정리의 $T$는 $\frac{n\times T}{k+n}$가 된다. 그리고, 버퍼 크기 $BS(n)$도 동적 버퍼 할당 기법에서의 버퍼 크기인 $BS_k^{Sweep}(n)$이 된다. 따라서, 서비스중인 사용자 요청 수 $n$이고, 추가 요청 수가 $k$인 경우에 동적 버퍼 할당 기법을 적용한 스윕* 방식의 최소 메모리 요구량은 아래의 수식이 된다.

$$(n-1)\times BS_k^{Sweep}(n) + \left(\frac{n\times T}{k+n} - \frac{(n-2)\times BS_k^{Sweep}(n)}{TR}\right)\times CR\times n$$

지금까지 유도한 스윕* 방식의 최소 메모리 요구량은 $n$이 1보다 큰 경우이다. $n$이 1인 경우에는 시스템에 하나의 버퍼만 있기 때문에, 이 버퍼를 서비스하는 시점에서의 메모리 요구량이 시스템의 최소 메모리 요구량이 된다. 이 시점에서 서버가 할당한 메모리량은 $BS_k^{Sweep}(1)$이고, 서버가 버퍼를 서비스하는 동안에 사용자 요청이 사용할 데이타를 위한 메모리량은 $\left(\frac{BS_k^{Sweep}(1)}{TR} + DL^{Sweep}\right)\times CR$이다. 따라서, 이 시점에서의 메모리 요구량은 이 두 값을 합한 아래의 수식이 된다.

$$BS_k^{Sweep}(1) + \left(\frac{BS_k^{Sweep}(1)}{TR} + DL^{Sweep}\right)\times CR$$

결론적으로, 동적 버퍼 할당 기법을 적용한 스윕* 방식의 최소 메모리 요구량 $Mem_{min}^{Sweep}(k,n)$은 아래와 같다.

$$Mem_{min}^{Sweep}(k,n) = \begin{cases} (n-1)\times BS_k^{Sweep}(n) + \left(\frac{T\times n}{k+n} - \frac{(n-2)\times BS_k^{Sweep}(n)}{TR}\right)\times CR\times n & , n>1 \\ BS_k^{Sweep}(1) + \left(\frac{BS_k^{Sweep}(1)}{TR} + DL^{Sweep}\right)\times CR & , n=1 \end{cases}$$

## 부록 D. 정리 4의 증명

GSS:: 방식은 각 그룹들을 BubbleUp으로 서비스하기 때문에 각 그룹의 메모리 요구량은 주기 함수가 되고, 각 그룹의 메모리 요구 함수들의 합이 시스템의 메모리 요구량이 된다. 먼저, 각 그룹의 메모리 요구 함수들을 구하도록 한다.

GSS* 방식은 초기대기시간을 줄이기 위해서 현재 서비스 중인 그룹에 가능한 많은 버퍼들을 포함시켜 서비

3) 동적 버퍼 할당 기법에서는 하나의 서비스 주기내에서도 각 버퍼에 할당되는 메모리량이 다를 수 있다. 이 경우는 서비스 주기내에서 새로운 사용자 요청이 도착하거나 서비스 중인 사용자 요청의 서비스가 완료되어 $n$의 값이 서비스 주기내에서 변경되는 경우이다. 이 경우의 전체 메모리 요구량은 $n$의 값이 증가한 후이거나 감소하기 전의 안정상태(steady state)에서의 전체 메모리 요구량보다 적기 때문에, 본 절에서는 n의 값에 변화가 없는 안정상태만을 고려한다.

스함으로써 새로 도착하는 사용자 요청이 다음 서비스 그룹에 바로 포함되어 서비스될 수 있도록 한다[4]. 따라서, 최대 $g$개의 버퍼로 그룹을 구성하는 GSS* 방식에서 첫번째부터 $\lfloor n/g \rfloor$번째에 구성되는 그룹들은 $g$개의 버퍼로 구성되고, 마지막에 구성되는 그룹은 $g'(=n-\lfloor n/g \rfloor \times g)$개의 버퍼로 구성된다. GSS* 방식에서 $g$가 1인 경우는 라운드 로빈 방식과 같아지기 때문에, 여기서는 $g$가 1보다 큰 경우만을 고려한다. GSS* 방식에서 각 그룹내의 버퍼들은 스윕* 방식으로 서비스되기 때문에 각 그룹의 최대 메모리 요구량은 정리3으로 부터 유도할 수 있다. 각 그룹내에서 서비스되는 버퍼의 수는 $g$ 또는 $g'$ 이고, 이들 버퍼를 서비스하는 시간도 $\frac{g\times T}{k+n}$ 또는 $\frac{g'\times T}{k+n}$이기 때문에 이를 정리 3에 적용하면, 추가 요청 수가 $k$이고 서비스 중인 사용자 요청 수가 $n$인 경우에 $i$번째 그룹의 최대 메모리 요구량 $R^{GSS}_{i,max}(k,n)$은 수식(20)과 같다.

$$R^{GSS}_{i,max}(k,n)=$$

$$\begin{cases} (g-1)\times BS_k^{GSS}(n)+\left(\frac{T\times g}{k+n}-\frac{(g-2)\times BS_k^{GSS}(n)}{TR}\right)\times CR\times g & ,i\le\left\lfloor\frac{n}{g}\right\rfloor \\ (g'-1)\times BS_k^{GSS}(n)+\left(\frac{T\times g'}{k+n}-\frac{(g'-2)\times BS_k^{GSS}(n)}{TR}\right)\times CR\times g' & ,i=\left\lfloor\frac{n}{g}\right\rfloor,1<g'<g \\ BS_i^{GSS}(n)+\frac{T}{k+n}\times CR & ,i=\left\lfloor\frac{n}{g}\right\rfloor,g'=1 \end{cases} \quad (20)$$

단, 여기서 $1\le i\le G$ 이다.

GSS* 방식은 각 그룹들을 BubbleUp으로 서비스하기 때문에 각 그룹은 일정한 시간적 거리 $\frac{T\times g}{k+n}$을 두고, 서버로부터 주기적으로 메모리를 할당 받는다. 그리고, 사용자 요청들은 $CR$의 속도로 버퍼의 데이타를 사용한 후에 해당 메모리를 반환한다. 따라서, $i$번째 그룹이 마지막으로 서비스된 시점이 $\rho_i$인 경우에 $i$번째 그룹의 $t$ 시점에서의 메모리 요구량 $R_i^{GSS}(t,k,n)$은 수식(21)에서와 같이 서버로부터 $\rho_i$ 시점에 할당받은 메모리량($g\times BS_k^{GSS}(n)$ 또는 $g'\times BS_k^{GSS}(n)$)에서 사용자 요청들이 $t-\rho_i$ 시간 동안 $CR$의 속도로 반환한 메모리량($g\times CR\times(t-\rho_i)$ 또는 $g'\times CR\times(t-\rho_i)$)를 뺀 메모리량이다.

$$R_i^{GSS}(t,k,n)=$$

$$\begin{cases} g\times BS_k^{GSS}(n)-g\times CR\times(t-\rho_i) & ,i\le\left\lfloor\frac{n}{g}\right\rfloor \\ g'\times BS_k^{GSS}(n)-g'\times CR\times(t-\rho_i) & ,i=\left\lfloor\frac{n}{g}\right\rfloor,1\le g'\le g \end{cases} \quad (21)$$

단, 여기서 $1\le i\le G$ 이다.

GSS* 방식의 최소 메모리 요구량은 정리 2와 같이 전체 메모리 요구량의 최대치이다. 전체 메모리 요구량이 최대가 되는 시점은 $g$개의 버퍼로 구성된 그룹의 메모리 요구량이 최대가 되는 시점으로서, 이 시점은 각 그룹내에서 $(g-2)$개의 버퍼를 서비스한 후에 $(g-1)$번째 버퍼에 메모리를 할당한 시점이다[4]. 따라서, 이 시점은 서버가 그룹내에 있는 버퍼들을 서비스하기 시작하는 시점으로부터 $\frac{(g-2)\times BS_k^{GSS}(n)}{k+n}$시간이 지난 후의 시점이다. 수식(22)는 $g$개의 버퍼로 구성된 첫번째 그룹의 메모리 요구량이 최대가 되는 경우의 전체 메모리 요구량이다. 수식(23)은 모든 버퍼들이 한 번씩 서비스된 후에 첫번째 그룹의 메모리 요구량이 최대가 되는 시점인 $\frac{n\times T}{k+n}+\frac{(g-2)\times BS_k^{GSS}(n)}{TR}$을 수식(22)의 $t$에 대입하고, 이에 따라 수식(21)에서 $2\le i\le G-1$인 경우의 $\rho_i$로 $\frac{i\times g\times T}{k+n}$를 대입하며, $\rho_G$로 $\frac{n\times T}{k+n}$를 대입하여 전개한 것이다. 수식(23)은 $G$가 1보다 큰 경우에 GSS* 방식의 최소 메모리 요구량 $Mem^{GSS}_{min}(k,n,g)$이고, $G$가 1인 경우는 스윕* 방식과 같기 때문에 이 경우의 최소 메모리 요구량은 정리 3과 같다.

$$Mem^{GSS}_{min}(k,n,g)=$$

$$\begin{cases} \sum_{i=1}^{G} R_i^{GSS}(t,k,n)+R^{GSS}_{i,max}(k,n) & ,G>1 \quad (22) \\ (G-1)\times\left\{g\times BS_k^{GSS}(n)-\left(\frac{n\times T}{k+n}+\frac{(g-2)\times BS_k^{GSS}(n)}{TR}-\frac{g\times T\times(G+2)}{2\times(k+n)}\right)\times CR\times g\right\}+ \\ (g-1)\times BS_k^{GSS}(n)+\left(\frac{T\times g}{k+n}-\frac{(g-2)\times BS_k^{GSS}(n)}{TR}\right)\times CR\times g \\ \qquad ,G=\frac{n}{g} \\ (G-2)\times\left\{g\times BS_k^{GSS}(n)-\left(\frac{n\times T}{k+n}+\frac{(g-2)\times BS_k^{GSS}(n)}{TR}-\frac{g\times T\times(G+1)}{2\times(k+n)}\right)\times CR\times g\right\}+ \quad (23) \\ BS_k^{GSS}(n)\times(g+g'-1)+CR\times\left\{\left(\frac{T\times g}{k+n}-\frac{(g-2)\times BS_k^{GSS}(n)}{TR}\right)\times g-\frac{(g-2)\times g\times BS_k^{GSS}(n)}{TR}\right\} \\ \qquad ,G>\frac{n}{g},1\le g'<g \end{cases}$$


이 상 호

1992년 2월 경북대학교 컴퓨터과학과 학사. 1995년 2월 한국과학기술원 전산학과 석사. 1995년 3월 ~ 현재 한국과학기술원 전산학과 박사과정. 관심분야는 멀티미디어 DBMS, 주문형 비디오 시스템, 하이퍼미디어 시스템

문 양 세

1991년 2월 한국과학기술원 과학기술대 전산학과 학사. 1993년 2월 한국과학기술원 전산학과 석사. 1997년 3월 ~ 현재 한국과학기술원 전자전산학과 전산학 전공 박사과정. 1991년 3월 ~ 현재 현대전자산업(주) 통신사업본부 통신연구소. 관심분야는 데이타 마이닝, Knowledge Discovery, 저장 시스템, Access Method.

황 규 영

1973년 서울대학교 전자공학과 졸업(B.S). 1975년 한국과학기술원 전기 및 전자학과 졸업(M.S.). 1982년 Stanford University(M.S.). 1983년 Stanford University(Ph.D.). 1975년 ~ 1978년 국방과학연구소(ADD), 선임연구원. 1983년 ~ 1990년 IBM T.J. Watson Research Center, Research Staff Member. 1992년 ~ 1994년 한국정보과학회 데이타베이스 연구회(SIGDB) 운영위원장. 1995년 한국정보과학회 이사 겸 논문지 편집위원장. 1999년 ~ 2000년 한국정보과학회 부회장. Editior: the VLDB Journal, 1990년 ~ 현재 Editior: Distributed and Parallel Databases: An International Journal, 1991년 ~ 1995년 Editor: International Journal of Geographical Information Systems, 1994년 ~ 현재 Associate Editor: IEEE Data Engineering Bulletin, 1990년 ~ 1993년. 1998년 ~ 2004년 Trustee, The VLDB Endowment. 1999년 ~ 2005년 Steering Committee Member, DASFAA. 1999년 ~ 현재 한국과학기술원 전자전산학과 전산학전공 교수. 1999년 ~ 현재 첨단정보기술연구센터(과학재단 우수연구센터) 소장. 관심분야는 데이타베이스 시스템, 멀티미디어, GIS.

조 완 섭

1985년 2월 경북대학교 통계학과 학사. 1987년 2월 한국과학기술원 전산학과 석사. 1996년 2월 한국과학기술원 전산학과 박사. 1987년 2월 ~ 1990년 12월 한국전자통신연구소 연구원. 1996년 2월 ~ 1997년 2월 한국전자통신연구소 Post Doc. 1997년 ~ 현재 충북대학교 경영정보학과 조교수. 관심분야는 전자 상거래, 데이타 웨어하우스 & OLAP, CRM (고객관계관리), 데이타베이스, Web Databases, Multimedia Information Systems, 정보검색, GIS, MIS,Query Optimization and DBMS Performance, Graphical User Interfaces, Computer Graphics