산업기술연구(강원대학교 산업기술연구소 논문집), 제27권 B호, 2007.
Journal of Industrial Technology, Kangwon Natl. Univ., Korea, No. 27 B, 2007.

트랜스코딩 프록시를 위한 메타데이터 추가 캐싱

Meta-tailed Caching for Transcoding Proxies

강 재 웅*　　최 창 열**
Kang, Jai-Woong　　Choi, Chang-Yeol

Abstract

Transcoding video proxy is necessary to support various bandwidth requirements for mobile multimedia and to provide adapting video streams to mobile clients. Caching algorithms for proxy are to reduce the network traffic between the content servers and the proxy. This paper proposes a Meta-tailed caching for transcoding proxy that is efficient to lower network load and CPU load. Caching of two different data types - transcoded video, and metadata - provides a foundation to achieve superior balance between network resource and computation resource at transcoding proxies. Experimental results show that the Meta-tailed caching lowers at least 10% of CPU-load and at least 9% of network-load at a transcoding proxy.

키워드 : *트랜스코딩, 스트리밍, 프록시 캐싱*
Keywords : *transcoding video proxy, mobile streaming, proxy caching*

1. 서론

최근 급속히 확대되고 있는 다양한 형태의 멀티미디어 서비스를 위해서는 멀티미디어 데이터의 획득, 압축, 전송과 저장에 관련된 기반 기술은 물론 유무선 네트워크를 통합하는 기술 또한 매우 중요하다[2]. 특히 3G가 등장하면서 유선 환경에 적용하던 멀티미디어 데이터를 무선 단말로 서비스가 가능하도록 멀티미디어 데이터를 변환하는 트랜스코딩 프록시에 대한 관심이 증대되고 있다.

트랜스코딩 프록시는 IP 네트워크와 무선 환경이 접속되는 경계에 위치하며, 몇 단계를 거쳐 스트리밍 서비스를 지원한다. 먼저 모바일 사용자가 원하는 비디오의 종류와 단말의 성능, 대역폭 정보를 전송하면 사용자 요청을 관리하는 서버는 스트리밍할 트랜스코딩 프록시를 선택한다. 선택된 트랜스코딩 프록시는 사용자가 요청하는 비디오가 자신의 캐쉬에 들어있는지 확인하여, 캐쉬에 있으면 즉각 서비스하고 그렇지 않으면, 사용자 요청 관리자가 멀티미디어 서버에게 원본비디오를 요청하여 트랜스코딩 프록시로 전달 받는다. 트랜스코딩 프록시는 서버로부터 전송 받은 또는 캐쉬한 원본 비디오를 사용자의 요구에 맞게 변환하고 변환된 비디오를 캐쉬하여 추후 동일한 비디오의 재요청에 대비한다. 따라서, 트랜스코딩 프록시에는 원본비디오와 변환비디오가 함께 캐쉬되며, 각각으로부터 야기되는 서버와 프록시 사이의 네트워크 부하와 트랜스코딩에 따른 CPU 부하를 줄이는 방법이 연구되었다.[1][3][4][5].

* 강원대학교 컴퓨터정보통신공학과 석사과정
** 강원대학교 컴퓨터정보통신공학과 교수, 공학박사

일반적으로 트랜스코딩 프록시에서 네트워크 부하와 CPU 부하를 줄이기 위해 각각 서로 다른 캐싱 알고리즘을 사용하기 때문에 원본비디오 캐싱이나 변환비디오 캐싱으로 편중되는 현상이 발생한다. 즉, 양분된 캐싱 알고리즘은 트랜스코딩 프록시에서 발생하는 네트워크 부하와 CPU 부하를 트레이드-오프 관계로 만든다. 또한 CPU 부하를 줄이기 위해서 크기가 큰 변환비디오를 캐쉬하고 동일한 비디오에 대해 변환비디오와 원본비디오를 중복 저장하게 되어 캐쉬 공간을 효율적으로 사용하지 못하게 된다[3][4].

본 논문에서는 세그먼트 캐싱을 하는 트랜스코딩 프록시에서 네트워크 부하와 CPU 부하를 효과적으로 줄이는 메타데이터 추가 캐싱 기법을 제안한다. 메타데이터 추가 캐싱은 캐쉬하는 데이터를 메타데이터까지 확장하고 자주 요청되는 구간은 변환비디오로, CPU 부하를 줄이는 구간은 메타데이터로 저장함으로써 한정된 캐쉬 공간에서 네트워크 부하와 CPU 부하를 동시에 최소화한다.

본 논문의 구성은 다음과 같다. 2장에서는 트랜스코딩 프록시 캐싱 방식들을 다양한 관점에서 비교, 분석하고, 3장에서는 메타데이터 추가 캐싱의 기본원리와 구현에 대해 살펴본다. 4장에서는 구현한 시스템의 시험과 검증에 대해 기술하고, 5장에서 결론을 맺는다.

2. 관련 연구

2.1 세그먼트 캐싱

기존의 트랜스코딩 프록시[1, 3, 5]는 비디오를 분할하지 않고 비디오 전체를 캐싱해왔으나, 무선 인터넷이 확대되고 단말의 성능이 좋아지면서 유선 환경에서의 분할 캐싱이 트랜스코딩 프록시에 적용되기 시작했다[4].

세그먼트 캐싱은 비디오 전체를 저장하는 대신 비디오 블록들을 세그먼트 단위로 나누어 접근이 많은 비디오의 앞부분을 우선 캐쉬하고 요청이 증가되면 비디오의 뒷부분을 추가한다[2]. 세그먼트 캐싱 중 바이트 적중률이 높은 피라미드 방식에 따라 저장된 비디오의 모습은 그림 1과 같은데, 세그먼트 단계가 증가할수록 세그먼트의 크기는 기하급수적으로 증가한다.

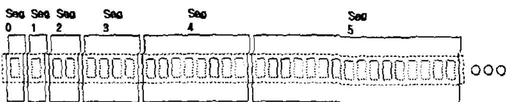

그림 1. 피라미드 분할로 저장된 비디오 모습

그림 1처럼 나누어진 세그먼트들 중 프리패칭과 관련된 앞부분의 세그먼트들은 초기 세그먼트, 나머지 뒷부분의 세그먼트는 후반 세그먼트가 되며, 각각은 그림 2와 같이 따로 저장된다. 초기 지연을 줄이기 위해 우선 캐쉬하는 초기 세그먼트들을 합한 크기는 연속 스트리밍을 보장하는 크기(prefix size)이며, 서버에서 프록시로 전송되는 데이터량을 줄이기 위한 후반 세그먼트들은 비디오의 참조수가 증가함에 따라 그 길이가 늘어난다.

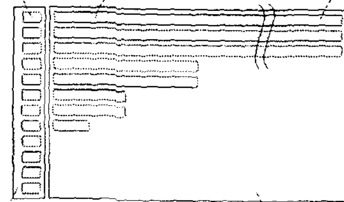

그림 2. 피라미드 분할 방식의 캐싱

2.2 변환비디오 캐싱

변환된 비디오를 캐쉬하면 반복된 트랜스코딩을 방지하여 연산 부하를 줄이고 서버로부터의 원본 비디오 전송을 줄여 네트워크 부하를 줄인다. 그러나 중복된 재생구간에 변환비디오를 모두 저장하여 제한된 캐쉬공간 내에서 바이트 적중률이 떨어지고 네트워크 부하와 CPU 부하를 줄이는 데에 한계가 있다. 변환비디오를 저장하는 TVO[3]에서 비디오의 인기도 $D_{cpu}(v)$는 식 (1)과 같다. v는 변환비디오, $c(v)$는 원본비디오를 로 변환할 때 필요한 CPU 소모량이다.

$$D_{cpu}(v) = \frac{f(v) \cdot c(v) \cdot l(v)}{s(v)} \quad (1)$$

2.3 메타 캐싱

메타데이터를 저장하는 메타 캐싱은 데이터 자체를 트랜스코딩하는 대신 메타데이터로 트랜스코딩함으로 CPU 부하가 줄어든다. 메타데이터를 사용하는 AMTrac[5]에서 메타데이터의 인기도 $D(m)$은 식 (2)와 같다. α는 메타데이터의 크기이고 β는 메타데이터를 이용하여 트랜스코딩할 때 필요한 CPU 소모량이다.

$$D(m) = \frac{\beta}{\alpha} \quad (2)$$

메타데이터는 변환비디오에 비해 크기가 작고 CPU 부하 축소 효율이 좋아 캐쉬공간이 작아도 된다. 그러나 사용자 요청이 있을 때 서버로부터 원본비디오를 가져와야함으로 네트워크 부하가 크다.

2.4 세그먼트 기반 트랜스코딩 프록시 캐싱

세그먼트 캐싱을 트랜스코딩 프록시에 적용한 세그먼트 기반 트랜스코딩 프록시 캐싱[4]에서는 비디오 변환에 필요한 CPU 소모량을 참조하여 캐쉬된 세그먼트의 인기도를 구성함으로써 CPU 소모가 큰 트랜스코딩이 반복되지 않도록 세그먼트를 대체한다. 트랜스코딩 프록시의 CPU 부하가 증가하면 원본비디오를 캐싱 대상에서 제외하고 변환비디오를 캐싱하여 CPU 부하를 줄이지만 하나의 컨텐트에 대해 여러 비트율의 비디오를 저장함으로써 캐쉬 공간을 많이 소모하고 바이트 적중률을 떨어뜨려 네트워크 부하가 크다.

3. 메타데이터 추가 캐싱

기존 캐싱 방식[1, 3, 5]은 네트워크 부하와 CPU 부하 때문에 사용자 요청이 차단(blocking)되는 현상을 최소화하기 위해 단순히 원본비디오, 변환비디오, 메타데이터 캐싱을 병행하였다. 그러나 캐쉬 공간이 제한되는 환경에서는 캐쉬하는 대상의 특성을 파악하고 서로 다른 데이터를 알맞게 배분, 저장함으로써 네트워크 부하와 CPU 부하 각각을 효과적으로 줄이는 방법이 요구된다.

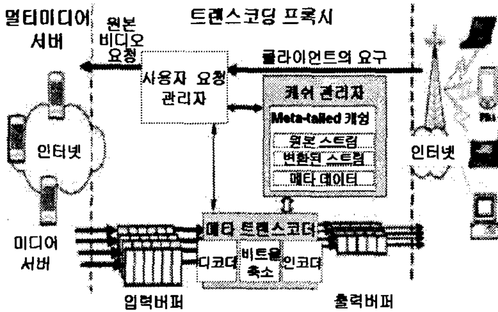

그림 3 메타데이터 추가 캐싱을 사용하는 트랜스코딩 프록시 시스템의 구성

트랜스코딩 프록시가 캐쉬하는 변환비디오, 메타데이터의 네트워크 부하축소 성능과 CPU 부하축소 성능, 객체의 크기를 반영하여 부하축소 효율을 극대화하도록 배분함으로써 트랜스코딩 프록시의 처리량을 최대화하기 위해 그림 3과 같은 시스템을 제안한다. 캐쉬관리자는 비디오 전체를 캐쉬하는 대신 참조수에 따라 비디오의 일부를 변환스트림으로 캐쉬하고, 프록시의 CPU 부하에 따라 메타데이터를 추가로 캐쉬한다.

3.1 기본 원리

그림 4는 요청된 하나의 변환비디오의 인기도를 블록 단위로 나타낸다. 트랜스코딩 프록시는 트랜스코딩 부하를 줄이기 위해 변환비디오를 캐쉬하며 참조수만 반영하는 인기도 $D(b)$에 변환비디오의 블록 b를 만드는 CPU 소모량 $c(b)$를 반영한다. 그림 4에서 Th는 캐쉬에 저장된 블록중 인기도가 가장 낮은 블록의 인기도로서 인기도가 Th보다 높은 블록들은 캐쉬된다. 여기서 인기도를 $c(b)$와 비례하도록 설정하여 CPU 부하를 우선적으로 줄이고[3] 네트워크 부하축소 알고리즘을 사용할 때는 $c(v)$대신 서버-프록시 대역 소모량인 $b(v)$를 곱한다.

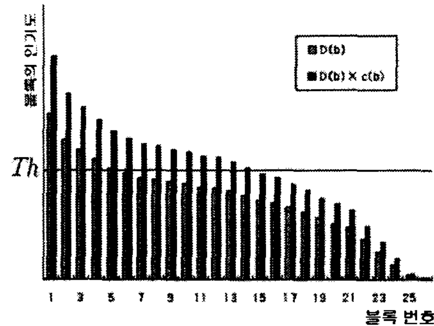

그림 4. 변환비디오에 속한 블록들의 인기도

그림 5는 캐쉬된 변환비디오 하나에 속한 스트림의 역할을 보인다. $D_{net}(b)$, $D_{cpu}(b)$는 각각 네트워크 부하축소 알고리즘과 CPU 부하축소 알고리즘을 사용한 블록들의 인기도이며, 네트워크 부하축소 알고리즘을 사용할 때는, v_1만 캐쉬하나 CPU 부하축소 알고리즘을 사용할 때는 v_c를 추가로 캐쉬한다. 제안하는 메타데이터 추가 캐싱은 CPU 부하축소를 위해 캐쉬하는 v_c의 크기를 줄이고, 새로 저장된 블록들에 의해 Th가 낮게 설정되어 네트워크 부하와 CPU 부하를 동시에 줄이는 v_1의 크기를 늘린다.

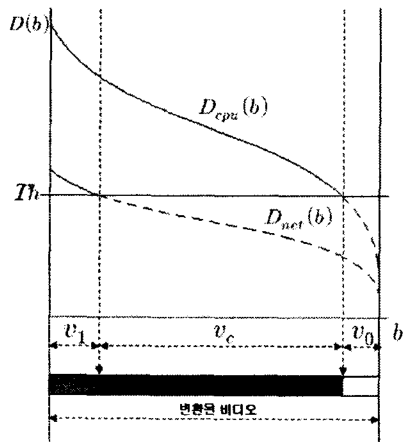

그림 5. 트랜스코딩 연산부하가 높은 비디오에서의 블록 인기도

3.2 동작

변환스트림 캐싱을 개선한 메타데이터 추가 캐싱에서는 그림 5의 v_c 영역을 여러 개의 변환스트림 대신에 메타데이터를 저장함으로써, v_c의 크기가 줄고 v_1이 늘어나 양쪽 부하가 동시에 줄어든다.

메타스트림 추가 캐싱으로 저장된 미디어의 모습은 그림 6과 같다. v_1은 변환스트림을, v_c는 메타데이터를 저장한다. 메타데이터의 길이가 v_c보다 긴 것은 v_c를 변환스트림으로 캐쉬할 때 얻는 CPU 이득을 메타데이터로 얻기 위함이다.

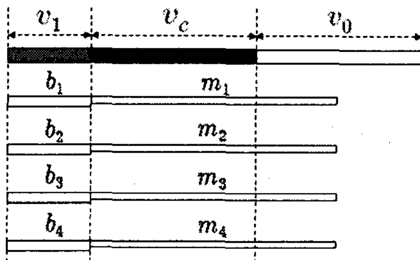

그림 6 캐쉬에 저장된 비디오의 모습

3.2.1 세그먼트의 저장

메타데이터 추가 캐싱을 실제 트랜스코딩 프록시에 적용하기 위해 피라미드 세그먼트 캐싱을 사용한다. 초기세그먼트들은 변환세그먼트로 후반세그먼트들은 변환스트림과 메타데이터로 구성된 세그먼트로 저장한다. 변환세그먼트의 인기도 $D(b_n)$과 메타세그먼트의 인기도 $D(m)$은 식 (3)과 같다.

$$D(b_n) = \frac{f(b_n) \cdot l(b_n) \cdot c_{mean}}{s(b_n)} \quad ,\text{if } c(b_n) \ge c_{mean} \qquad (3)$$

$$D(b_n) = \frac{f(b_n) \cdot l(b_n) \cdot c(b_n)}{s(b_n)} \quad ,\text{if } c(b_n) < c_{mean}$$

$$D(m) = \frac{f(b_n) \cdot l(b_n) \cdot (c_{mean} + \alpha)}{s(b_n)}$$

$$\alpha = (c(b_n) - c_{mean}) \times \frac{\frac{c(m)}{s(m)}}{\frac{c(b_n)}{s(b_n)}}$$

m은 메타세그먼트이며, $c(m)$은 m을 사용한 메타 트랜스코딩[5]에 요구되는 CPU 소모량이다. c_{mean}은 캐쉬에 상주하는 변환스트림들의 평균 $c(b_n)$이다. $c(b_n) < c_{mean}$일 때, 메타세그먼트는 추가되지 않고 $c(b_n) > c_{mean}$일 때, 메타세그먼트는 변환세그먼트와 중복되지 않는 구간에 추가된다. α는 v_c를 변환세그먼트로 저장할 때 줄일 수 있는 CPU 소모량을 메타세그먼트로 얻을 수 있도록 메타세그먼트의 인기도를 조정한다.

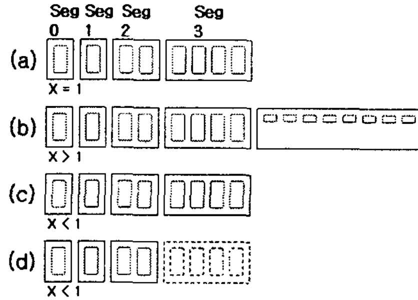

그림 7. 세그먼트의 저장 형태

메타데이터 추가 캐싱으로 비디오를 세그먼트 단위로 저장한 모습은 그림 7과 같다. $c(v) = c_{mean}$일 때, 세그먼트들은 전형적인 피라미드 구조가 된다. $c(b) > c_{mean}$일 때, 메타세그먼트를 추가하고 $c(b) < c_{mean}$일 때, 메타세그먼트는 추가되지 않고 새로 요청된 세그먼트를 저장하기 위한 캐쉬 공간이 부족할 때, 우선 삭제된다.

3.2.2 세그먼트의 삭제

초기 세그먼트들은 크기에 상관없이 LFU로 대체한다. 후반 세그먼트들은 식 (3)의 인기도에 따라 인기도가 가장 낮은 세그먼트와 새로 요청된 세그먼트의 인기도를 비교하여 인기도가 낮은 세그먼트를 삭제하고 높은 세그먼트를 저장한다.

메타데이터를 추가하여 캐쉬한 비디오의 인기도가 감소하여 길이를 줄이고자 할 때, 변환세그먼트를 메타세그먼트로 바꾸게 되는 경우가 발생한다. 해당 비디오가 서비스 중일 때, 트랜스코더로부터 메타데이터를 캐쉬하는 것은 문제가 되지 않으나 세그먼트의 인기도는 해당 비디오가 서비스 중이 아닐 때 감소되므로 메타데이터를 캐쉬할 수 없다. 따라서 메타세그먼트로 바꿀 변환세그먼트를 메타세그먼트의 크기에 맞게 내부 블록을 삭제하여 메타세그먼트를 저장할 공간을 확보한다.

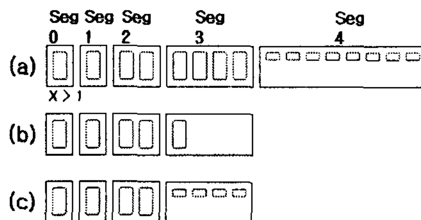

그림 8. 세그먼트의 삭제 절차

그림 8은 인기도가 감소한 변환세그먼트 Seg. 3를 메타세그먼트로 바꾸는 과정이다. 캐쉬에 저장된 세그먼트가 (a)와 같은 모습일 때, 인기도가 Th 이하로 감소한 메타세그먼트는 삭제하고 변환세그먼트에서 메타세그먼트로 바뀔 세그먼트의 내부 블록을 메타세그먼트 크기에 맞춰 삭제한다. 이후, 메타세그먼트로 교체될 세그먼트의 요청이 있으면 (c)처럼 변환세그먼트를 메타세그먼트로 바꾼다.

3.2.3 세그먼트의 대체

세그먼트의 저장과 삭제를 포함한 메타데이터 추가 캐싱의 대체 알고리즘은 그림 9와 같다. 요청된 세그먼트 i가 캐쉬에 없고, i가 $K_{\min}$보다 작으면 초기세그먼트로 캐쉬하고 LFU로 대체한다. 반면 i가 $K_{\min}$과 같거나 크면 후반세그먼트로 캐쉬하고 캐쉬 공간이 부족하면 대체후보 세그먼트 j와 인기도를 비교하여 대체를 결정한다. 만약 변환세그먼트 j의 트랜스코딩 CPU 소모량 $c(v)$가 캐쉬에 저장된 변환세그먼트의 평균 CPU 소모량 보다 크면 세그먼트 j를 메타 세그먼트로 대체하여 캐쉬 공간을 늘린다.

```
if (i<K_min) { // 초기세그먼트 캐싱
    필요시, 세그먼트 대체;
    세그먼트 i 저장;
}
else { // 후반세그먼트 캐싱
    if (처음 요청된 비디오다)
        exit;
    while ((세그먼트 i 저장을 위한 캐쉬 공간 부족)
      and (대체 후보 세그먼트를 발견하지 못했다)) {
        비디오 Q의 삭제 후보 세그먼트 j결정;
        if ((Q가 재생되고 있지 않다)
         and (D(i) > D(j))) // 인기도 비교
            if (세그먼트 j가 변환된 비디오이다)
             and( c(v) > c_mean )
                세그먼트 j를 매타세그먼트로 대체,
                  캐쉬공간 확보;
            else
                세그먼트 j를 삭제,
                캐쉬공간 확보;
    }
    if (세그먼트 i 저장을 위한 캐쉬 공간이 충분하다)
        세그먼트 i 저장;
}
```

그림 9. 세그먼트의 캐쉬 대체 알고리즘

4. 실험 및 결과

4.1 시험 시스템

메타데이터 추가 캐싱의 기능과 성능을 검증하기 위하여 두 대의 데스크탑 PC와 한 대의 노트북으로 그림 10와 같이 시스템을 구성하고 시험을 하였다. 컨텐트 서버는 펜티엄4 2.6GHz CPU에 1GB 메모리를 장착한 데스크탑 PC를, 연산이 많은 트랜스코딩 프록시는 펜티엄4 3.2GHz CPU에 2GB 메모리를 장착하였고, 클라이언트는 펜티엄M 1.7GHz CPU에 512MB 메모리, 801.11b 무선랜을 장착한 노트북을 사용하였다. 트랜스코딩 프록시의 운영체제는 리눅스 Redhat 9.0, 트랜스코더의 디코더는 mencoder[8], 트랜스코더의 인코더는 x264[9]를 수정하여 사용하였다.

시험 시스템의 작업 흐름은 다음과 같다. 트랜스코딩 프록시의 RTSP 클라이언트는 서버로부터 RTSP 패킷을 수신하고 역패킷화를 수행하여 원본스트림을 입력버퍼에 저장한다. 저장된 원본스트림을 클라이언트의 재생 속도에 맞춰 mencoder로 디코딩한 후, 파이프 통신을 위한 mkfifo로 yuv4mpeg 영상을 인코더로 전송한다. x264와 Mpeg-2인코더로 클라이언트의 요구정보에 따라 트랜스코딩하고 변환스트림을 RTSP 서버로 보내 클라이언트에게 전송한다. 자원관리 모듈은 트랜스코딩 프록시의 CPU 부하와 네트워크 부하를 모니터하고 프록시의 과부하 때문에 정상 서비스가 불가능하면 사용자 요청을 차단한다.

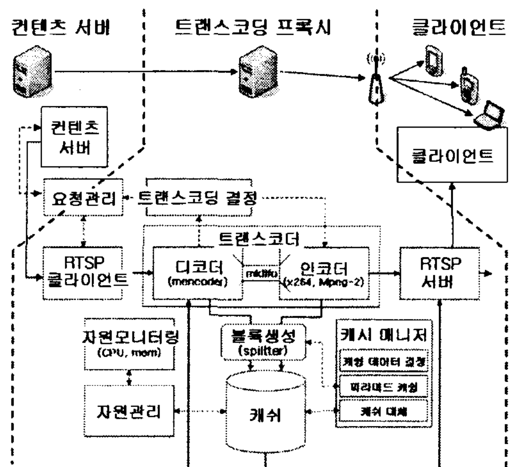

그림 10. 시험 시스템의 구성

실험에 사용한 파라미터들은 표 1과 같다. 전체 캐쉬 용량 C는 요청 가능한 변환된 비디오들의 크기를 합한 값($B \times M \times n$)의 10%이다[2]. 변환비디오에 대한 사용자 요청은 Zipf-like 확률분포를 따르며 GISMO[6]로 생성하였다. 트랜스코딩은 해상도 축소를 하지 않고 비트율 축소만 하며, 비트율 512Kbps의 원본비디오를 384Kbps, 256Kbps, 128Kbps로 변환한다.

표 1. 실험을 위한 파라미터

기 호	설 정 값
C	전체 캐쉬 블록 수 (160,000)
B	변환비디오의 평균 블록 수 (2,000)
λ	평균 요청간 inter-arrival 시간 (60초)
M	서로 다른 원본비디오의 수 (800)
n	변환 버전의 개수
C_{init}	초기 세그먼트들을 캐쉬하기 위한 공간 (10%)
K_{min}	하나의 비디오를 캐쉬하기 위한 초기 세그먼트 수 (6)
Zipf(x, M)	변환비디오의 요청 Zipf-like 분포 (Zipf(0.2, 800))

4.2 시뮬레이션 프로그램

다양한 조건에서 메타데이터 추가 캐싱의 성능을 비교하기 위해 시뮬레이션 프로그램을 작성하였다. 시뮬레이션 프로그램에서는 표 1, 표 2의 파라미터들을 사용하였으며 트랜스코딩 CPU 사용량과 변환세그먼트의 크기는 시험시스템의 측정 결과를 사용하였다. 메타데이터의 크기 효율을 나타내는 α는 $\frac{\text{메타데이터의크기}}{\text{변환비디오의크기}}$이고 메타데이터를 이용한 트랜스코딩의 효율을 나타내는 β는 $\frac{\text{메타트랜스코딩 }CPU\text{사용량}}{\text{트랜스코딩 }CPU\text{사용량}}$이다.

표 2. 메타데이터의 파라미터

	384Kbps	256Kbps	128Kbps
α	0.167	0.25	0.5
β	0.45	0.45	0.45

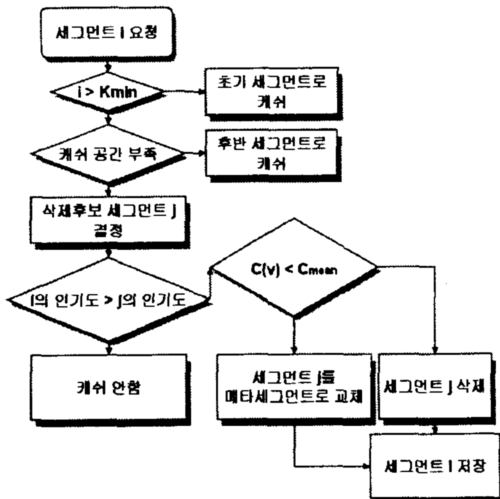

그림 11. 시뮬레이션의 흐름

시뮬레이션 흐름은 그림 11과 같다. 세그먼트 캐싱을 사용하는 메타데이터 추가 캐싱, 변환스트림 캐싱은 하나의 프로그램으로 동작하며 각 캐싱은 세그먼트를 구성하는 데이터의 종류와 세그먼트 인기도 식의 차이로 구분한다. 세그먼트 i의 크기가 프리패칭 파일 사이즈 K_{min}보다 작은 초기세그먼트는 우선 캐쉬하고 후반세그먼트는 인기도가 큰 것부터 캐시한다. 메타데이터 추가 캐싱에서 중복되는 세그먼트는 변환세그먼트를 우선 저장하고, 변환세그먼트를 삭제할 때는 변환세그먼트를 생성하기 위해 필요한 CPU 사용량 $c(v)$를 반영하여 참조수와 $c(v)$가 둘다 작은 세그먼트는 바로 삭제하고 $c(v)$가 큰 변환세그먼트는 메타세그먼트로 대체를 선행한다.

4.3 결과 및 분석

트랜스코딩 프록시 캐싱의 목적은 네트워크 부하와 CPU 부하를 줄여 안정된 서비스를 제공하고, 사용자 QoS를 만족하는데 있다[3]. 메타데이터 추가 캐싱의 성능을 첫째, 사용자 요청이 급격히 증가할 때 얼마나 빠르게 프록시의 부하를 안정시키는가, 둘째, 사용자 요청에 대한 초기지연 시간, 셋째, 네트워크 부하, CPU 부하, 사용자 요청패턴을 반영한 각 캐싱이 요구하는 자원의 특성, 관점에서 비교한다.

4.3.1 요청순서에 따른 부하변동

캐싱 방식이 사용자 요청의 변화에 얼마나 빠르게 적응하는지를 보기위해 요청순서에 따른 부하변동, 즉 시험 시스템에서 완료된 세션, 평균 CPU 점유율, 서버-프록시 대역폭을, 천 번의 요청 중 백 개씩 평균한 값으로 측정하였다.

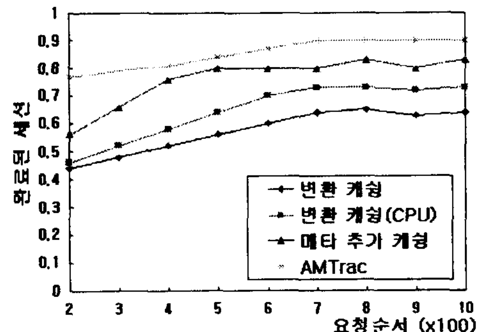

그림 12. 완료된 세션

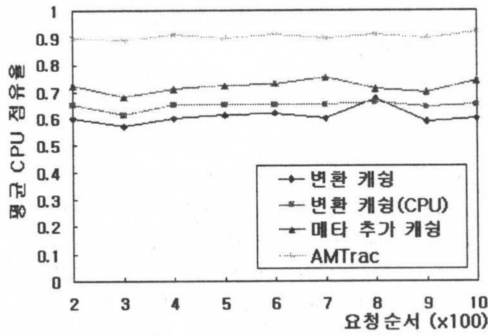

그림 13. 평균 CPU 점유율

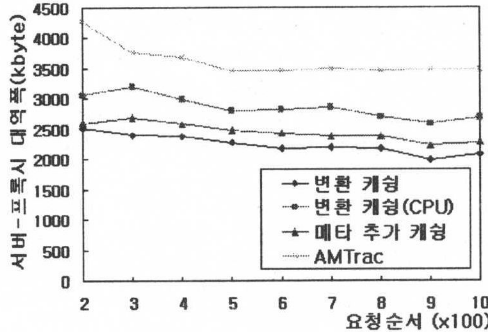

그림 14. 서버-프록시 대역폭

메타데이터 추가 캐싱은, 후반 세그먼트로 추가된 메타데이터가 변환스트림을 생성하는 CPU 부하를 줄이므로 변환비디오 방식에 비해 사용자 요청에 빠르게 적응한다. 또한 바이트 적중률이 높고 사용자 요청이 적게 차단되어 서버와 프록시 사이의 네트워크 부하가 작고 CPU 활용율이 높다. AMTrac은 완료된 세션 수에서는 우수하지만 그림 14처럼 대역폭 소모가 커 여러 프록시를 사용할 때는, 서버에서 병목이 발생할 수 있다.

4.3.2 초기 지연

사용자에게 변환스트림을 전송하는 트랜스코딩 프록시는 초기세그먼트를 변환스트림으로 채워 프리패칭한다. 초기세그먼트에서 적중이 되지 않으면 원본스트림을 서버로부터 가져와야하므로 전송지연과 트랜스코딩 지연이 동시에 발생한다.

그림 15은 캐쉬의 크기에 따라 초기지연이 야기되는 요청의 비율을 보였다. 초기세그먼트를 저장하는 캐쉬의 크기가 전체 캐쉬 용량의 30% 이상이면 거의 모든 비디오의 초기세그먼트가 캐쉬에 저장되므로 초기지연이 발생하지 않는다. 초기세그먼트는 비디오의 도입부에 해당하는 스트림만으로 구성되므로, 전체 적중률을 고려하여, 초기세그먼트를 저장하는 캐쉬의 크기는 가급적 줄여 전체 캐쉬크기의 10%로 설정한다.

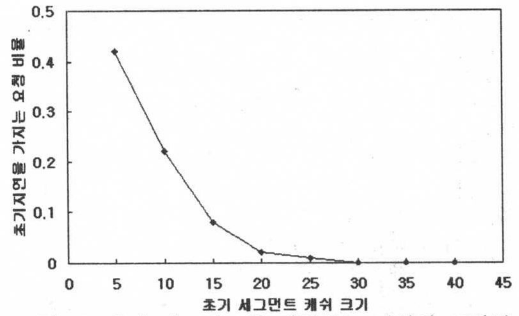

그림 15. 초기 세그먼트를 저장하는 캐쉬의 크기별 서비스 초기지연

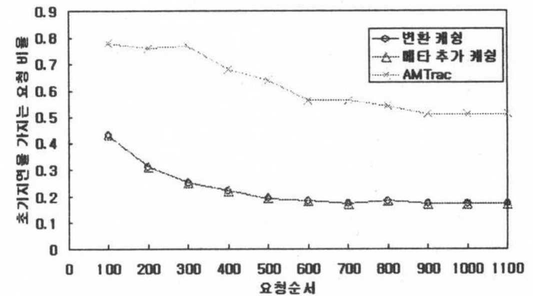

그림 16. 요청순서에 따른 초기지연

요청이 증가함에 따라 초기지연은 그림 16과 같이 변화한다. 메타데이터 추가 캐싱과 변환스트림 캐싱은 초기세그먼트에 한해 동일한 세그먼트를 캐쉬하므로 메타데이터 추가 캐싱은 기존 세그먼트 캐싱과 동일하게 초기지연이 낮다. AMTrac에서는 비디오를 분할하지 않고 전체를 저장하고 메타데이터를 우선 저장함으로써 초기지연이 크다.

4.3.3 요청분포별 자원의 요구량

시뮬레이션으로 각 캐싱 기법이 요구하는 시스템 자원의 특성을 비교하였다. 모든 사용자 요청이 차단없이 수용되고, 가장 많은 대역을 요구하는 메타 캐싱에서도 서버와 프록시 사이의 패킷 손실은 없다고 가정하였다.

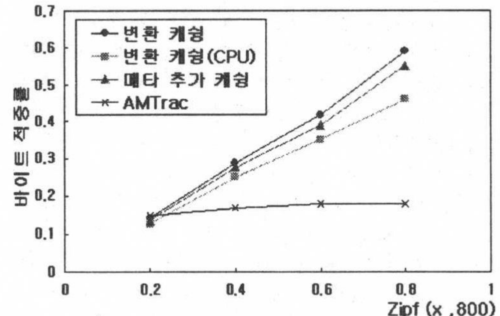

그림 17. 바이트 적중률

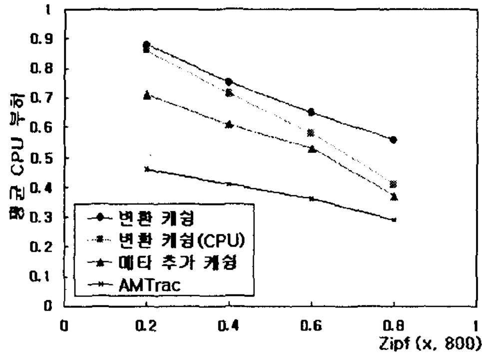

그림 18. CPU 부하

그림 17, 그림 18은 요청 분포에 따른 바이트 적중률과 CPU 부하를 보인다. 사용자 요청 패턴을 생성하는 함수인 Zipf(x, M)에서 x가 0에 가까울수록 인기 비디오들의 요청 빈도가 낮고 대체가 빈번해져 바이트 적중률이 감소한다. x가 0.2에서 0.8까지 변하여 사용자 요청빈도 차이가 커지면, 메타데이터 추가 캐싱은 네트워크 부하와 CPU 부하를 기존 방식보다 더 많이 줄인다.

x가 0.8 일 때, 메타데이터 추가 캐싱은 변환스트림 캐싱에 비해 네트워크 부하는 2% 크지만 CPU 부하는 16% 작고, 변환스트림 캐싱(CPU 부하축소 알고리즘)에 비해 네트워크 부하는 9% 작고 CPU 부하도 10% 작다. 즉 메타데이터 추가 캐싱은 기존 방식이 가지는 네트워크 부하와 CPU 부하 사이의 트레이드-오프 관계를 양쪽 부하를 동시에 줄임으로써 상쇄하고 전체 처리량을 증가시킨다.

5. 결론

무선 환경을 위한, 트랜스코딩 프록시에서는 일반 멀티미디어 프록시와는 달리 많은 연산을 수반하는 트랜스코딩이 요구되므로 시스템을 설계할 때 네트워크 부하 외에 CPU 부하도 고려해야 한다. 트랜스코딩 프록시를 위한 기존의 캐싱 기법들은 트랜스코더의 입력, 출력, 트랜스코딩을 위한 메타 정보 중의 한 가지 요소로 편중되어 프록시의 성능을 충족하는데 한계가 있거나, 네트워크부하와 CPU부하 사이의 트레이드오프관계를 보인다.

본 논문에서는 세그먼트 캐싱을 하는 트랜스코딩 프록시에서 네트워크 부하와 CPU 부하간의 트레이드오프관계를 개선하고 네트워크 부하와 CPU 부하를 효과적으로 줄이는 메타데이터 추가 캐싱을 제안, 구현하였다. 메타데이터 추가 캐싱에서는 자주 요청되는 구간은 변환비디오를, CPU 부하를 줄이는 구간은 메타데이터를 저장함으로써 제한된 캐쉬 공간내에서 양 부하를 동시에 줄이니다. 리눅스 환경에서 제안된 기법을 구현, 시험한 결과, 네트워크 부하는 9%이상, CPU 부하는 10%이상 감소됨을 확인하였다.

참 고 문 헌

[1] R. Rejaie, M. H. H. Yu, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet," *in Proc. IEEE INFOCOM*, Vol.2, pp.980-989, Israel, March 2000.

[2] K. Wu, P. S. Yu, J. Wolf, "Segment-Based Proxy Caching of Multimedia Streams," *in Proc. WWW*, Hong Kong, May 2001.

[3] Xueyan Tang, Fan Zhang, and Samuel T. Chanson, "Streaming Media Caching Algorithms for Transcoding Proxies," *in Proc. ICPP 2002*, Vancouver, Canada, August. 2002.

[4] K. Chang, R. Wu, T. Chen, "Efficient Segment-Based Video Transcoding Proxy for Mobile Multimedia Services", *ICME 2005*, pp. 755-758, July 2005.

[5] D. Liu, S. Chen, B. Shen, "AMTrac: Adaptive Meta-caching for Transcoding" *in Proc. ACM International Workshop on NOSSDAV 2006*, Rhod Island, May 2006.

[6] http://csr.bu.edu/gismo/

[7] http://streamingmedia.usc.edu

[8] http://www.mplayerhq.hu

[9] http://x264.nl