비디오를 위한 효율적인 프록시 서버 캐쉬의 관리*

조경산**, 홍병천**

Efficient Management of Proxy Server Cache for Video

Kyungsan Cho, ByungChun Hong

Abstract

Because of explosive growth in demand for web-based multimedia applications, proper proxy caching for large multimedia object (especially video) has become needed. For a video object which is much larger in size and has different access characteristics than the traditional web object such as image and text, caching the whole video file as a single web object is not efficient for the proxy cache. In this paper, we propose a proxy caching strategy with the constant-sized segment for video file and an improved proxy cache replacement policy. Through the event-driven simulation under various conditions, we show that our proposal is more efficient than the variable-sized segment strategy which has been proven to have higher hit ratio than other traditional proxy cache strategies.

Key Words: Video Data, Proxy Caching, Replacement Policy

* 이 연구는 2003학년도 단국대학교 대학 연구비의 지원으로 연구되었음.

** 단국대학교 정보컴퓨터학부

1. 서론

웹(world wide web)상의 정보는 매월 15%이상 증가하고 인터넷에 연결되는 호스트 컴퓨터의 수는 매 9개월에 두 배 이상 증가하고 있으며, 웹에서 제공되는 서비스도 단순 텍스트 및 이미지에서 오디오 및 비디오와 같은 다양한 미디어 정보로 변하고 있다. 이러한 변화로 인한 웹의 기하급수적인 성장과 인기 있는 웹사이트에 대한 수많은 사용자들의 동시 접근 요구는 인터넷 사용자에게 접근 지연과 네트워크 과부하의 문제를 발생시켰다[4]. 이러한 문제를 해결하기 위하여 웹 접근 지연시간을 감소하고 서버의 과부하와 네트워크 혼잡을 감소시키기 위한 방법으로서 웹 캐쉬가 제안되었다[3]. 클라이언트와 서버사이의 경로에 위치한 프록시 서버는 웹서버 대신 클라이언트에게 요청된 객체를 신속하게 제공하여 제기된 문제에 대해 높은 성능향상을 제공할 수 있으므로, 프록시 서버의 멀티미디어 데이터 캐싱이 보다 중요해졌다. 그러나, 이미지나 텍스트와 같은 전통적인 웹 데이터에 대해 기존의 웹 캐쉬가 높은 성능향상을 제공했음에도 불구하고, 전통적인 데이터에 비해 용량이 크고 참조 접근 특성이 다른 비디오 같은 멀티미디어 데이터를 기존의 웹 캐쉬처럼 파일 단위로 캐싱하는 것은 프록시 캐쉬의 가용성을 낭비하게된다. 이에 따라, 프록시 캐쉬의 효율성을 높이려는 여러 연구가 있었다. 특히 비디오를 재생할 때, 프록시 서버에 비디오 데이터의 충분한 양이 저장되어 있지 않으면 초기 지연 문제가 발생하므로 이를 해결하기 위한 연구가 있었다[7,8,10]. 또한, 프록시 캐쉬의 효율적인 관리를 위해 가변크기의 세그먼트 기반 캐쉬 구조가 제안되었으며, 이 기법은 기존의 다른 기법에 비해 높은 히트율을 갖는 것으로 분석되었다[7]. 각 사용자에게 적절한 대역폭으로 지연이나 지터(jitter)없이 적절하게 멀티미디어 데이터를 전송하기 위한 자료 전송 제어의 연구도 제시되었다[1,6]. 하지만, 기존 연구들은 대용량 비디오 데이터의 캐싱에 효율성과 관리의 간단성을 함께 제공하지는 못하였다.

본 연구에서는 비디오 데이터를 위한 고정 크기 세그먼트를 갖는 효율적인 프록시 캐쉬 구조와 참조수, 최근성 및 저장 용량을 반영한 캐쉬 제거 기법을 제안하고, 제안된 구조와 기법에 의한 성능 향상을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 웹 캐쉬와 프록시 캐쉬 및 비디오 데이터의 특성을 설명하고, 3장에서 비디오 데이터의 캐싱에 효율적인 프록시 캐쉬의 구조 및 관리 기법을 제안한다. 4장에서 다양한 환경에서의 시뮬레이션을 통해 제안 기법에 의한 성능 개선도를 제시하고, 5장의 결론으로 마무리한다.

2. 관련 연구

웹의 기하 급수적 성장은 웹 서버의 부하와 인터넷의 통신량을 급속히 증가 시켰고, 이로 인해 웹 페이지로의 접근 지연 시간도 증가하였다. 이러한 접근 지연 시간을 감소시키고 서버의 부하와 네트워크 혼잡을 줄이기 위한 웹 캐쉬 기법이 연구되었다.

2.1 웹 캐쉬

웹은 인터넷으로 연결된 대단위 분산 정보 시스템으로 서버, 프록시, 클라이언트 등으로 구성된다. 클라이언트가 웹서버에게 정보를 요청하면 요청된 페이지의 파일이 네트워크를 통해 제공된다. 웹의 성능 향상을 위한 웹 캐쉬는 정보를 웹 서버와 클라이언트 사이에 배치하여 사용자의 참조 요청을 신속히 제공함으로서 서비스의 지연을 줄일 수 있는데, 캐쉬의 위치에 따라 <표 1>처럼 클라이언트 캐쉬, 프록시 캐쉬 그리고 서버 캐쉬로 나눌 수 있다[3].

2.2 웹 캐쉬 제거 정책

웹에서는 전통적 메모리 시스템의 캐쉬와는 달리 파일 단위로 캐쉬에 저장한다. 또한, 웹 참조는 대부분이 읽기(read)이며, 참조의 인기도에

따라 파일의 참조수가 정해지는 특성을 가지므로, 기존 메모리 시스템에서 널리 사용되었던

<표 1> 웹에서 캐쉬의 사용 환경 및 목적

캐쉬위치	목적	효과	캐쉬 매체	문제점
클라이언트	응답시간 단축, 네트워크 부하줄감	네트워크 부하줄감, 응답시간 감소	디스크	문서 동기화
네트워크	네트워크 부하줄감	네트워크 부하줄감, 빠른 서비스	디스크	문서 동기화
서버	서버 성능 향상	응답시간 감소	메모리	하드웨어 비용증대

LRU 제거 정책은 적절하지 않다고 분석되었다. 웹 참조의 참조특성 분석 결과를 활용하기 위해 참조 횟수, 저장 크기 및 캐쉬에 저장된 시간 등의 특성 인수들을 활용하는 제거 방식인 LFU, SIZE, FIFO 등의 기법들이 연구되었다. 하지만, 제거 정책에 한가지 특성만을 적용하여서는 웹 캐쉬의 성능 향상에 한계가 있으므로, 여러 특성 인수들을 이용하는 혼합 인수기법인 SLRU 및 WLFU 등이 제시되었다. 혼합 인수기법들은 캐싱의 비용에 대한 성능적 비율($\frac{\text{이익}}{\text{비용}}$)을 수식화하고, 이 값이 가장 작은(캐싱 비용에 비해 이익이 가장 작은) 객체를 캐쉬에서 제거하는 것이다. 이 때 비용은 캐쉬에 저장되는 객체의 크기(메모리 비용)를 나타낸다. 예를 들어, SLRU 기법에서는 다음과 같은 식을 제시하였다.

$$\frac{\text{이익}}{\text{비용}} = \frac{1}{T - T'} \times \frac{1}{SIZE}$$

T:현재 시간, T':최종 참조시간, SIZE:파일크기

또한, WLFU 정책은 $\frac{\text{이익}}{\text{비용}} = \frac{\text{참조수}}{\text{파일크기}}$ 으로 제시하였다. 이러한 혼합 인수기법들은 파일 단위로 저장되는 웹 서버 캐쉬에 매우 효율적인 것으로 분석되었다[3].

2.3 비디오 데이터의 특성과 프록시 캐싱

텍스트나 이미지와 같은 전통적인 웹 데이터에 대해 기존의 웹 캐쉬가 높은 성능향상을 제공하였지만, 대용량 멀티미디어 데이터의 캐싱에는 다음과 같은 특성 때문에 효율적으로 이용되지 못했다[8]. 첫째, 비디오 같은 멀티미디어 데이터들은 전통적인 웹 데이터들보다 매우 큰 용량을 가지므로 파일 단위의 웹 캐쉬 관리 기법은 부적절하다. 둘째, 이미지나 텍스트 같은 전통적인 웹 접근 패턴과 멀티미디어 데이터에 접근하는 패턴들은 다른 특성을 가진다. 셋째, 전통적인 웹 데이터와 멀티미디어 데이터의 용량 차이와 전송 방식 그리고 기존 프로토콜의 부적절성 때문에 전통적인 웹 캐싱 기법은 적절하지 못하다. [2]의 분석에 의하면 멀티미디어 데이터의 생명주기는 생성된 시간에 반비례하며, 참조횟수에 비례하는 특성을 가진다. 이러한 특성들에 의하여 총 참조수, 일정 시간동안의 참조수, 재생량, 최근성 등이 고려되는 비디오 데이터의 인기도에 따라 캐싱의 저장량 및 제거 등을 차별하는 방법들이 제안되었고, 특히 대용량의 멀티미디어 데이터를 위한 웹 서비스의 성능 향상을 위해서 멀티미디어 데이터를 위한 프록시 캐싱 기법이 제시되었다[2,7,8,10]. [10]은 프록시 캐쉬에 멀티미디어 데이터 파일 전부를 저장하는 대신에 멀티미디어 데이터의 앞부분만을 별도의 캐쉬에 저장하는 prefix 기법을 제안하였다. [7]에서는 대부분의 비디오 데이터들은 부분적으로 캐싱된다는 분석에 근거하여 인기도에 따라 캐싱되는 데이터의 양을 가변 크기의 세그먼트로 저장하고 비디오 데이터를 관리하는 캐쉬 기법을 제안하였다. 가변 크기 세그먼트 기법에서는 각 세그먼트마다 다음과 같이 표현되는 캐싱 임계값을 계산하여 저장과 제거를 결정하였다.

$$\text{캐싱 임계값} = \frac{1}{T - T'} \times \frac{1}{I}$$

T:현재 시간, T':마지막 참조시간, I:세그먼트의 거리

[7]에서는 비디오 데이터에 대해 가변 크기 세

그먼트 기법이 기존의 다른 프록시 캐쉬 기법보다 우수한 것으로 분석되었다.

3. 프록시 서버의 캐싱 기법 제안

본 연구에서는 비디오 파일을 대상으로 캐쉬의 히트율을 높이고 캐쉬 관리를 간단히 할 수 있는 고정 크기 세그먼트 기법을 제안한다.

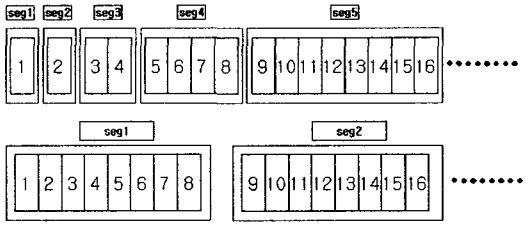
3.1 고정 크기 세그먼트 프록시 캐쉬 구조

비디오 데이터의 사용자 접근 패턴은 인기도가 높은 파일을 자주 참조하는 특성을 보이며, 일부 사용자는 모든 데이터를 재생하기 이전에 정지 버튼을 선택하는 특성이 있다고 분석되었다[9]. 이러한 사용자 접근 패턴을 반영하여, 비디오 파일의 인기도가 증가할수록 해당 파일의 캐싱량을 증가시키고 인기도가 감소하면 줄이도록 하여 캐쉬의 활용도를 높일 수 있다. 프록시 서버의 캐쉬를 세그먼트 단위로 관리할 경우 세그먼트의 크기는 클라이언트와 프록시 사이의 데이터 전송 및 캐쉬 관리의 효율성에 따라 고정 크기 또는 가변 크기로 정할 수 있다.

세그먼트의 크기를 두 배씩 증가시키는 가변 크기 세그먼트 관리 기법[7]에서는 저장된 데이터에 대한 상태 저장과 연산을 각 세그먼트 단위로 수행하므로 프록시 서버의 캐쉬 관리에 과부하를 증가시킬 수 있다. 본 연구에서는 가변 크기 세그먼트 기법의 캐쉬 관리에 따른 과부하를 줄이기 위해 웹서버 캐싱에서 사용한 파일 단위 캐싱 기법을 도입하되, 비디오 파일의 특성과 측정된 대역폭을 활용할 수 있도록 고정 크기 세그먼트를 갖는 프록시 캐쉬 구조를 제안한다.

고정 크기 세그먼트 기법에서는 프록시와 클라이언트 사이의 가변적인 전송 대역폭의 측정이 가능하도록 세그먼트의 크기를 정할 수 있으며, 별도의 추가 장치 없이 멀티미디어 전송을 위한 제어 프로토콜인 RTCP를 이용하여 대역폭을 측정할 수 있다[1]. 이때, 세그먼트의 구성은 측정된 대역폭에 적절하게 고정된 개수의 블록으로 그룹화 할 수 있다.

<그림 1>은 [7]에서 제안된 가변 크기 세그먼트의 구성과 본 연구에서 사용한 고정 크기 세그먼트의 구성을 나타낸다.



<그림 1> 가변 크기 및 고정 크기 세그먼트

세그먼트를 구성하는 각 블록은 [7]에서와 같이 재생 시간을 1.8초로 정하고, RTCP를 4.8초마다 전송할 수 있다면, 8개의 블록을 하나의 세그먼트(이 경우에는 각 세그먼트 재생 시간은 14.4초)로 정할 수 있다. RTCP를 이용하면 프록시 서버와 클라이언트 사이에서 재생 시간이 14.4초인 세그먼트의 전송마다 3번의 RTCP를 이용한 대역폭 측정이 가능하다. 따라서, 측정된 대역폭에 맞게 세그먼트의 전체 또는 선별된 블록만 전송할 수 있다. 대역폭 측정 및 세그먼트 전송 제어의 상세 사항은 본 논문의 연구 범위를 넘으므로 더 이상 논의하지 않으며 별도의 과제로 연구 중이다. 본 연구에서는 위와 같이 RTCP의 전송 주기 시간의 3배정도 크기로 고정 세그먼트를 설정한다.

비디오 데이터는 초기 지연과 지터에 의한 영향 때문에 일정 부분이 버퍼링 되어야만 사용자에게 제공되기 시작한다. 따라서, 본 연구에서 제안하는 프록시 캐쉬는 최근의 프록시 서버 구조의 추세에 따라 비디오 데이터를 저장하기 위해 두 부분으로 구성하였다. 첫째 부분은 멀티미디어 데이터의 초기 세그먼트만을 저장하기 위한 캐쉬(초기 세그먼트 캐쉬)이고, 둘째 부분은 나머지 세그먼트 부분을 저장하기 위한 캐쉬(나중 세그먼트 캐쉬)이다. 초기 세그먼트 캐쉬 공간은 시뮬레이션을 통해 전체 캐쉬 용량의 10%가 가장 효율적인 것으로 분석되었다.

3.2 개선된 캐쉬 제거 기법의 제안

본 연구에서는 비디오 데이터의 참조 특성에 대한 분석 결과를 활용하여 다음과 같이 해당 세그먼트를 프록시에 캐싱하는 이익과 캐싱에 따른 비용의 함수를 캐싱 임계값으로 정하고, 그 값에 따라 캐쉬에 저장 또는 제거를 정하도록 하였다.
첫째, 해당 파일이 참조된 횟수가 많을수록 다시 요청될 확률이 높다는 파일의 인기도를 고려하여 그 파일의 참조 횟수(n)를 캐싱 이익으로 사용하며, 참조 횟수가 인기도에 미치는 큰 영향력을 고려하고 시뮬레이션 분석으로 검증된 n^2으로 반영한다.
둘째, 캐쉬에 생성된(처음 저장된) 시점이 최근인 파일일수록 새로 요청될 확률이 높고, 오래된 파일일수록 새로 요청될 확률이 낮다는 최근성의 특성을 고려하여 세그먼트의 캐싱 나이(현재 시간(T) - 생성 시간(S))의 역수를 캐싱 이익으로 반영한다.
셋째, 해당 파일의 캐쉬에 현재 저장된 크기(SIZE)는 그 파일을 캐쉬에 유지하는 비용이 되므로, 그 크기를 캐싱 비용으로 반영한다.

위의 세 항목으로부터, 캐싱으로 인한 $\frac{\text{이익}}{\text{비용}}$을 다음과 같이 캐싱 임계값으로 설정하였다.

$$\frac{n^2}{T-S} \times \frac{1}{SIZE}$$

T:현재 시간, S:생성 시간, n:참조 횟수,
SIZE:현재 저장 크기

위 식에서 각 인수는 세그먼트마다 별도의 연산을 필요로 하지 않으며, 각 파일 단위로 연산을 한다. 따라서, 각 파일에 대해 마지막으로 저장된 세그먼트와 캐싱 임계값이 저장되어 세그먼트의 저장 및 제거에 사용된다. <그림 2>는 앞에서 제시된 캐싱 임계값을 활용하여 세그먼트의 저장 또는 제거를 정하기 위해 본 연구에서 사용하는 캐싱 알고리즘이다.

```
if (i < Kmin) {//초기 세그먼트를 위한 캐쉬
 if(i가 포함된 파일이 초기 세그먼트에 저장 되어있지 않으면?){
   제거할 파일의 초기 세그먼트를 선택하여 제거;
   세그먼트 i 가 포함된 새로운 파일의 초기 세그먼트를 저장;
 }else{
   exit;
 }
}else{//나머지 세그먼트를 위한 캐쉬
    if(파일 P의 세그먼트 i를 저장 공간이 있다면){
       파일P의 세그먼트 i를 저장;
    }else{
          캐싱 임계값이 가장 작은 파일 J선택;
          if(J의 캐싱 임계 값 < i의 캐싱 임계 값){
             if(J를 디스플레이 하는 사용자가 없다면)
                파일 J에서 가장 마지막 세그먼트 삭제;
                파일P의 세그먼트 i를 저장; }
    }
}
/* Kmin : 초기 세그먼트크기,
 캐싱 임계 값 : (n² /(T-S)) * (1 / SIZE) */
```

<그림 2> 비디오 파일P의 세그먼트 i를 위한 캐싱 알고리즘

4. 시뮬레이션을 통한 성능분석

본 장에서는 프록시 캐쉬의 크기, 비디오 인기도, 파일의 크기, 인기도 변화, 사용자 접근 패턴 등을 다양하게 변경하면서 비디오 요청에 대한 프록시 서버의 시뮬레이션을 통해 제안 기법의 성능 향상을 분석하였다. 본 연구에서 제안한 기법이 가변적 크기 세그먼트 기법[7]보다 높은 바이트 히트율을 보이는 분석 결과를 4.2절에서 제시한다.

4.1 시뮬레이션 환경

본 연구에서는 [7]에서 기존의 프록시 서버들에 대한 성능 비교 연구를 위해 사용한 것과 동일한 환경에서 비디오 파일 요청과 프록시 서버의 처리 과정을 위해 C언어로 구현된 사건 기반 시뮬레이션을 수행하였으며, <표 2>가 본 연구의 시뮬레이션에서 사용한 환경 인수와 그의 기

본 값이다. 비디오 파일의 평균 크기는 2,000블록, 한 블록의 재생 시간은 1.8초, 비디오의 요청 도착 간격은 평균 60초, 비디오 파일의 수는 2,000개로 가정하였다. 시뮬레이션에서 사용한 비디오의 인기도는 기존 연구와 같이 두 개의 변화 인수 x, M을 갖고, Zipf-like 분포법칙을 따르는 다음 요청 확률식을 이용하였다.

$$P_i = \frac{c}{i^{1-x}}, \ \forall i \in \{1,.....,M\},$$

$$c = 1 \Big/ \sum_{i=1}^{M} \frac{1}{i^{1-x}}$$

P_i: i의 요청확률, i: 인기도, c: 상수, M과 x는 변화되는 인수.

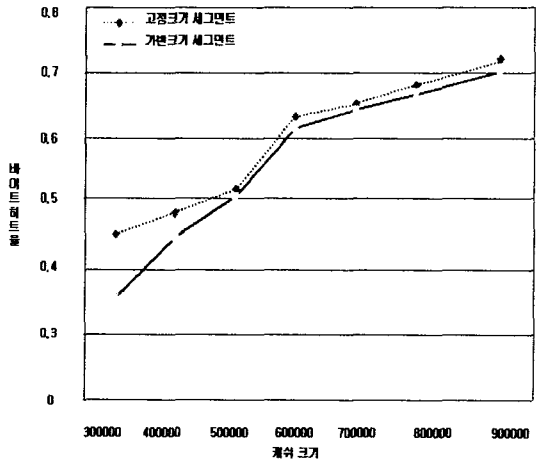
<표 2> 시뮬레이션에서 사용된 인수

기호	정의	기본값
B	비디오 파일의 평균 블록 수	2,000블록
t	평균 요청 간격	60초
b	블록의 재생시간	1.8초
C	캐쉬 용량 (기본값)	300,000블록
Cinit	초기 세그먼트 캐쉬의 비율	10%
Kmin	초기 세그먼트의 수	32개 블록
M	비디오 파일의 수	2,000
Zipf(x,M)	비디오 파일의 Zipf의 인수	x= 0.2, M= 2000
k	인기 파일의 최대 이동간격	5
R	인기 파일의 이동 사이의 요청 숫자	2,000

[7]에서 가변 크기 세그먼트기법과 기존 프록시 캐쉬 기법의 성능 특성을 비교분석 하였는데, 가변 크기 세그먼트기법이 항상 우수한 것으로 제시되었다. 따라서, 본 연구에서는 가변 크기 세그먼트 기법과 본 연구에서 제안한 기법의 성능 특성을 비교 분석하도록 한다. 다음 소절에서는 다양한 인수를 변화하면서 본 연구의 제안에 의한 성능 개선 도를 제시한다.

4.2 성능 분석 결과

캐쉬의 크기를 300,000블록에서 900,000블록까지 변경시키면서 프록시 캐쉬의 크기가 바이트 히트율에 미치는 영향을 분석한 결과, <그림 3>과 같이 본 연구에서 제안한 기법(고정크기 세그먼트로 표시됨) 기법이 가변크기 세그먼트 기법에 비해 향상된 히트율을 보여준다.



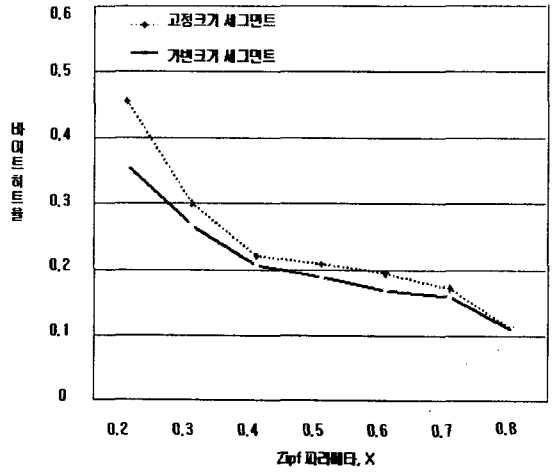
<그림 3> 캐쉬 크기에 따른 히트율 비교

특히 프록시 캐쉬의 크기가 작을 때 더 효율적인 성능을 보였는데, 가변 크기 세그먼트 기법은 한 파일에 대해 캐쉬에 저장되는 세그먼트의 양이 급속히 증가하므로 캐쉬 크기가 작은 경우 캐쉬의 공간을 효율적으로 이용하지 못하는 것으로 분석된다.

Zipf-like 식의 인수 x(비디오의 인기도) 값을 0.2부터 0.8까지 변화시키면서 히트율에 미치는 영향을 분석한 결과는 <그림 4>와 같다.

인기도의 편중도를 나타내는 x값이 작을수록 실제 비디오 요청의 특성을 나타내는 인기도의 영향을 더 많이 반영하는 환경이므로, x값이 작을 때 본 연구의 제안 기법이 인기도의 반영에 더 충실함을 보인다. 그러나 x값이 커질수록 모든

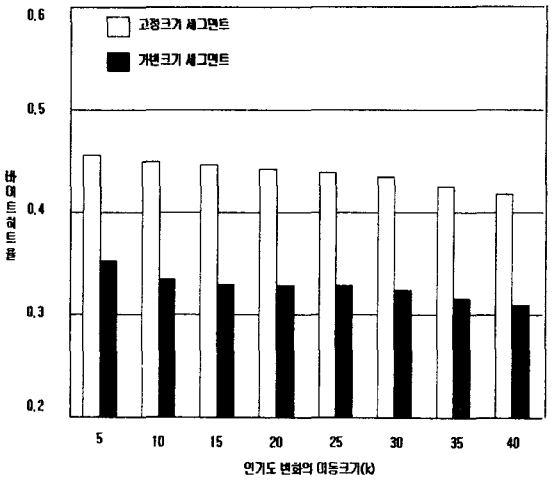
파일의 인기도가 유사하게 되어, 각 기법의 특성 차이가 줄어든다.



<그림 4> Zipf 인수 x값에 따른 히트율 비교

비디오의 인기도는 시간이 흐름에 따라서 변화된다. 한 순간에 가장인기 있던 비디오도 시간이 흐른 후에는 인기도가 변화한다. 본 연구에서는 인기도의 변화를 반영하기 위해 R번의 비디오 요청 후에 비디오의 인기 순위가 변경 되도록 하였다. 변경 이전과 이후의 인기도인 Zipf-like 분포 사이의 연관성은 1과 M 사이의 값을 갖는 k (파일의 인기도 변경 크기)에 의해서 결정된다. 즉, 인기도 변경 이전에 가장 인기 있던 비디오는 인기도 변경 이후에는 r_1(1에서 k사이에 임의로 선택된 값)번째의 인기도를 갖도록 인기도 분포가 변경된다. 그리고 두 번째로 인기 있었던 비디오는 r_2(r_1을 제외한 1에서 min(M, k+1)사이의 임의 값)번째의 인기도로 변화된다. 동일한 과정의 인기도 변경을 모든 비디오에 대해 반복 수행된다.

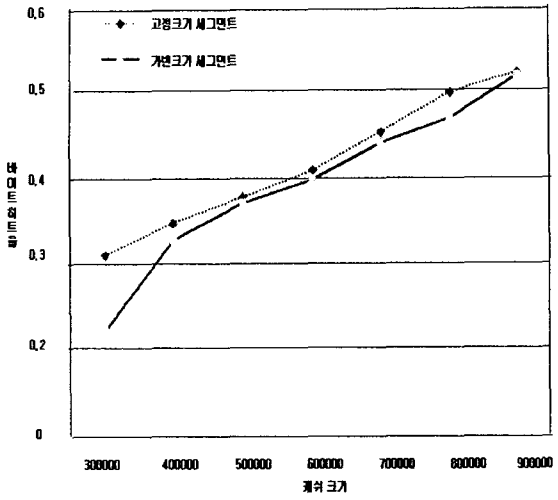
<그림 5>는 인기도 변화의 이동 크기 k값에 따른 바이트 히트율을 보여준다.



<그림 5> 비디오 인기도의 변화량에 따른 히트율의 변화

인기도 변화가 클수록 캐쉬에 저장된 내용의 효율성이 떨어져 히트율이 감소한다. 제안 기법에서는 비디오가 캐쉬에 최초로 참조된 시간을 이용한 인기도의 최근성을 반영한 결과 인기도의 변화에 대해 좋은 성능을 보였다.

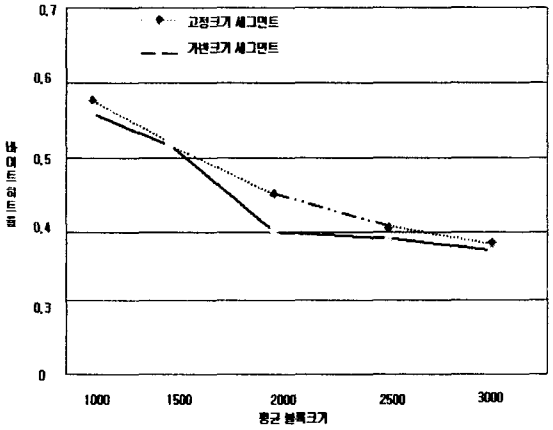
[9]의 분석에 의하면 61%의 사용자는 비디오 파일 전체를 디스플레이 하지만 39%의 사용자는 일찍 정지 버튼을 선택하여 비디오 데이터의 수신을 중지한다. 본 연구에서는 이를 반영하기 위하여, 61%의 사용자는 요청한 비디오 데이터 전체를 수신하며, 30%의 사용자는 초기 세그먼트 부분 만을 수신하고, 나머지 9%의 사용자는 전체 데이터의 50%를 수신한다고 가정한 환경에서 사용자 접근 패턴이 바이트 히트 율에 미치는 영향을 시뮬레이션한 결과는 <그림 6>과 같다. 39%의 파일은 일부만 수신된다는 사용자 접근 패턴을 고려한 분석인 <그림 6>은 이를 고려하지 않은 <그림 3>에 비해 전반적인 히트율은 감소한다. 이는 저장된 내용의 일부만 수신하여 캐쉬의 효율성이 떨어지기 때문이다.



<그림 6>사용자 접근 패턴을 고려한 분석

하지만, 제안 기법은 사용자의 접근 패턴을 고려하여도, 기존 방법에 비해 향상된 히트율을 보여준다.

비디오 파일의 평균 크기가 바이트 히트율에 미치는 영향을 분석한 결과는 <그림 7>과 같다. 비디오 파일의 평균 크기가 증가할수록 캐싱되는 비디오 데이터의 수가 감소하게되므로 바이트 히트율이 감소한다.



<그림 7> 비디오의 평균 크기에 따른 분석

5. 결론

본 논문에서는 대용량 비디오 데이터를 위한 고정 크기 세그먼트를 갖는 프록시 캐싱 기법을 제안하고 이에 따른 성능 향상 정도를 분석하였다. 제안된 프록시 캐쉬의 특성은 다음과 같다. 첫째, 초기 지연을 감소하기 위한 최근의 프록시 캐쉬의 추세에 따라 제안 프록시 캐쉬도 초기 세그먼트 캐쉬와 나중 세그먼트 캐쉬로 구성하였다. 둘째, 프록시 캐쉬는 비디오 데이터의 대용량, 참조의 인기도와 부분적으로 참조되는 접근 특성을 고려하고 클라이언트의 대역폭 측정과 이에 따른 전송 제어와 관리의 효율성을 위해 고정된 세그먼트 단위로 관리한다. 셋째, 비디오 파일의 참조 특성을 고려하여 참조수, 최근성 및 저장 용량 등을 고려한 캐싱이익/캐싱비용의 함수를 캐싱 임계값으로 제안하고 이에 따라 저장과 제거를 수행하였다.

다양한 인수의 변화에 따른 프록시 캐싱의 시뮬레이션을 통한 분석에서 제안된 기법은 기존 연구에 의해 프록시 캐쉬의 가장 효율적인 관리 기법으로 알려진 가변 크기 세그먼트 기법보다 히트율이 높고 관리 및 연산이 간단해 짐을 제시하였다. 하지만, 본 연구는 기존 연구와의 비교를 위해 한정된 특성을 갖는 비디오 작업부하에 대한 분석으로 제한되었다.

본 연구에서 제안한 고정 크기 세그먼트 기법의 효율적인 운영을 위해서는 다양한 작업 부하의 생성, 선인출 기법과 비디오 재생의 QoS를 높이기 위한 세그먼트 크기 설정 및 대역폭 측정에 의한 전송 제어 등이 함께 고려되어야하며, 이들은 본 논문의 향후 과제로 연구가 진행중이다.

참고문헌

[1] 고동환, 나승구, 안종석, "화상회의 시스템에서 RTCP를 이용한 네트워크 대역폭 예측," 「정보과학회 가을 학술 발표 논문집」, Vol.24, No.2, pp. 283-286, 1997.

[2] 임은지, 최태욱, 박성호, 정기동, "인터넷 상의 연속 미디어를 위한 Proxy Caching 기법," 「정보과학회 2000년 춘계 학술대회 논문집」, Vol.27, No.1, pp.382-384, 2000.

[3] 안효범, 조경산, "웹 서버의 참조 특성 분석과 성능 개선," 「한국정보처리학회 논문지」, 제8-A권, 제3호 pp. 201-208, 2001.

[4] 조경산, 「컴퓨터 네트워크와 인터넷」 개정3판, 도서출판 그린, 2002.

[5] Jaeyeon Jung, Dongman Lee and Kil-nam Chon, "Proactive Web Caching with Cumulative Prefetching for Large Multimedia Data," Computer Networks, Vol.33 pp. 645-655, 2000.

[6] J. Rexford, S. Gruber and Andrea Basso, "Protocol Considerations for a Prefix Caching Proxy for Multimedia Streams," Computer Networks, Vol. 33, No. 6, pp. 657-668, 2000.

[7] Kun-Lung Wu, Philip S.Yu and Joel L. Wolf, "Segment-Based Proxy Caching of Multimedia Streams," Proc. of World Wide Web, pp. 36-44, 2001.

[8] Reza Rejaie, Mark Handley, Haobo Yu and Deborah Estrin, "Proxy Caching Mechanism for Multimedia PlayBack Streams in the Internet," Proc. of International Web Caching Workshop, 1999.

[9] Soam Acharya and Brian Smith, "MiddleMan: A Video Caching Proxy Server," Proc. of Workshop on Net-work and Operating System Support for Digital Audio and Video, 2000.

[10] S. Sen, J. Rexford and D. Towsely, "Proxy Prefix Caching For Multimedia Streams," IEEE INFOCOM, pp. 1310-1319, 1999.

저자소개

조경산

1979 서울대학교 전자공학과 학사

1981 한국과학원 전기전자공학과 석사

1988 Univ. of Texas at Austin 전기전산공학과 박사

1988~1990 삼성전자 컴퓨터부문 책임연구원

1990~현재 단국대학교 정보컴퓨터학부 교수

관심분야: 컴퓨터 시스템 설계 및 분석, 네트워크 시스템 설계 및 분석, 웹 응용, 시뮬레이션

홍병천

2001 삼척대학교 컴퓨터공학과 학사

2003 단국대학교 정보컴퓨터학부 석사

관심분야: 멀티미디어, 이동통신, 시뮬레이션