생방송 비디오 스트림 전송을 위한 프록시 서버의 설계와 구현

김현민, 낭종호
서강대학교 컴퓨터학과

A Design and Implementation of Proxy Server for Live-Video Stream Transmission

Kim Hyun Min, Nang Jong Ho
Department of Computer Science, Sogang University

요 약

생방송 비디오 프록시는 사용자에게 최신의 비디오 데이터를 전송해야 한다. 그러나 큰 파일을 나누어 미리 캐싱하는 기존의 방법으로는 최신의 데이터를 보여줘야 하는 생방송 VOD의 제약을 만족시킬 수 없다. 본 논문에서는 생방송 비디오 스트림을 대상으로 효과적으로 캐싱할 수 있도록 클라이언트의 요청을 받는 큐와, 생방송 비디오 데이터를 저장하는 버퍼풀을 구성하였고, 이들이 제대로 동작하기 위한 동기화 과정을 설계하였다. 이 구조의 특징은 프록시가 클라이언트들의 요청을 일정시간 동안 모아서, 미디어 서버에게 한번만 요청한다는 데에 있다. 미디어 서버는 프록시에게만 요청을 받고 데이터를 전달하게 되며, 모든 부하는 프록시 쪽으로 옮겨간다. 결국 프록시 서버의 개수를 늘리거나 성능을 확장하면 서비스 가능한 클라이언트의 수를 확장시킬 수 있는, 이른바 확장성의 개선을 기대할 수 있다. 실제로 구현해본 결과 지연시간 개선, 확장성 효과 등 프록시로서 만족할 만한 성능이 나옴을 확인하였다. 비연속적인 비디오 데이터를 대상으로 한 본 논문은, 오디오와 같이 연속적인 성질의 데이터에 대한 처리를 보강한다면 수요가 늘고있는 생방송 VOD서비스에 대비한 프록시 개발에 도움을 줄 것이다.

1. 서론

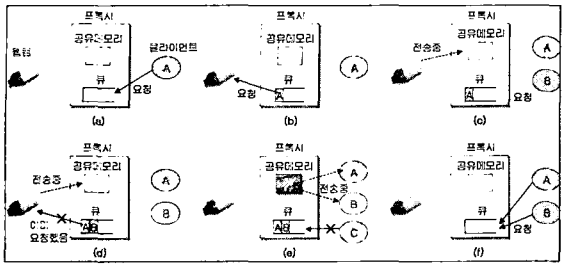
요즈음 초고속 인터넷 환경의 보급이 급증하면서, 네트워크 사용량 역시 폭발적으로 증가하고 있다. 특히 동영상과 같은 용량이 큰 파일에 대한 요구가 많아지면서 심화되고 있는 네트워크 정체현상을 해결하기 위해서는 늘어나는 사이트, 사용자의 수에 비례하여 네트워크 자원 역시 증가되어야 한다. 그러나 단순히 인터넷 백본의 대역폭을 확장하는 것은 매우 비효율적이고 비싼 해결책이다. 게다가 그러한 방법은 인터넷의 병목을 없애기보다는 사이트 쪽으로 이동시키는 결과를 초래한다[1]. 그에 대한 해결책으로 프록시 서버에 대한 연구가 있었으며 지연시간 개선, 확장성 면에서 성능향상을 얻을 수 있었다. 그러나 대부분의 프록시 서버에 대한 연구는 텍스트를 대상으로 하는 것으로서, 점차 데이터의 형태가 예전의 텍스트나 그림 이미지 파일에서 연속적인 미디어 스트림의 형태로 변화하고 있는 상황을 고려한다면, 적합한 새로운 방법을 생각해낼 필요가 있다. 멀티미디어 파일을 대상으로 하는 프록시 서버에 대한 여러 연구들이 있어왔다[1,2,3,4]. 이 연구들은 멀티미디어 파일의 크기가 크고, 그에 대한 요청은 순차적으로 이루어진다는 특성을 이용하여 큰 파일을 캐싱할 수 있는 적절한 크기로 나누고, 기존의 텍스트 대상의 프록시 서버에서 사용하였던 캐싱 알고리즘[5]과의 조합을 통해 개선하였다. 이들 논문들은 큰 용량의 파일을 어떠한 단위로 나누며, 어떻게 캐싱할 것인지에 대한 내용을 중점으로 하고 있다. 그러나 이들 논문들은 저장된 파일에 대한 VOD에는 유용하지만 생방송으로 제공하는 멀티미디어의 스트리밍에 대한 프록시 서버로는 부적절하다. 본 논문에서는 기존의 멀티미디어 프록시 서버와 달리 생방송 비디오 이미지를 실시간으로 보여주는 프록시 서버를 제안한다. 생방송 비디오 이미지에 대한 실시간 프록시 서버라 함은 클라이언트는 항상 최신의 이미지를 제공받아야 하고, 프록시 서버로서의 기능을 가져야 한다. 본 논문에서는 이런 기능을 제공하기 위해서 클라이언트들의 요청을 받는 큐와, 미디어 데이터를 저장하는 공유메모리를 구성하였으며, 이 두 요소들에 의해 효과적으로 캐싱이 되도록 하는 알고리즘을 제안하였으며 실제로 리눅스 상에서 구현하고 그 성능을 분석하였다. 성능 분석 결과, 생방송 스트림에 대해서 프록시 서버로서의 장점을 보임을 확인할 수 있었다.

2. 생방송 비디오 프록시의 설계

2.1 실시간 비디오 전송과정

<그림 1>은 미디어 서버와 클라이언트 사이에서 비디오 데이터

가 전송되는 과정을 보여주고 있다. 이 그림에서 보듯 프록시는 클라이언트의 요청을 저장하는 큐와 비디오 데이터를 저장하는 메모리 공간으로 이루어진다. 프록시는 클라이언트들에게

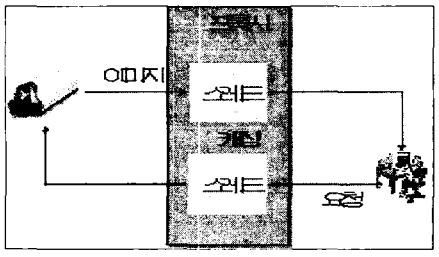


<그림 1> 실시간 비디오 전송과정

요청을 받으면 이를 큐에 쌓아놓고 있다가 비디오 데이터가 오면 큐에 쌓인 모든 요청의 주체들인 클라이언트에게 그 데이터를 전송한다. 이때 중요한 것은 <그림 1>(d)에서 보듯 이미 어떤 클라이언트가 요청한 상태에서 다른 클라이언트가 또 요청을 했다면 미디어 서버에 재차 요구하지 않는다는 점이다. 생방송 비디오임을 감안한다면 프록시가 보내주어야 할 데이터가 같기 때문이다. <그림 1>(e)는 프록시가 클라이언트에게 비디오 데이터를 전송하는 동안에는 데이터 요청을 받지 않음으로써 클라이언트 C에게 최신 데이터 전송을 보장하게 됨을 보여주고 있다.

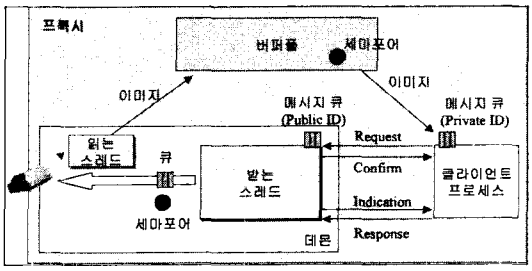
2.2 전체 시스템 구조

<그림 1>에서의 동작을 위해서 시스템을 두 개의 스레드로 분리하였다. 하나는 클라이언트로부터 요청을 기다리고 있다가 요청이 들어오면 이를 큐에 넣고 비디오 데이터를 요청하는 것이고, 다른 하나는 미디어 서버로부터 비디오 데이터를 기다리고 있다가 비디오 데이터를 수신하면 이를 요청한 클라이언트에게 전송하는 스레드이다. 그 구조는 <그림 2>와 같다.



<그림 2> 시스템을 이루는 두 스레드

<그림 2>에서의 구조를 바탕으로 전체적인 동작 과정을 알아볼 수 있도록 <그림 3>에 시스템을 이루는 세부 요소를 표현하였다. <그림 1>에서 보았듯이 프록시를 이루는 두 요소는 공유메모리와 큐이다. 먼저, 클라이언트가 요청을 하면 프록시에 프로세스가 생성되어 데몬에게 요청을 전달한다. 데몬의 두 스레드 중 요청을 "받는 스레드"는 큐에 요청을 넣고 비디오 데이터 요청을 보낸다. 그리고 받는 스레드는 다시 요청 대기 상태로 들어간다. 그러다가 비디오 데이터가 들어오면 이 데이터를 기다리던 "읽는 스레드"는 버퍼풀에 복사하고 큐에 저장된 클라이언트의 리스트를 보고 요청을 한 모든 클라이언트에게 전송한다. 전송이 완료되면 다시 비디오 데이터 대기 상태로 들어간다. <그림 3>에서 데몬 프로세스와 클라이언트 프로세스는 메시지 큐를 통해 통신하게 되며, 버퍼풀은 공유메모리를 사용하여 두 프로세스에서 모두 접근할 수 있도록 한다.



<그림 3> 시스템 전체구조

2.3 캐싱단위

프록시 서버에서 중요한 것은 캐싱단위를 정하는 것이다. 기존 연구에서는 가변 크기의 세그먼트로 나누는 방법[9], 인코딩을 계층을 나누어서 하는 방법[10], 그리고 첫부분의 일부 프레임을 미리 읽어오는 방법[3]을 사용하였다. 캐싱의 단위는 의미있는 최소 데이터 단위가 되어야 한다. 본 연구에서는 비디오의 한 프레임을 캐싱 단위로 결정하였다. 30fps, 1Mbps의 I-frame only MPEG1 비디오의 경우 한 프레임의 크기가 약 32K정도이고, GOP를 가지는 MPEG에서 I-frame도 수십 K정도의 크기이며, 이는 캐싱하기에 충분히 작은 크기이다.

3. 구현 및 실험

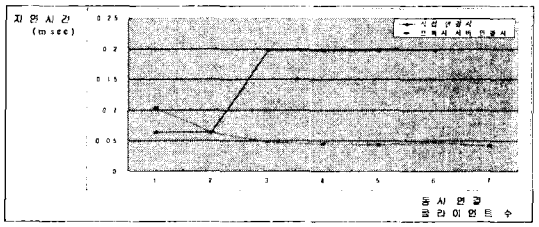
3.1 구현 및 실험 환경

이제까지 설계한 시스템을 실제로 리눅스 기반으로 구현해 보았다. 본 실험에서 사용한 커널 버전은 2.4.9이며 구현을 위해 필요한 세마포어, 메시지큐, 공유메모리 등의 IPC라이브러리는 리눅스에서 제공하는 시스템V 계열을 사용하였으며, 스레드는 pthread라이브러리를 이용하였다. 또 실험을 위해서 하드웨어 사양이 Intel PentiumIII-933Mhz, RAM 256MB, 10/100Mbps Ethernet 인 서버 실험환경을 갖추었다. 실험에 사용한 미디어 서버는 웹캠[6]을 사용하였으며 편의상 웹캠, 프록시, 클라이언트 모두 같은 LAN상에 위치한 상태에서 실험을 하였다.

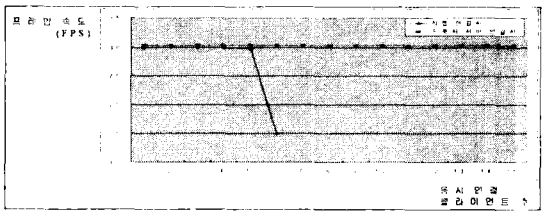
3.2 실험결과

<그림 4>는 웹캠에 이미지를 직접 요청할 때와 프록시 서버를

통해서 요청할 때의 두 경우를 클라이언트 수를 늘려가면서 지연시간과 프레임 속도를 측정한 결과이다. 지연시간이란 첫번째 요청에 대해 응답이 오는 시간이다. 웹캠에 직접 요청했을 경우와 달리 프록시 서버를 통해 요청을 하면, 캐싱이 안 돼있을 경우 데이터가 웹캠에서 프록시 서버를 거쳐서 전송되므로 지연시간이 커진다. 그러나 <그림 4>(a)에서 보듯 클라이언트 수가 늘어날수록 캐싱이 돼있을 확률이 커지므로 지연시간이 개선된다. 프레임속도에 있어서도 프록시 서버를 사용했을 때에 저하되지 않음을 알 수 있다.



(a) 지연시간



(b) 프레임속도

<그림 4> 시스템을 이루는 두 스레드

4. 결론 및 앞으로의 연구방향

이 논문에서의 중요한 요소는 생방송 이미지를 실시간으로 받기 위한 요소들 즉, 버퍼풀과 큐이다. 프록시에서는 캐싱이 가장 중요한데, 이들 두 요소는 각각 캐싱에 중요한 역할을 한다. 버퍼풀은 이미지 데이터를 캐싱하고, 큐는 클라이언트로부터의 요청을 캐싱한다. 이 둘은 동작과정 안에 동기화를 포함하고 있으며, 이들의 동작을 중심으로 전체 시스템이 돌아가게 된다. 실험 여건상 많은 수의 클라이언트가 접속할 때 생길 수 있는 변수들에 대해서는 실험해보지 못하였다. 물론 실험에 사용한 웹캠의 하드웨어 제약이 크기 때문에 프록시 서버를 사용함으로써 확장성에서는 월등한 성능을 보인 것은 사실이지만 엄청난 사용자들의 수요가 예상되는 실제 상황에서의 실용성은 증명할 수 없었다. 한편, 클라이언트가 CGI이기 때문에 클라이언트가 증가할 때 늘어나는 프로세스의 부하도 고려하지 못했다. 그러나 이제까지 프록시 서버를 이용해 확장성을 추구한 논문들처럼 프록시 서버를 계층적으로 구성하면 큰 숫자의 사용자에 대한 서비스도 충분히 할 수 있으리라고 생각한다.

본 논문에서는 비연속적인 비디오 프레임을 제공하는 웹캠을 대상으로 실험하였지만, 실제로 가장 많이 사용되는 스트리밍은 비디오 데이터가 프레임마다 크기가 일정하지 않고, 오디오 데이터와 같이 연속적인 데이터들을 포함한다. 이러한 데이터들에 대해서는 다른 방법을 사용해야 하는데 예를 들어 MPEG 비디오의 경우에는 캐싱단위를 프레임이 아닌 GOP단위로 바꾸는 것을 생각해볼 수 있다. 앞으로는 이외에 오디오와 같이 연속적인 데이터들에 대해서 어떻게 나누어서 적용해야 하는가에 대해서 연구가 진행되어야 하겠다.

참고문헌

[1] Yong Woon Park; Kun Hyo Baek; Ki Dong Chung, "Reducing network traffic using two-layered cache servers for continuous media data on the Internet," Computer Software and Applications Conference, 2000. COMPSAC 2000. The 24th Annual International , 2000 Page(s): 389~394.

[2] Wagner Meira Jr., Erik Fonseca, Cristina Murta and Virgilio Almeida, "Analyzing Performance of Cache Server Hierarchies," Proceedings of the XVIII International Conference of the Chilean Computer Science Society, 1998.

[3] Sen, S.; Rexford, J.; Towsley, D. "Proxy prefix caching for multimedia streams," INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , Volume: 3 , 1999. Page(s): 1310~1319.

[4] Rejaie, R.; Haobo Yu; Handley, M.; Estrin, D. "Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet" NFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , Volume: 2 , 2000 Page(s): 980-989.

[5] Busari, M.; Williamson, C., "On the sensitivity of web proxy cache performance to workload characteristics," INFOCOM 2001. Proceedings. IEEE , Volume: 3 , 2001 Page(s): 1225~1234.

[6] Seyeon Tech. FlexWatch Official Homepage, http://www.flexwatch.com/.

[7] Sven Goldt, Sven van der Meer, Scott Burkett, Matt Welsh, "The Linux Programmer' s Guide," http://users.unitel.co.kr/~sangeun5/linux/lpg.html.

[8] Fabmi, H.; Latif, M.; Sedigh-Ali, S.; Ghafoor, A.; Liu, P.; Hsu, L.H. , "Proxy servers for scalable interactive video suppor t," Computer , Volume: 34 Issue: 9 , Sept. 2001 Page(s): 54 ~60.

[9] K.L.Wu, P.S.Yu, and J.L. Wolf, "Segment-Based Proxy Caching of Multimedia Streams," Proc. 10th Int'l World Wide Web Conf., Elsevier Science, Amsterdam, 2001, pp.36~44.

[10] Rejaie, R. , "On design of adaptive Internet streaming applications: an architectural perspective," Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on , Volume: 1 , 2000 Page(s): 327~330.