실시간 멀티미디어 시스템에서의 캐슁을 위한 동적 버퍼 할당 기법

(Dynamic Buffer Allocation Scheme for Caching in Realtime Multimedia Systems)

권 진 백 † 염 헌 영 †† 이 경 오 †††

(Jin Baek Kwon) (Heon Young Yeom) (Kyung-Oh Lee)

요 약 멀티미디어 시스템에서 여러 가지 캐슁 기법들이 제안되어 왔다. 기존의 기법들은 캐쉬 적중률을 높이는 데에 초점을 맞추고 있는 반면, 캐슁 효과에 의해 절약된 디스크 대역을 활용하는 방법을 제시하고 있지는 않다. 멀티미디어 시스템에서는 서비스의 질을 보장하면서 동시에 얼마나 많은 사용자를 서비스할 수 있는 지가 시스템의 성능을 나타내는 가장 중요한 척도이다. 이 점에 착안해 캐슁의 장점을 살리면서 보장형 서비스를 제공하는 PSIC(Preemptive but Safe Interval Caching) 기법이 제안되었지만, 이 기법은 캐쉬 크기를 고정시킴으로써 시스템 환경의 변화에 대처할 수 없다는 문제를 가지고 있다. 본 논문에서, 우리는 보장형 서비스를 제공하면서 캐슁을 위해 메모리 버퍼를 동적으로 관리함으로써 접근 성향에 상관없이 시스템의 성능을 극대화시킬 수 있는 DIC(Dynamic Interval Caching) 기법을 제안한다. 그리고, PSIC 기법과의 실험적 비교를 통해, DIC가 캐쉬를 최적으로 할당한다는 것을 보였다.

Abstract Several caching schemes for realtime multimedia systems have been proposed, but they focus only on increasing the hit ratio without providing any means to utilize the saved disk bandwidth due to cache hits. One of the most important metrics in multimedia systems is the number of clients that the systems can service simultaneously guaranteeing Quality of Service(QoS). Preemptive but Safe Interval Caching(PSIC) was proposed as a caching scheme which makes it possible to provide deterministic QoS.. However, it has no ability to adapt to the change of system environments since it has no mechanism to change the cache size. In this paper, we present a new caching scheme, Dynamic Interval Caching(DIC), which maximizes the performance, regardless of the change of system environments, providing hiccup-free service, by managing memory buffers dynamically. And it is demonstrated that DIC allocates buffer cache optimally, by comparing with PSIC through trace-driven simulations.

1. 서 론

멀티미디어 시스템의 근본적인 문제는 텍스트나 이미지 데이타와는 그 특성이 다른 오디오나 비디오 데이타를 다루어야 한다는 데 있다. 오디오와 비디오 스트림(stream)들은 미디어 퀀타(quanta)들(비디오 프레임 또는 오디오 샘플)의 연속으로, 제 시간에 연속적으로 재생되어야 의미를 갖는 데이타들이다. 이런 특성을 갖는 데이타를 CM(Continuous Media) 개체라고 한다. 어떤 CM 개체에 대한 요청이 왔을 때, 서버는 그 스트림의 연속적 배달(delivery)을 보장하기 위해 다양한 자원들(디스크 대역폭, 메모리, 네트워 대역폭 등)을 예약한다. 이 때, 예약 가능한 자원량은 서버가 서비스의 질(QoS)을 보장하면서 서비스할 수 있는 스트림들의 개수에 의존한다.

디스크로부터 읽혀진 각 블록은 전송되기 전에 메모

· 본 논문은 두뇌한국21 사업의 지원을 받았음.

† 비 회 원 : 서울대학교 전산과학과
jbkwon@dcslab.snu.ac.kr
†† 종신회원 : 서울대학교 전산과학과 교수
yeom@dcslab.snu.ac.kr
††† 비 회 원 : 선문대학교 컴퓨터정보학부 교수
leeko@rainbow.sunmoon.ac.kr

논문접수 : 1999년 4월 27일
심사완료 : 2000년 1월 27일

리 버퍼에 로드(load) 된다. 이것을 사전 버퍼링(read-ahead buffering)이라고 한다. 초기 연구에서는 각 클라이언트(스트림)에 전용 버퍼가 할당된다고 가정했었다. 즉, 한 스트림에 할당된 RA-버퍼(Read-Ahead buffer)는 다른 스트림이 사용할 수 없다. CM 개체는 일반적으로 높은 디스크 대역을 필요로 하므로, 이런 CM 개체를 저장하고 있는 멀티미디어 시스템에서 디스크 I/O를 줄이는 것이 중요한 문제가 된다. 따라서, 최근 연구[9, 11]에서는 디스크 I/O를 줄이기 위한 방법으로 전통적인 저장 시스템[8, 15]과 비슷한 공유 버퍼(shared buffer) 캐쉬를 제안했다. CM 개체 접근 성향(access pattern)의 특성 때문에, LRU(Least-Recently Used), MRU(Most-Recently Used) 등 전통적인 대치(replacement) 기법은 멀티미디어 시스템에서 높은 적중률(hit ratio)을 얻을 수 없다는 것이 알려져 있다.[3].

멀티미디어 시스템에서 하나의 스트림은 하나의 CM 개체를 처음부터 끝까지 순차적으로(sequentially) 읽는다. 즉, 스트림이 앞으로 읽을 데이타 블록과 그 블록을 읽은 시간은 사전에 결정된다. 이러한 CM 개체에 대한 접근 성향을 고려한 캐슁 기법들이 제안되어 왔다[6, 12]. 그러나, 이 기법들은 적중률(hit ratio)을 높이는 데에만 초점을 두고, 멀티미디어 시스템이 사용자에게 제공해야 하는 QoS(Quality of Service)를 고려하고 있지 않다. 이 점에 착안해서, 우리는 캐슁에 의해 멀티미디어 시스템의 성능을 향상시키면서 동시에 보장형 서비스(deterministic service)를 제공하는 PSIC(Preemptive but Safe Interval Caching) 기법을 제안했다[14]. 그러나, 이 기법은 캐쉬-버퍼를 할당하는 방법이 유동적이지 못해서, 시스템 환경의 변화에 대처하지 못하는 단점을 가지고 있다. 본 논문에서는, 보장형(deterministic) 서비스를 제공하면서 캐슁을 위한 메모리 버퍼를 동적으로 관리함으로써 접근 성향의 변화에 상관없이 시스템의 성능을 극대화시키는 캐슁 기법으로 DIC(Dynamic Interval Caching) 기법을 제안한다.

본 논문의 나머지 부분은 다음과 같이 구성된다. 2절에서 관련 연구를 소개하고, 3절에서는 본 논문이 가정하고 있는 시스템 모델을 설명한다. 우리가 제안하는 DIC의 동적인 캐쉬 버퍼 할당법과, DIC의 알고리듬과 타당성을 4절에서 설명한다. 5절에서는 실험 환경과 PSIC과의 비교 결과를 보여주고, 마지막 6절에서 총괄적 결론을 내린다.

2. 관련 연구

Dan등은 *Interval Caching*이라는 캐슁 기법을 제안했다[6]. 이 기법은 멀티미디어 서버에 적합하게 고안된 캐슁 기법으로, 시간적 지역성(temporal locality)과 시스템 부하의 변동(fluctuation)을 이용한다. 스트림 S_i가 읽은 블록을 다음에 스트림 S_j가 읽는다면, 두 개의 스트림 S_i와 S_j는 "*연속되었다(consecutive)*"고 정의한다. 한 쌍의 연속된 스트림들을 각각 선행 스트림(preceding stream)과 후행 스트림(following stream)이라고 한다. 이 선행 스트림과 후행 스트림 사이의 데이타를 *interval*이라고 정의한다. 하나의 *interval*을 캐슁하면, 그 *interval*의 후행 스트림은 디스크 I/O 없이 캐쉬-버퍼에서 필요한 블록을 읽게 된다. *interval Caching*은 *interval*을 캐슁의 단위로 사용할 것을 제안한 기법이다. 하지만, 이 기법에 적중률을 높이기 위한 교체 기법은 포함되어 있지 않다.

Ozden등은 BASIC과 DISTANCE라는 캐슁 기법을 제안했다[12]. 이 기법들은 캐쉬 적중률을 높임으로써, 디스크 I/O의 회수를 줄였다. DISTANCE는 *interval* 단위 캐슁 기법으로서, 적중률을 높이기 위한 교체기법을 제안했다. *interval*의 크기에 따라 그 *interval*의 가치를 판단하고, 교체 시에 캐슁된 *interval*들 중 크기가 가장 큰 것이 교체대상(victim)으로 선정된다. BASIC은 블록 단위 캐슁으로서, TTNR(Time To Next Reference) 값에 의해 각 블록의 가치를 평가한다. 교체 시에 서비스되고 있는 스트림에 의해 가장 오랫동안 읽히지 않을 블록(즉, 가장 큰 TTNR 값을 가지는 블록)을 교체대상으로 선택한다. 그리고, 만약 존재하는 스트림에 의해 읽히지 않을 블록(TTNR 값을 알 수 없는 블록)들이 있다면, 가장 높은 옵셋 율(offset-rate ratio)을 가진 블록을 교체대상으로 선택한다. 예를 들면, 한 블록의 크기가 32KB일 때, 비트율 1.5Mbps인 비디오의 열 번째 블록은 288(=32*10-32)의 옵셋을 가지고, 1.536초(=288KB/1.5Mbps)의 옵셋 율을 가진다. 이 두 기법은 오직 적중률을 높이는 데에만 초점을 맞추고 있다. 따라서, 캐슁에 의해 절약된 디스크 대역을 이용해 더 많은 스트림들을 서비스할 때, 보장형(deterministic) 서비스를 제공하지 못한다.

[14]에서, 우리는 캐슁에 의해 멀티미디어 시스템의 성능을 향상시키면서 동시에 보장형 서비스(deterministic service)를 제공하는 PSIC(Preemptive but Safe *interval* Caching) 기법을 제안했다. *interval* 단위 캐슁에서 교체 시에 *interval*의 크기만을 고려한다면, 교체된 *interval*의 후행 스트림이 디스크에 과부하를 일으키거나 RA-버퍼를 확보하지 못할 가능성이 존재한다.

그런 상황이 발생한다면, 교체된 스트림 또는 다른 어떤 스트림들은 제시간에 필요한 블록을 읽지 못하는 "떨림 현상"을 경험하게 된다. PSIC은 "떨림 없는(hiccup-free)" 서비스를 제공하기 위해 DISTANCE의 교체 기법에 3.3절에서 설명할 수용 제어(admission control) 기법을 도입했다. 즉, *interval*의 크기가 큰 순서대로 교체 대상 후보를 선정하고, 그 후보 *interval*의 후행 스트림이 디스크에서 보장된 서비스 받기 위해 필요한 자원을 확보할 수 있을 때(디스크 제약조건과 버퍼 제약조건이 만족될 때) 실제로 그 후보 *interval*을 교체한다. 하지만, PSIC은 *interval*을 저장할 캐쉬-버퍼의 비율을 시스템 설치 시에 고정해 놓기 때문에, 시스템 환경의 변화에 따라 적절히 버퍼의 비율을 조정하지 못한다는 단점을 가지고 있다.

3. 시스템 모델

3.1 시스템 구조

본 논문에서 가정하고 있는 시스템 모델은 디스크를 접근하는 I/O 관리자와 버퍼 관리자, 네트워크 관리자로 구성된다. 멀티미디어 데이타 개체는 동시에 요청된 스트림들을 지원하기 위해 여러 개의 디스크에 걸쳐서 스트라이핑(striping)되며, 3.2 절에서 설명한다. 메모리 버퍼는 캐쉬-버퍼(cache buffer)와 RA-버퍼(read-ahead buffer)로 *논리적*으로 분리된다. 네트워크로 보내질 데이타는 RA-버퍼에 보관되고, 이미 보내진 데이타는 캐쉬-버퍼에 보관된다. 실제로 데이타 블록이 RA-버퍼에서 캐쉬-버퍼로 이동하는 것은 아니고, 단지 캐쉬 플래그(flag)들만 셋(set)된다. 네트워크 관리자(network manager)는 필요한 데이타 블록을 메모리 버퍼(캐쉬-버퍼 또는 RA-버퍼)에서 읽어서 네트워크를 통해 사용자들에게 보낸다. 네트워크 관리자가 필요한 데이타를 제시간에 메모리 버퍼에서 찾을 수 없을 때 "떨림(hiccup)" 현상이 발생한다.

3.2 멀티미디어 개체의 스트라이핑(striping)

일반적으로, 굵은(coarse-grained) 스트라이핑 기법이 가는(fine-grained) 기법보다 성능이 좋다. 이 것은 가는 스트라이핑 기법은 과도한 디스크 검색(seek) 때문에 디스크 대역폭을 낭비하기 때문이다[1, 13]. 또한, CTL(Constant Time Length) 기법이 CDL(Constant Data Length)[4, 5] 기법보다 더 좋은 성능을 보인다. CDL 기법이 더 많은 버퍼를 요구한다는 사실 때문이다[7]. 따라서, 본 논문에서는 기본 스트라이핑 기법으로서 굵은 CTL을 가정한다. CBR (Constant Bit Rate)의 경우에는, CDL과 CTL이 동일하다.

디스크에서 데이타를 읽을 때 소요되는 오버헤드는 디스크의 검색시간(seek time)과 회전 시간(rotational latency)에 의해 결정되는 데, 이런 오버헤드는 기계적인 동작을 요구하므로 그 것을 줄이는 데는 한계가 있고, 현재 기술은 거의 그 한계에 왔다고 볼 수 있다[15]. 또한 검색시간이나 회전시간을 줄이는 것은 하드웨어 기술로서 본 논문의 범위를 벗어난다. 따라서, 검색 시간 또는 회전 시간을 줄이는 것보다 필요한 데이타를 읽는데 요구되는 검색 회수를 줄이는 것이 성능 향상을 위해 더 합리적이다. 한 번의 디스크 접근으로 한 라운드 동안 필요한 데이타를 모두 읽을 수 있도록 함으로써, 디스크 검색 회수를 줄일 수 있다. 하나의 스트라이핑 단위의 재생 시간 DT를 라운드 길이 T와 같다고 하면, 각 스트림은 정확하게 한 라운드에 한번의 디스크 접근이 필요하게 된다. 그러므로, 각 CM 개체를 같은 재생 시간을 가지는 세그먼트들(segments)로 쪼개서 디스크 배열(array)에 저장한다. 한 디스크에 더 많은 세그먼트가 저장됨으로써 발생하는 부하 불균형(load imbalance) 확률을 줄이기 위해, 각 개체의 첫 번째 세그먼트를 저장할 때 교차(staggered) 스트라이핑 기법[2]을 사용한다. 이런 스트라이핑 기법을 RTL(Round Time Length) 스트라이핑[10]이라고 한다([그림 1]).

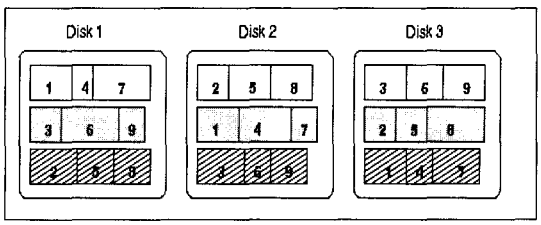

그림 1 RTL(Round Time Length) Striping

k개의 활동중인 스트림이 디스크 i에서 서비스받고 있다면, 이 스트림들은 다음 라운드에 디스크 $(i+1) \bmod n$에서 서비스받게 된다. 여기서, n은 RTL 스트라이핑 하에서의 디스크의 개수를 나타낸다. 디스크 배열의 모든 디스크의 성능이 동일하다면, 시스템이 라운드 j에서 스트림들의 집합을 문제없이 서비스했다면, 라운드 $j+1$에서도 문제는 발생하지 않을 것이다. 하나의 디스크에서 서비스 받고 있는 모든 스트림을 서비스 그룹이라 하고, 디스크 배열에 n개의 디스크가 있다면, n개의 서비스 그룹이 존재한다.

3.3 수용 제어

서비스 중인 스트림의 수를 m, 한 세그먼트의 재생

시간을 DT이라 하자, 스트림 i가 요청한 개체의 가장 큰 세그먼트의 재생율은 $R_i(DT)$이다. 새로운 스트림을 받아들이기 위해서, 다음 두 조건이 만족되어야 한다 [16].

버퍼 제약조건:

$$\text{total amount of data to be stored} \le \text{total buffer size}$$
$$2 \cdot \sum_{i=1}^{m} R_i(DT) \cdot DT \le \text{total buffer size}$$

디스크 제약조건:

$$\text{seek time} + \text{transfer time} + \text{rotational latency} + \text{other overhead} \le DT$$
$$\equiv \text{seek}(k) + \sum_{i=1}^{m}\left(\frac{R_i(DT)\cdot DT}{\text{transfer rate}} + \text{rotational latency} + \varepsilon\right) \le DT$$

여기서 k는 가장 부하가 높은 디스크에서 읽어야 하는 블록의 수($\approx \lceil \frac{m}{d} \rceil$, d는 디스크의 개수)이다.

4. Dynamic Interval Caching (DIC)

이 절에서는 *interval* 단위 캐싱에서 캐쉬 버퍼 할당법이 중요한 이유를 설명하고, 보장형 서비스를 제공하면서 시스템 성능을 극대화시키도록 캐쉬-버퍼를 동적으로 할당하는 DIC 기법을 소개한다. 그리고, DIC의 알고리듬과 타당성도 여기서 다룬다.

4.1 *Interval* 단위 캐싱과 스트림의 상태 전이

같은 CM 개체를 읽는 *연속된* 두 개의 스트림이 있다고 하자. 이 때, 앞서는 스트림(즉, 선행 스트림)이 읽은 모든 데이타 블록을 후행 스트림이 읽을 때까지 메모리에 보관한다면, 후행 스트림이 네트웍으로 보내야 하는 모든 데이타는 메모리에 있게 된다. 따라서, 그 후행 스트림은 필요한 데이타를 디스크 I/O 없이 사용자에게 전송할 수 있고, 절약된 디스크 대역을 이용해 더 많은 스트림을 서비스할 수 있다. 이 것이 *interval* 단위 캐싱의 기본 개념이다. 이 때, 선행 스트림은 사용자에게 이미 전송했지만 후행 스트림은 아직 전송하지 않은 데이타 블록들을 *interval*이라고 하고 이 *interval*을 저장하는 메모리 버퍼를 *캐쉬-버퍼*라고 한다. *Interval* 단위 캐싱 기법을 사용하는 시스템은 전체 메모리 버퍼를 RA-버퍼와 캐쉬-버퍼 두 가지 형태로 사용한다. 여기서 캐쉬-버퍼의 할당은 디스크나 메모리 버퍼 어느 한 쪽 자원에서 병목(bottleneck)이 발생하지 않도록 결정되어야 한다. 따라서, 할당된 전체 캐쉬-버퍼의 크기는 시스템의 성능을 좌우하는 중요한 파라미터이므로, 캐쉬-버퍼를 할당하는 방법은 신중히 고려되어야 한다.

멀티미디어 시스템에서 서비스하는 모든 스트림은 그 스트림이 서비스 받는 동안 여러 가지 상태에 놓인다. [그림 2]는 스트림의 상태 전이 그림(state transition diagram)이다. 사용자가 어떤 CM 개체에 대한 시청을 요청하게 되면 새로운 스트림이 생성되어 그 스트림은 먼저 AC(Admission Control) 상태에 들어가고, 시스템은 그 스트림에 대한 서비스 조건을 검사한다. 조건이 만족되지 않으면 REJECT 상태로 끝난다. 조건이 만족되고 그 스트림을 후행 스트림으로 하는 *interval*이 생성되며 이 *interval*을 저장할 캐쉬-버퍼의 할당이 가능하면, 스트림은 *interval*이 캐쉬-버퍼에 로드될 때까지 PREPARE 상태에서 머무르면서 디스크를 접근한다. *interval*의 길이가 l이라고 하면, l 라운드 후 후행 스트림은 더 이상 디스크로부터 데이타를 읽지 않고 캐쉬-버퍼로부터 데이타를 서비스하는 BYCACHE 상태로 전이된다. 반면 그 *interval*을 위한 캐쉬-버퍼를 할당할 수 없으면, 일단 BYDISK 상태로 가서 디스크에서 직접 데이타를 읽는다. PREPARE 상태나 BYCACHE 상태에 있는 스트림은 자신이 후행 스트림으로 있는 *interval*이 메모리로부터 교체되는 경우에는 BYDISK 상태로 전이될 수 있다. 반대로, 시스템을 떠나는 스트림들이 반환하는 메모리 공간을 이용해서 캐싱되지 않은 *interval*을 캐싱할 수 있는 경우에, 이 *interval*의 후행 스트림은 BYDISK 상태에서 PREPARE 상태로의 전이가 발생한다.

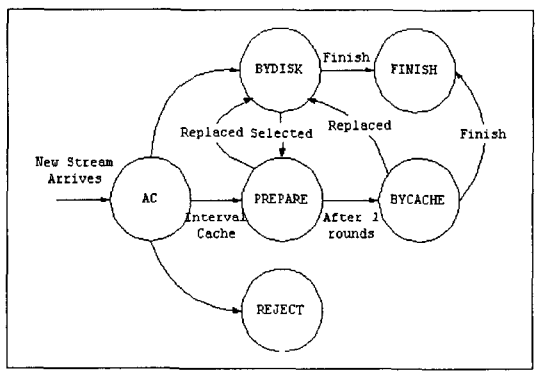

그림 2 스트림의 상태 전이 그림

4.2 동적 캐쉬 버퍼 할당법

캐쉬-버퍼를 할당하는 방법에는 정적으로(statically) 캐쉬-버퍼를 할당하는 방법과 동적으로(dynamically) 할당하는 방법이 있다. 정적인 할당법에서는 캐쉬-버퍼 풀과 RA-버퍼 풀을 따로 가지고 각 풀이 가질 수 있는 최대 메모리 크기가 미리 결정되고 변하지 않으므로, 두 풀 사이에서 메모리 버퍼의 이동이 발생하지 않는다. 반면에 동적인 할당법에서는 통합된 하나의 버퍼 풀을 갖고 스트림의 요구에 따라 캐쉬-버퍼과 RA-버퍼를 둘

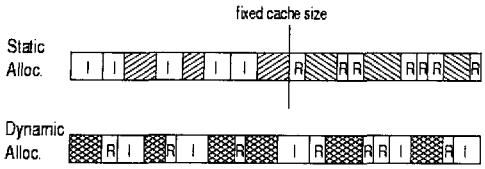

그림 3 정적인(static) 캐쉬-버퍼 할당과 동적인(dynamic) 캐쉬-버퍼 할당: I는 캐쉬-버퍼로 사용되는 메모리 버퍼를, R은 RA-버퍼로 사용되는 메모리 버퍼를 나타낸다.

다 할당한다. [그림 3]은 이 두 개의 캐쉬 버퍼 할당법을 사용했을 경우의 전체 메모리 상태를 보여준다. 그림에서 보듯이, 정적인 할당법(Static Alloc.)에서 사선으로 된 영역은 캐쉬-버퍼로 할당할 수 있는 메모리 버퍼를, 역사선으로 된 영역은 RA-버퍼로 할당할 수 있는 메모리 버퍼를 나타낸다. 동적인 할당법(Dynamic Alloc.)에서 사선과 역사선이 합쳐진 영역은 캐쉬-버퍼 또는 RA-버퍼로 할당할 수 있는 메모리 버퍼를 나타내고, 두 할당법에서 I와 R은 각각 캐쉬-버퍼로 할당된 영역과 RA-버퍼로 할당된 영역을 나타낸다. 그림에서 전체 크기는 물리적인(physical) 메모리 전체의 크기를 의미하고, RA-버퍼와 캐쉬-버퍼들이 차지하는 메모리 크기를 K라고 하면 사용가능한 메모리 크기(MEM)는 $TOTAL - K$가 된다. 정적 할당법에서는 캐쉬-버퍼 풀(pool)과 RA-버퍼 풀을 따로 가지고 각 풀이 가질 수 있는 최대 메모리 크기는 P_C와 P_R ($TOTAL=P_C+P_R$)로 미리 결정되고 변하지 않는다. 캐쉬-버퍼로 할당된 메모리 크기를 A_C, RA-버퍼로 할당된 메모리 크기를 A_R라고 하면, 캐쉬-버퍼와 RA-버퍼로 할당가능한(즉, 사용되지 않고 있는) 메모리 크기는 각각 P_C-A_C ($=MEM_C$)와 P_R-A_R($=MEM_R$)이고 $K=A_C+A_R$이다. 이 때, $P_C \geq A_C$과 $P_R \geq A_R$는 항상 만족된다. 따라서, 캐쉬-버퍼 풀에 할당할 수 있는 캐쉬-버퍼가 있다고 하더라도(즉 $P_C-A_C > 0$) 그 것이 RA-버퍼로 할당될 수 없고 반대의 경우도 마찬가지이기 때문에, 캐쉬 크기가 정적으로 P_C로 고정되어있다고 볼 수 있다. 반면에 동적인 할당법에서는 통합된 하나의 버퍼 풀을 갖고 스트림의 요구에 따라 캐쉬-버퍼와 RA-버퍼를 둘 다 할당한다. 즉, $TOTAL = MEM + A_C + A_R$이다 (정적할당법의 경우, $TOTAL = MEM_C + MEM_R + A_C + A_R$). 따라서, 동적 할당법에서 캐쉬 크기를 캐쉬-버퍼로 할당된 메모리의 크기(즉 A_C)로 정의할 때, 캐쉬 크기는 실행 중에 캐쉬-버퍼로 할당된 메모리 양에 따라 유동적으로 변하게 된다. 다시 말하면, 동적 할당법은 메모리 전체를 캐쉬-버퍼로 할당할 수도 있고 역으로 RA-버퍼로 할당할 수도 있다.

정적 캐쉬 버퍼 할당법을 사용하는 PSIC은 캐쉬 크기를 *최적 캐쉬 크기*로 고정시킨다[14]. 여기서 최적 캐쉬 크기(optimal cache size)는 주어진 시스템 환경에서 서버의 성능을 극대화시키는 캐쉬 크기를 뜻한다 최적 캐쉬 크기는 사용자의 요청 성향(CM 개체의 인기도)과 CM 개체의 특성(비트율과 길이), 기타 시스템 파라미터들(한 라운드의 길이, 디스크 개수, 메모리 크기 등)의 함수로서, 주어진 시스템 환경에서 시뮬레이션을 통해 찾을 수 있다. [14]의 PSIC은 정적 캐쉬 버퍼 할당법을 사용하기 때문에 다음 세 가지 단점을 가진다. 첫째, *최적 캐쉬 크기*는 시스템 설치 시에 반복적인 시뮬레이션을 통해 결정해야 한다. 둘째, *최적 캐쉬 크기*는 사용자의 요청 성향과 CM 개체의 특성에 따라 다르고, 이 것들을 시스템 설치 시에 미리 알기가 어렵다. 마지막으로, 멀티미디어 서버의 특성 상 요청 성향은 시간에 따라 변하고 이에 따라 최적 캐쉬의 크기가 변한다는 것이다.

따라서, 캐쉬 크기를 미리 고정시키는 것보다는 요청 성향과 저장된 개체의 특성에 따라 동적으로 최적 값으로 변할 수 있도록 하는 것이 바람직하다. PSIC과는 달리 DIC 기법은 동적으로 캐쉬의 크기를 조정함으로써 시스템 환경 변화에 대처할 수 있다.

하나의 스트림이 캐쉬-버퍼에서 서비스 받기 위해서는 그 스트림이 뒤따르는 *interval*이 상주할 캐쉬-버퍼가 필요하고, 디스크에서 서비스 받기 위해서는 디스크 대역폭과 RA-버퍼가 필요하다. SCAN 디스크 스케쥴링[8]을 사용할 경우 두 라운드의 데이타를 저장할 공간이 RA-버퍼로서 사용된다(double buffering). 대부분의 경우 *interval*의 길이는 두 라운드보다 크므로, *interval*을 캐싱하는 것은 디스크 자원을 절약하지만 많은 메모리를 요구한다. 다시 말하면, 두 라운드보다 긴 *interval* 전체를 저장하기 위한 캐쉬-버퍼의 메모리 요구량은 RA-버퍼의 메모리 요구량보다 크다. 따라서, 새로운 요청이 왔을 때 메모리에서 병목이 발생하면 캐쉬-버퍼로 할당된 메모리 공간(캐쉬 크기)을 줄이고 디스크에서 병목이 발생하면 늘이는 것이 타당하며, 이 것이 DIC의 기본 개념이다.

DIC는 새로운 스트림에 대한 수용 제어(admission control)시에 캐쉬의 크기를 조정함으로써 그 스트림을 서비스할 수 있는 모든 가능성을 타진한다. 메모리에서 병목이 발생한 경우, DIC가 캐쉬-버퍼의 반환과 RA-

버퍼의 할당을 통해 캐쉬 크기를 조정하는 모습이 [그림 4]에 보여진다. 앞에서도 언급했듯이 어떤 스트림이 디스크에서 서비스 받게 될 때(BYDISK) 필요한 RA-버퍼는 두 라운드의 데이타를 저장할 공간만을 요구한다. 반면에 *interval*은 일반적으로 두 라운드보다 길기 때문에, *interval*을 저장하기 위한 캐쉬-버퍼는 RA-버퍼 보다 많은 메모리 공간을 요구한다. *Interval x*를 저장한 캐쉬-버퍼에서 서비스받고 있는 *stream x*를 디스크에서 서비스 받도록 하면(BYCACHE→BYDISK), RA-버퍼의 크기와 *interval*의 크기(즉 캐쉬-버퍼의 크기)의 차이에 따라 [그림 4]의 왼쪽 그림에서 빗금 친 부분만큼 메모리 공간이 남게 된다. 이 때, *stream x*의 디스크 제약조건이 만족되어야 한다. 이 공간을 새 스트림을 서비스하기 위해 필요한 버퍼로 사용할 수 있다. 오른쪽 그림은 한 개의 *interval*을 교체하는 것만으로는 새 스트림을 받아들일 수 없을 경우, 여러 개의 *interval*들을 교체한 후의 메모리의 상태를 보여준다.

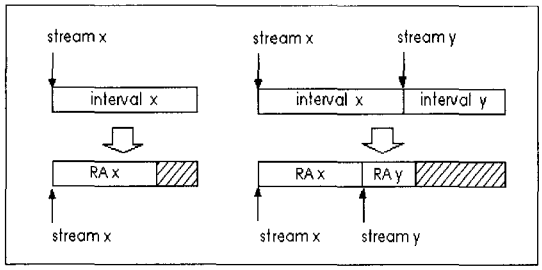

그림 4 버퍼 병목이 발생한 경우, 캐쉬 크기 조정 전후의 모습: 빗금 친 부분은 새 스트림이 사용할 수 있는 메모리 공간이다. 여기서 RA x는 interval x의 후행 스트림(stream x)을 디스크로부터 서비스하기 위해 요구되는 RA 버퍼의 크기이다.

디스크에서 병목이 발생하면 캐싱되지 않은 *interval*을 캐싱해서 그 것의 후행 스트림이 캐쉬-버퍼에서 서비스 받도록 함으로써 디스크 부하를 줄여야 한다. DIC는 피할 수 있는 디스크 병목을 방지하기 위해 두 가지 방법을 사용한다. 첫 번째는 새로운 스트림을 가능하면 캐쉬-버퍼에서 서비스하는 것이다. 처음에(시스템이 비어 있을 때) 요청된 스트림들은 메모리가 전부 할당될 때까지 캐쉬-버퍼에서 서비스될 것이다. 할당할 메모리가 없어서 버퍼 병목이 생기는 순간

캐쉬 크기는 전체 메모리 크기에 가까울 것이고, 그 이후에 들어오는 스트림들은 [그림 4]처럼 캐쉬 크기를 줄이게 된다. 디스크 제약조건이 만족된 스트림만이 캐쉬-버퍼를 반환시킬 수 있기 때문에 디스크 병목이 생기기 시작하는 순간부터는 캐쉬 크기가 더 이상 작아지지는 않을 것이다. 두 번째 방법은 스트림 종료시 반환되는 메모리 버퍼를 캐쉬-버퍼로 할당해서 캐싱되지 않은 *interval*을 캐싱하는 것이다. 선택된 *interval*의 길이만큼의 라운드가 지난 후에 그 *interval*의 후행 스트림은 캐쉬-버퍼에서 데이타를 읽기 때문에(BYDISK→PREPARE→BYCACHE), 후행 스트림에게 할당되었던 디스크 대역을 새로운 스트림을 위해 사용할 수 있다.

표 1 알고리듬을 설명하기 위한 기호들

INT_i	i번째 *interval*
RA_i	i번째 스트림이 BYDISK 상태에서 요구하는 RA-버퍼
TOTAL	전체 메모리 크기
MEM	메모리에서 비어 있는 공간의 크기 (초기값 *TOTAL*)
STM_i	i번째 스트림
CLIST	캐쉬-버퍼에서 서비스 받는 스트림들의 리스트
DLIST	디스크에서 서비스 받는 스트림들의 리스트
ILIST	캐싱되어 있는 *interval*들의 리스트

4.3 알고리듬

여기에서 DIC의 구체적인 알고리듬을 설명한다. 설명과 이해의 편의를 위해 사용될 기호는 [표 1]에서 정의된다.

앞에서 언급했듯이, DIC는 새로운 요청에 대한 수용 제어 시에 시스템의 상황을 보고 캐쉬 크기를 조정한다. 이렇게 조정된 캐쉬 크기는 그 스트림을 서비스할 수 있는 가능성을 높일 수 있다. [그림 5]는 새로운 스트림에 대한 수용 제어 과정을 보여주고 있다. 그림에서 AC_DISK()와 AC_BUF()는 3.3절의 디스크 제약조건과 버퍼 제약조건을 검사하는 루틴이다. 조건이 만족되면 Yes를 아니면 No를 반환한다. AC_DISK()와 AC_BUF()를 만족하면 새 스트림은 기본적으로 RA-버퍼는 할당받고 디스크에서 서비스받는 것이 보장된다. 왜냐하면, 캐쉬-버퍼에서 서비스 받을 수 있는 스트림이라도 PREPARE상태에서 디스크에서 최초의 *interval*을 캐쉬-버퍼로 로드해야 하기 때문이다. 먼저 디스크 조건이 만족되는 지를 검사해서, 만족되지 않으면 요청을 거부(REJECT)한다. AllocCache()는 디스크 조건과 버퍼 조건이 만족되고 *interval*($=INT_{new}$)이 캐싱될 메모리

공간이 남아 있는 경우(*INT<=MEM*)에 수행되고, 다른 *interval*들을 교체하지 않으므로 다른 스트림의 상태에 영향을 주지 않는다. Replace()은 새 스트림의 *interval*이 캐싱될 공간이 없는 경우에 수행된다. INT_{new}를 저장할 캐쉬-버퍼를 할당하기 위해, 캐싱되어 있는 *interval*들의 리스트인 *ILIST*에서 크기가 INT_{new}보다 크고 그 것의 후행 스트림이 디스크 대역폭과 RA-버퍼를 확보할 수 있는 가장 큰 *interval*이 교체된다. 가능한 가장 큰 *interval*을 교체하는 것은 크기가 큰 *interval*일수록 교체로 얻는 이득이 더 크기 때문이다. Replace()에 의해 캐쉬의 크기는 교체되는 *interval*과 새 *interval*의 크기 차이만큼 감소한다. 그 다음 AllocCache()를 수행해서 캐쉬-버퍼를 할당하고 새 스트림은 캐쉬에서 서비스받게 된다. 후행 스트림이 AC_DISK()를 만족시키는, 즉 교체시킬 수 있는 *interval*이 없을 때는 AllocRA()을 통해 RA-버퍼를 할당하고 새 스트림을 디스크에서 서비스한다. 이 때, *캐쉬 크기*에는 변화가 없다.

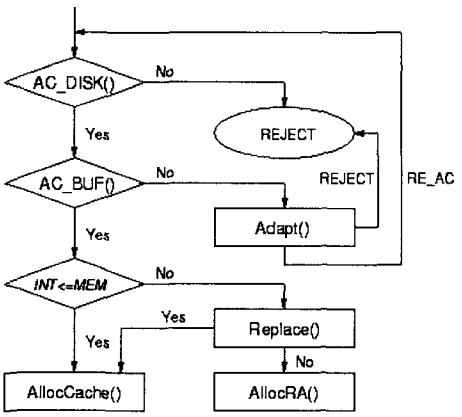

그림 5 수용 제어 과정

[그림 6]의 Adapt()가 DIC의 핵심이라고 할 수 있는 부분이다. 그림에서 AC_DISK(STM_{old})는 STM_{old}가 디스크에서 서비스받을 경우 디스크 제약조건을 만족시킬 수 있는 지를 테스트한다. Adapt()의 작업은 캐쉬 크기를 줄여서(기존의 *interval*들을 교체시켜서) 새 스트림을 위한 메모리 공간을 확보해 주는 역할을 한다. 어떤 *interval*도 교체될 수 없다면 요청은 거부될 것이고(REJECT 상수 반환), 하나의 *interval*이 교체되면 수용 제어를 다시 시작하게 된다(RE_AC 상수 반환). 따라서, Adapt()에 의해 수용 제어 과정은 재귀적으로(recursively) 수행된다. 이 과정 중에 결국 수용이 불가능하다는 판단을 내릴 수도 있지만, 수용 제어가 여러 번 수행되는 중간에 수용될 가능성이 있다. Adapt()가 아니라면 어차피 거부될 요청이었으므로, 수용 가능성을 줄 수 있다는 것으로 충분한 의미가 있다. 게다가, DIC가 최적 캐쉬 크기를 계속해서 유지할 수 있는 능력을 주는 가장 중요한 부분이 바로 Adapt()이다.

```
PROCEDURE Adapt()
BEGIN
   foreach INT_i in ILIST
   // repeat statements to "end foreach" for each intervals in
      ILIST,
   // from the largest interval to the smallest interval
      STM_old = Follow(INT_i); // following stream of INT_i
      if ( AC_DISK(STM_old) )
           Move STM_old from CLIST to DLIST;
           MEM := MEM + Size(INT_i);
           cache_size := cache_size - Size(INT_i);
           Delete INT_i from ILIST ;
           return RE_AC; // restart admission control
      end if
   end foreach
   return REJECT;
END
```

그림 6 Adapt(): 버퍼 자원이 부족할 경우에 수행되며, 할당된 캐쉬 크기를 줄여서 사용할 수 있는 메모리 버퍼를 늘여주는 역할을 한다. ([표 1] 참조)

```
PROCEDURE OnFinish(STM_fin)
BEGIN
   if ( STM_fin has serviced from cache buffer )
           MEM := MEM + Size(INT_fin) ;
   else  MEM := MEM + Size(RA_fin) ;
   INT_sel = the smallest interval which is not cached;
   STM_sel = Follow(INT_sel);
   if ( Size(INT_sel) <= MEM + Size(RA_sel))
           Move STM_old from DLIST to CLIST;
           MEM := MEM - Size(INT_sel) + Size(RA_sel);
           cache_size := cache_size + Size(INT_sel) - Size(RA_sel);
           Insert INT_i to ILIST ;
   end if
   return;
END
```

그림 7 OnFinish(): 스트림이 종료될 때마다 수행되는 프로시져로서, 기존의 캐싱되지 않은 interval 중에 가장 작은 것을 선택해 캐싱한다. ([표1] 참조)

캐쉬-버퍼에서 서비스를 받던 한 스트림이 종료되면 그 스트림이 후행 스트림으로 있던 *interval*이 캐쉬-버퍼를 반환한다. 즉, 반환된 메모리만큼을 가용 메모리 공간에 더해주고 현재 디스크에서 서비스되고 있는 스트림들 중 캐쉬-버퍼에서 서비스할 스트림을 선정한다. 이 때, 크기가 가장 작은 *interval*의 후행 스트림이 선택된다. 어떤 스트림이 종료될 때 호출되는 OnFinish()를 [그림 7]에 주어져 있고, DIC는 이 OnFinish()를 통해 디스크 병목을 미리 방지할 수 있다.

수용 제어 과정에서 캐쉬 크기를 조정하는 데 따른 오버헤드는 Replace()와 Adjust()를 제외하고는 상수 시간이 걸린다. Replace()는 캐쉬-버퍼가 할당된 n개의 *interval*들 중에 조건을 만족하는 하나를 찾는 알고리즘이므로 시간 복잡도는 $O(n)$이다. Adjust()는 Replace()와 마찬가지로 시간 복잡도는 $O(n)$이지만, 이 것은 재귀적으로 수용 제어 과정을 호출한다. 따라서 최악의 경우, 즉 Adjust()가 n번 호출되어 캐쉬-버퍼가 할당된 n개의 *interval*이 모두 교체된 경우에 $O(n^2)$가 된다. 하지만 그런 경우가 발생할 확률은 극도로 희박하고 일반적으로 최적 캐쉬 크기는 급변하지 않으므로, DIC의 오버헤드는 평균적인 경우에 n의 선형함수로 볼 수 있다. 설령 최악의 경우가 발생했다하더라도 $O(n^2)$는 디스크 I/O에 소비되는 시간을 감안하면 무시할 정도로 작다.

5. 실험 결과

5.1 시뮬레이션 모델

실험을 행한 시스템 모델과 디스크 특성은 각각 [표 2]과 [표 3]에서 보여진다. [표 2]에서 *skewed factor*는 CM 개체의 인기도를 나타내는 Zipf분포의 파라미터로서 0에서 1사이의 값을 가진다. 개체의 인기도가 균일할 때 1이고, 인기도 차이가 가장 현저할 때 0이다. 개체의 상영 길이는 10분과 50분 사이에 균일하게 분포한다. 요청이 도착하는 시간 간격(*Inter-Arrival Time*)은 파라미터가 0.5 초인 지수분포를 따른다. [표 3]는 실험의 디스크 모델로 사용한 IBM UltraStar 9zx[17]의 특성을 보여준다. 데이타(개체)는 [18]의 MPEG-1 실험 데이타를 사용했다.

표 2 시스템 모델

Number of Disks	10개
Number of Objects	20개
Memory	256 MB
Skewed Factor	0.2
Inter-Arrival Time	0.5 sec

표 3 디스크 특성: IBM UltraStar 9ZX

Max. Rotational Latency	5.99 ms
Min. Seek Time	0.7 ms
Max. Seek Time	17 ms
Transfer Rate	167 Mbps
Number of Cylinders	7400

5.2 실험 결과

[그림 8]은 시스템 초기부터 시간 흐름에 따른 캐쉬 크기의 변화 모습을 나타낸 것이다. 처음에는 메모리 자원이 풍부하므로, 캐쉬 크기가 전체 메모리 크기에 접근할 때까지 캐쉬-버퍼를 할당한다. 그 다음, 남는 메모리가 바닥이 나는 순간 캐쉬-버퍼가 반환됨에 따라 캐쉬 크기가 줄어들어 최적 캐쉬 크기에 접근하게 된다.

DIC 기법의 결과는 고정된 캐쉬 크기를 사용해서 보장형 서비스를 제공하는 캐싱 기법인 PSIC(Preemptive but Safe Interval Caching)과 비교했다. PSIC의 결과는 반복적 실험을 통해 얻은 최적 캐쉬 크기를 사용해서 얻은 결과이므로, DIC가 PSIC과 비슷한 성능을 보이면 DIC의 캐쉬 버퍼 할당 기법이 옳다고 할 수 있다. [그림 9]는 라운드 길이를 변화에 따른 DIC와 PSIC의 성능 차이를 나타낸 것이다. 여기서, 캐쉬 비율(cache ratio)은 캐쉬 크기를 전체 메모리 크기로 나눈 값이다. PSIC은 각 라운드 길이에 대해 캐쉬 비율를 0.05씩 변화시키면서 실험한 후, 최고의 성능을 그린 것이다. 그림에서 보듯이 DIC가 PSIC보다 항상 약간 좋은 성능을 나타내고, 두 기법 간 차이는 2%보다 작다. PSIC의 캐쉬 비율을 더 세밀하게 조정하면서 최적값을 찾는다면, 차이가 더 줄어들 것을 예상할 수 있다. 캐싱을 사용하지 않는 경우 라운드 길이를 1400일 때 최고의 성능을 보인다. 라운드 길이가 1600 msec를 넘게 되면 버퍼 병목이 발생하기 때문에 캐싱의 효과가 줄어드는 것을 볼 수 있다. 다시 말하면, 라운드 길이 1600부터는 캐싱을 사용하지 않아도 디스크 대역이 충분하기 때문에 캐싱의 의미가 없어진다.

DIC와 PSIC가 각각 할당하는 캐쉬 크기를 비교한 그림이 [그림 10]에 보여진다. 이 실험에서, DIC에 해당되는 값은 각 라운드에 할당된 캐쉬 크기들의 평균값을 나타내고, PSIC에 해당되는 값은 고정 최적 캐쉬 크기를 나타낸다. 두 기법간의 차이는 비교적 작기는 하지

만, 이런 차이는 근본적인 캐쉬 버퍼 할당법이 다르고, 위에서도 언급했듯이 PSIC의 캐쉬 비율의 실험 정밀도(resolution)가 0.05로 비교적 크기 때문이다. 예를 들어, PSIC의 최적 캐쉬 비율을 좀 더 정확히 구하면 라운드 길이가 1400msec일 때 0.38에 가깝다.

사용자의 요청 성향의 변화가 있는 환경에서 DIC와 PSIC의 성능을 비교한 결과가 [그림 11]이다. 여기서 PSIC의 캐쉬 비율은 Zipf 파라미터가 0.2일 때의 최적값(0.38)으로 설정했다. 그림은 시간(라운드)가 흐름에 따른 서비스 중인 스트림의 개수를 보여주고 있다. 변하는 환경을 만들기 위해 다음과 같은 4가지 실험용 데이타를 사용했다: (1) z=0.2 (round 5000~round 10000), (2) z=1 (round 10001~round 15000) (3) z=0.2, 인기도 순위는 (1)과 반대(round 15001~round 20000), (4) z=0, 인기도 순위는 (1)과 반대 (round 20001~ round 25000). 구간(1)에서는 PSIC 그 환경(z=0.2)에 최적화되어 있기 때문에, DIC와 매우 비슷한 성능을 보인다. 그러나, 구간(2), 구간(3)과 구간(4)에서는 캐쉬 크기가 최적값이 아니므로, 환경에 무관하게 최고의 성능을 내는 DIC와의 성능 차이가 현저하게 나타난다. [그림 12]에서, 위의 실험이 진행되는 동안 DIC기법에 의한 캐쉬 비율의 변화를 볼 수 있다.

실험을 통해, DIC는 시스템의 성능을 극대화시키기 위해 동적이면서 최적으로 캐쉬-버퍼를 할당한다는 것을 알 수 있다.

6. 결 론

지금까지 제안된 멀티미디어 서버의 캐싱 기법들은 오로지 적중률의 향상에만 초점을 맞추었기 때문에, 보장형 서비스를 제공하지 못했고, "떨림 없는" 서비스를 제공하는 PSIC 기법은 할당할 수 있는 캐쉬의 크기를 시스템 상황에 대한 사전 예측을 통해 최적값으로 고정시키므로써, 사용자의 요청 성향이 동적으로 변하는 상황에서는 시스템의 최고 성능을 보장하지 못하는 단점을 가지고 있다.

본 논문에서는, 보장형(deterministic) 서비스를 제공하면서, PSIC과 달리 캐쉬-버퍼를 동적으로 할당하는 새로운 캐싱 기법, DIC(Dynamic Interval Caching for Deterministic service)를 제안했다. DIC는 *interval* 단위 캐싱 기법으로서, 캐쉬 크기를 고정시키지 않고 사용자의 요청 성향의 변화에 따라 시스템의 성능을 극대화시키도록 조정한다.

어떤 시스템 환경에서도 DIC가 PSIC에 비해 항상 좋은 결과를 보였으며 시스템 환경이 초기에 예상한 것과 다르게 변하는 경우 두 기법 간의 성능 차이가 커진다는 것을 실험을 통해 입증하였다.

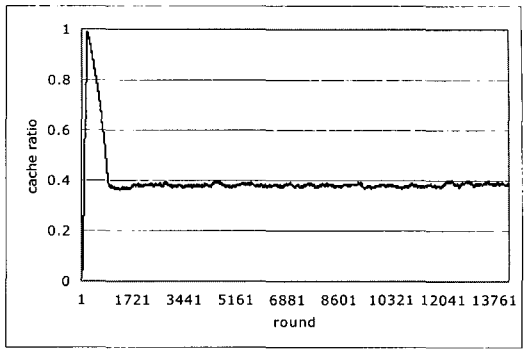

그림 8 DIC의 시간(라운드)에 따른 캐쉬 비율의 변화: 라운드 길이=1400msec, *cache ratio* = cache size / TOTAL

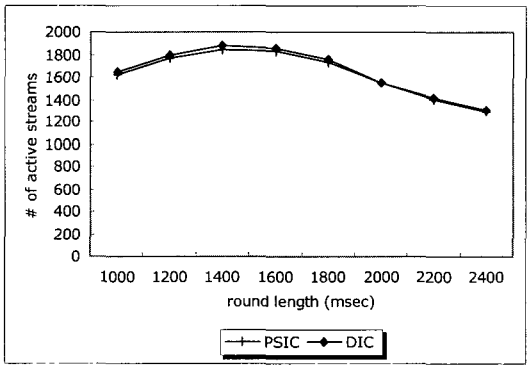

그림 9 라운드 길이에 따른 DIC와 PSIC의 성능 비교

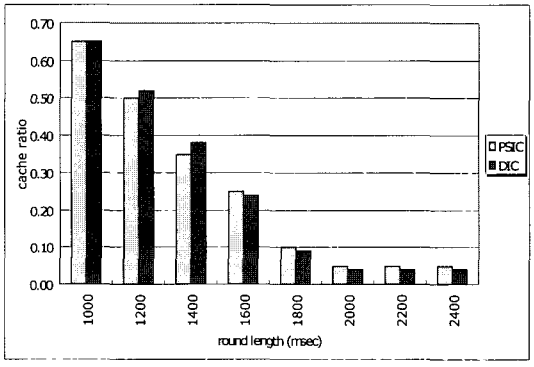

그림 10 DIC와 PSIC의 최적 캐쉬 비율 비교

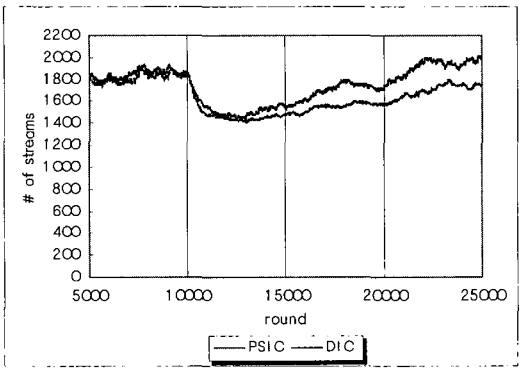

그림 11 사용자의 요청 성향이 변하는 환경에서, DIC와 PSIC의 성능 비교

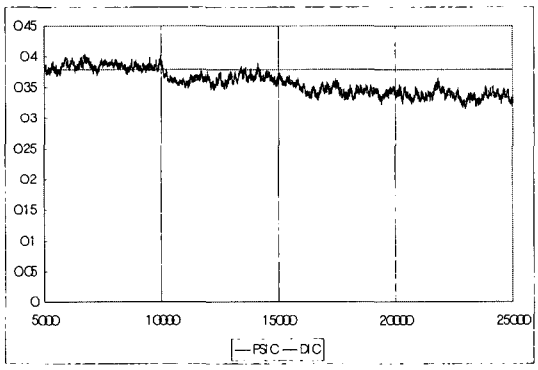

그림 12 DIC의 캐쉬 비율 변화

참 고 문 헌

[1] Scott A. Barnett and Gary J. Anido, Performability of Disk-array-based Video Servers, ACM Multimedia Systems Journal, 6(1):60-74, January 1998.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz and X. Ju, Staggered Striping in Multimedia Information Systems, In Proc. of ACMMOD, pages 79-90, Minnesota, USA, June 1994.

[3] Pei Cao and Sandy Irani, Cost-aware WWW Proxy Caching Algorithms, In Proceedings of the USENIX Symposium on Internet Technologies and Systems, pages 193-206, Monterey, California, December 1997

[4] E. Chang and A. Zakhor, Admission Control and Data Placement for VBR Video Servers, In Proc. of IEEE Int'l Conference

[5] E. Chang and A. Zakhor, Cost Analysis for VBR Video Servers, In IS&T/SPIE Int'l Symposium on Electronic Imaging: Science and Technology, pages 29-31, California, January 1996.

[6] Asit Dan, Rajat Mukherejee Daniel M. Dias, Dinakr Sitaram and Renu Tewari, Buffering and Caching in Large-scale Video Servers, In Compcon-Technologies for the Information Superhighway, pages 271-224, Los Alamitos, CA, January 1995;

[7] J. Dangler, E. Biersack and C. Berhart, Deterministic Admission Control Strategies in Video Servers with Variable Bit Rate, In Proc. of International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, pages 164-171, Hiroshima, Japan, June 1996.

[8] Harvey M. Deitel, An Introduction to Operating Systems, Addison Wesley, 1983.

[9] Sreenivas Gollapudi and Aidong Zhang, Buffer Management in Multimedia Database Systems, In Proc. of IEEE International Conference on Multimedia Computing and Systems, pages 186-190, Hiroshima, Japan, June 1996.

[10] KyungOh Lee and Heon Y. Yeom, Deciding Round Length and Striping Unit Size for Multimdeia Servers, In Proceedings of the 4th International Workshop on Multimedia information Systems (MIS'98), pages 33-44, Istanbul, Turkey, September 1998

[11] T. Raymond Ng and Jinhai Yang, An Analysis of Buffer Sharing and Prefetching Techniques for Multimedia Systems, ACM Multimedia Systems, 4(2):55-69, April 1996.

[12] B. Ozden, R. Rastogi and A. Silberschatz, Buffer Replacement Algorithms for Multimedia Storage Systems, in Proc. of IEEE International Conference on Multimedia Coputing and Systems, pages 172-180, Hirosima, Japan, June, 1996.

[13] B. Ozden, R. Rastogi, and A. Silberschatz, Disk Striping in Video Servers Environments, In Proc. of IEEE International Conference on Multimedia Computing and Systems, pages 580-589, Hiroshima, Japan, June 1996.

[14] KyungOh Lee, Jin B. Kwon and Heon Y. Yeom, Exploiting Caching for Realtime Multimedia Systems, In Proc. of IEEE International Conference of Multimedia Computing and Systems, Florence, Italy, June 1999

[15] David A. Patterson and John L. Hennessy, Computer Architecture - a Quantitative Approach, Morgan Kaufmann Publisher, 1996

[16] C. Ruemmler and J. Wilkes, An Introduction to Disk Drive Modeling, IEEE Computer, 27(3):17-28, March 1994

[17] Ultra 9zx Information, http://www.storage.ibm.com/hardsoft/diskdrdl/prod/ultra9zx.htm.

[18] Infomatik MPEG-I traces, ftp://ftp-inof3.informatik.uni-wuerzburg.de/pub/MPEG.

권 진 백

1998년 한국외국어대학교 통계학과 졸업. 1998년 ~ 1999년 서울대학교 전산과학과 석사. 2000년 ~ 현재 서울대학교 컴퓨터공학부 박사과정 재학중. 관심분야는 분산시스템, 멀티미디어

염 헌 영

정보과학회논문지 : 시스템 및 이론
제 27 권 제 2 호 참조

이 경 오

1989년 서울대학교 계산통계학과 졸업. 1989년 ~ 1995년 세일중공업 근무. 1991년 ~ 1994년 서울대학교 전산과학과 석사. 1995년 ~ 1998년 서울대학교 박사. 1999년 ~ 현재 선문대학교 컴퓨터정보학부 교수. 관심분야는 데이타베이스, 그룹웨어, 멀티미디어