# 연속형 미디어 스트림 서비스를 위한 네트워크 캐쉬 관리 정책

*박세철[0] **손유익
*동의공업대학 컴퓨터정보계열, **계명대학교 컴퓨터공학과
*scpark@dit.ac.kr, **yeson@keimyung.ac.kr

# A Network Cache Management Policy for Continuous Media Objects Service

*Seh Chul Park[0] **Yoo Ek Son
*Dept. of Computer Science, Dongeui Institute of Technology
**Dept. of Computer & Electronic Engineering, Kei Myung University

요 약

본 논문에서는 패칭 기법[3]을 사용한 프락시 관리 기법을 사용하여 연속형 스트림 서비스를 하는 스트림 서비스 기법을 제안한다. 제안한 기법에서는 프락시에 캐쉬된 데이터의 양에 따라 스트림 전송 방식을 달리한다. 첫째, 요청된 객체 전체가 캐쉬되어 있을 경우 프락시 만으로 서비스 한다. 둘째, 요청된 객체가 전혀 캐쉬되어있지 않을 경우 후행 스트림들이 서버로부터 객체를 전송할 때 발생하는 초기 지연을 상쇄할 만큼의 데이터를 선반입한다. 셋째, 일부분 만이 캐쉬된 경우에는 해당 객체를 요청하는 스트림 사이에 존재하는 데이터 양만큼을 프락시에 패칭하며 이 경우에는 사용자 노드는 두개의 채널을 열어 하나는 프락시에 패칭된 데이터를 읽는데 사용하며 또 하나의 채널로는 서버로부터 나머지 부분을 읽어오는데 사용한다.

## 1. 서 론

프락시 또는 네트워크 캐쉬은 사용자와 서버의 중간 지점에 캐쉬를 설치함으로써 중앙 서버로의 네트워크 교통량을 줄이기 위한 목적으로 소개되었다[1,2]. 그러나 기존의 네트워크 캐쉬는 주로 텍스트 이미지 등의 종래의 데이터의 캐쉬에 적합하도록 설계되었으며 캐쉬 정책도 파일 서버의 캐쉬 정책을 그대로 적용하였으므로 대용량의 저장 공간을 차지하고 실시간으로 전송되어야 하는 연속형 미디어 오브젝트의 캐쉬에는 적합하지 않다[5,7,8]. 최근 들어 연속형 미디어 오브젝트의 스트림 서비스를 위한 프락시 캐쉬에 관한 연구가 진행되고 있으나 캐쉬 정책 전반에 관한 내용보다는 초기 지연을 줄이기 위한 선반입 기준[5,7], VBR 스트림의 불 규칙성(burstiness)을 평활화하기 위한 연구[6] 등 주로 네트워크의 전송 대역폭의 가변성을 줄이기 위한 국부적인 주제에 맞추어서 진행되어 왔다.

본 논문에서는 패칭 기법[3]을 사용한 프락시 관리 기법을 사용하여 연속형 스트림 서비스를 하는 스트림 서비스 기법을 제안한다. 제안한 기법에서는 프락시에 캐쉬된 데이터의 양에 따라 스트림 전송 방식을 달리한다. 첫째, 요청된 객체 전체가 캐쉬되어 있을 경우 프락시 만으로 서비스 한다. 둘째, 요청된 객체가 전혀 캐쉬되어있지 않을 경우 후행 스트림들이 서버로부터 객체를 전송할 때 발생하는 초기 지연을 상쇄할 만큼의 데이터를 선반입한다. 셋째, 일부분 만이 캐쉬된 경우에는 해당 객체를 요청하는 스트림 사이에 존재하는 데이터 양만큼을 프락시에 패칭하며 이 경우에는 사용자 노드는 두개의 채널을 열어 하나는 프락시에 패칭된 데이터를 읽는데 사용하며 또 하나의 채널로는 서버로부터 나머지 부분을 읽어오는데 사용한다. 또한 공간 재 할당 정책으로는 기존의 캐쉬 공간을 새로이 요청된 오브젝트의 캐쉬을 위해 무조건 할당하지 않고 조건을 만족하는 경우에만 공간 재할당을 하는 선별적인 공간 재할당 방법을 적용하여 불 필요한 공간 재할당을 방지함으로써 캐쉬의 히트율을 높이고 불필요한 할당 오버헤드를 줄일 수 있다.

## 2. 패칭에 기반한 프락시 서버 운영 기법

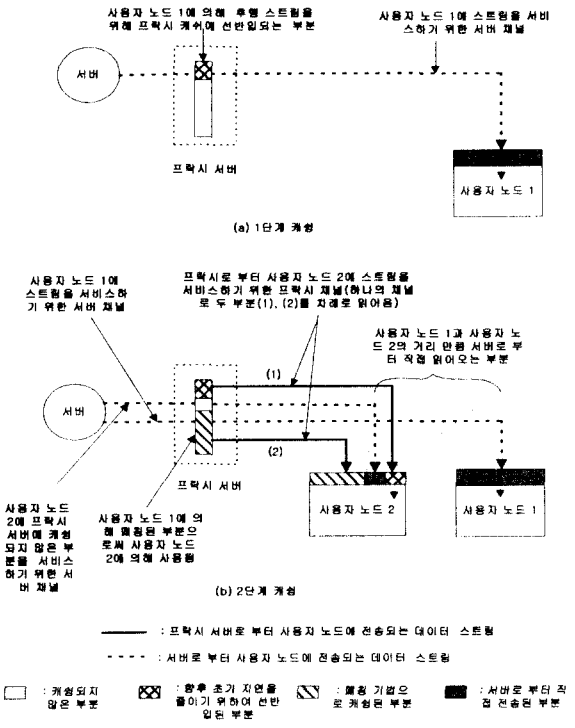
### 2.1 프락시 서버를 이용한 패칭의 목적

패칭[3]은 데이터 요청이 발생하였을 때 서버 측으로부터 스트림 서비스를 받는 동시에 동일한 데이터를 서비스 받고 있는 스트림의 전송 채널을 공유한 다음 사용자 측 노드의 메모리 또는 디스크 등의 자원을 사용하여 데이터를 캐쉬한 다음 서비스 함으로써 캐쉬한 부분 만큼의 서버의 로드를 줄일 수 있다. 그러므로 서버의 채널 사용을 최소화하기에는 적합한 방법이지만 사용자 노드의 자원을 이용하여 데이터를 캐쉬함으로써 캐쉬된 데이터를 재 사용할 수 없다는 문제점이 있다. 따라서 패칭이 프락시 캐쉬 상에 이루어져야 오브젝트 전송에 필요한 자원 절약 효과를 배가시킬 수 있다.

### 2.2 프락시를 이용한 패칭의 개념

기존의 네트워크 캐쉬에서는 캐쉬의 대상이 주로 텍스트나 이미지 등 비교적 크기가 크지 않기 때문에 캐쉬 단위가 주로 오브젝트가 된다[8]. 그러나 이미지나 텍스트와는 달리 오디오나 비디오 등의 연속형 미디어 오브젝트의 경우 압축된 형태라고 하더라도 대용량의 저장 공간이 필요하다. 따라서 캐쉬되는 단위가 오브젝트라면 캐쉬할 수 있는 오브젝트의 수가 제한되므로 이로 인한 공간 재할당 오버헤드가 커진다. 이러한 이유로 인하여 멀티미디어 오브젝트의 캐쉬에 관한 기존의 연구[5,7]에서도 캐쉬 단위를 오브젝트가 아닌 오브젝트의 일부분 만을

캐싱하는 부분 캐싱 기법을 사용하고 있다. 본 논문에서는 [그림 2.1]에서 처럼 프락시에 캐싱되는 부분을 두 스트림 사이에 존재하는 분량 만큼으로 두는 패칭 기반의 공간 할당 기법을 사용하여 패칭된 부분이 후행 스트림들이 공유할 수 있도록 한다. 그러나 패칭의 경우 요청된 오브젝트의 선두 부분은 항상 서버로부터 읽어와야 하므로 이러한 문제를 해결하기 위하여 요청된 오브젝트가 전혀 캐싱되어 있지 않은 경우 서버로부터 오브젝트를 전송할 때 발생하는 전송 지연을 상쇄할 만큼의 데이터 분량을 선반입할 수 있도록한다(이를 본 논문에서는 전방 분할로 칭한다).




[그림 2.1] 프락시 서버를 이용한 패칭 및 선반입

### 2.3 프락시 서버를 이용한 패칭 알고리즘

패칭을 이용한 프락시 서버의 운영시 오브젝트의 캐싱 상태는 두 단계(two phase)로 구성된다.

(1) 1 단계

요청된 오브젝트가 전혀 캐싱되어 있지 않을 때 : 동일 오브젝트에 대한 이 후의 스트림이 서버를 접근할 때 발생되는 초기 지연을 줄이기 위하여 오브젝트의 초기 일부분 또는 전방 분할을 캐싱 단위로 하여 선반입(prefetching)한다. 즉, 임의의 오브젝트 $o_i$에 대하여 오직 하나의 스트림이 존재할 때의 캐싱 단위 $S_c^s(o_i)$는 다음과 같다.

$$S_c^s(o_i) = \begin{cases} 0, & if\ already\ \ cached \\ S(o_i^F) \end{cases}$$

(2) 2 단계

요청된 오브젝트의 캐싱 상태가 1단계를 거친 상태일 때 : 요청된 오브젝트에 이미 선행 스트림이 존재한다고 하고 계속해서 동일한 오브젝트를 요청하는 새로운 스트림이 발생한다고 하면 선행 스트림과 후행 스트림과의 간격을 측정한 다음 해당 간격에 해당하는 분량의 데이터를 캐싱하며 이 때 캐싱의 주체는 선행 스트림이 된다. 이 때 두 스트림이 형성한 간격에 의해 정해진 캐싱되는 분량(선행 스트림은 $p$ 이고 후행 스트림은 $f$) 을 $S_c^{M(p,f)}(o_i)$ 로 표시하고 다음과 같이 결정한다.

$$S_c^{M(p,f)}(o_i) = \begin{cases} S(o_i^R) - S_C^A(o_i)_{L_p(q)}, S(o_i^F) \geq S(o_i^{p-f}) \\ S(o_i) - S(o_i^{p-f}) - S_C^A(o_i)_{L_p(q)}, S(o_i^F) < S(o_i^{p-f}) \end{cases}$$

where $L_p(o_i)$ : 선행 스트림에 의하여 $o_i$ 의 이미 읽혀진 부분, $S_C^A(o_i)_{L_p(o_i)}$ : $o_i$ 의 캐싱된 부분에서 $L_p(o_i)$ 를 뺀 부분, $S(o_i^{p-f})$ : $o_i$ 의 두 스트림(선행 $p$, 후행 $f$) 사이에 존재하는 데이터 량

위와 같이 캐싱이 결정되면 사용자는 주 채널을 사용하여 프락시 서버를 탐색하여 캐싱된 부분을 서비스받는 동시에 프락시 서버캐쉬에서 발견되지 않는 부분에 대해서는 부 채널로는 동시에 서버로부터 직접 다운로드 받아서 주 채널로 서비스가 되지 않는 부분에 대한 서비스를 한다.

## 3. 실험 결과

### 3.1 캐쉬 설정과 사용자 도착 유형

실험은 썬 마이크로 시스템즈 사의 ES3000(OS 버전 쏠라리스 2.6)상에서 C로 쓰레드 프로그램을 작성하여 시뮬레이션을 실행하였다. 스트림의 요청 형태는 일반적으로 널리 알려진 Zipf 분포($\Theta = 0.271$)를 따른 것으로, 도착 간격은 포아송 분포를 따르는 것으로 가정하였다. 또한 통상 CPU나 메모리 버스 등의 속도가 전체 시스템 성능에 영향을 미치지만 본 실험의 목적은 연속형 미디어 전송에 필요한 서버와 네트워크 채널의 경제적인 운영에 있으므로 저장 서버의 CPU나 메모리의 전송 속도는 충분하다고 가정하였다. 이 밖의 본 실험에 사용된 파라메터가 표 3.1에 나타나있다. [7]에 따르면 라운드 트립 지연(round trip delay)은 네트워크의 교통량에 따라 변화가 심하며 때에 따라서는 수 초에 이를 때도 흔히 있다고 한다. 그러므로 수 초(예를 들면 5초 정도의 분량)에 해당하는 오브젝트의 초기 부분을 캐쉬에 선반입하므로써 중앙에 존재하는 서버로부터 데이터를 전송할 때 발생하는 초기 지연을 줄일 수 있다.

[표 3.1] 네트워크 캐싱에 사용된 파라메터

| 파라메터 | 값 |
|---|---|
| 오브젝트의 수 | 300, 600,900,1200 |
| 오브젝트 길이(재생시간:단위 초) | 30 - 180(sec) |
| 스트림 도착 간격(초) | 1, 2, 3, 5 |
| 초당 스트림 재생률/초 | 4 Mbps |
| 디스크(캐쉬) 전송율/초 | 8 Mbytes/sec |
| 디스크 용량(바이트) | 3 GB |
| 디스크 블록 크기(바이트) | 512 |
| 디스크 회전율(RPM) | 7,200 |
| 최대 탐색 지연 (ms) | 8 |

### 3.2 네트워크 채널의 사용도 및 선반입 버퍼량 변화

첫 번째 실험은 스트림의 서비스에 필요한 네트워크 채널의 사용도를 스트림의 도착 간격과 오브젝트의 수를 변화시켜가면서 행하였다. 채널의 사용도는 요청된 오브젝트를 네트워크를 통하여 전송할 때 필요한 네트워크 자원을 말하는 것으로 실시간 전송을 위해서는 스트림이 종료될 때까지 확보되어야 한다. 본 실험에서는 이를 위하여 피크 타임 때의 각각의 네트워크 경로의 사용도를 측정한 다음 해당 값들의 평균을 구하여 [그림 3.1]에 나타내었다.

[그림 3.2]에서 lru-size는 LRU-SIZE 알고리즘을, lfu는 LFU 알고리즘을, mru는 MRU 알고리즘을 적용한 결과를 나타낸다. 본 논문에서 제안한 알고리즘은 *patch* 로 실험 결과를 보면 사용자의 도착 간격이 5초인 경우를 제외하고는 캐쉉 정책을 사용하는 것이 캐쉉 정책을 사용하지 않는 *pure*의 경우보다 적은 네트워크 채널이 필요하였다. 캐쉉 정책을 사용하지 않는 경우와 캐쉉 정책을 사용하는 경우의 성능 차이는 사용자의 도착율과 네트워크의 레벨이 높을수록 많이 났다.

특히 본 논문에서 제안하는 방식(*patch*) 을 사용하는 경우 사용자의 도착률이 높아질수록(도착 간격이 1 또는 3초) 오브젝트의 수의 변화가 캐쉉 정책에 영향을 미치고 있었다. 이는 오브젝트의 수가 증가하는 반면 네트워크 캐쉬의 크기는 고정되어 있으므로 재 할당이 빈번하게 일어나기 때문이다. 따라서 오브젝트의 수가 증가 할수록 캐쉉되는 오브젝트를 더욱 선별적으로 두어야 할 것으로 판단된다. 본 논문에서 제안한 알고리즘은 사용자의 도착률이 낮은 경우에는 성능 향상을 보이지 않았지만 사용자의 도착률이 높아질수록 다른 캐쉉 정책에 비해 성능이 증가하였다. 특히 도착 간격이 1초인 경우는 다른 알고리즘 대비 최고의 성능을 보였다.

## 4. 결 론

본 논문에서는 인터넷 상에서 연속형 미디어 오브젝트를 서비스할 때의 네트워크 자원을 효율적으로 관리하기 위한 캐쉉과 패칭을 동시에 적용하여 스트림을 서비스하는 정책을 제안하였다. 제안한 알고리즘에서는 미디어 오브젝트를 논리적으로 전방 분할(front-end partition)과 후방 분할(rear-end partition)로 나눈 다음 오브젝트 전체를 캐쉉 단위로 하는 것이 아니라 서비스의 요청 빈도에 따라 단계적으로 오브젝트가 통합 캐쉉되는 정책을 적용하였다.

제안된 알고리즘은 실험을 통하여 평가하였으며 성능의 비교 평가를 위해서 기존의 캐쉉 알고리즘의 성능을 본 논문에서 제안한 알고리즘과 동일한 환경에서 측정하였으며 실험 결과 일부 경우를 제외하고는 제안한 캐쉉 정책의 성능이 위에서 언급한 성능 평가 척도의 모든 면에서 다른 알고리즘의 결과보다 우수한 것으로 나타났다. 네트워크 채널의 경우 다른 캐쉉 알고리즘을 적용하였을 때 30% , 스트림 서비스를 위한 서버 자원의 경우 10%의 개선 효과가 있었다. 그리고 사용자 노드 초기 지연은 다른 알고리즘을 적용한 경우보다 15% 정도의 성능 개선 결과를 보였다.

**[참고문헌]**


[1] S. Glassman, " A Caching Relay for the World Wide Web" , Proc. of First International Worl Wide Web Conference, Geneva, 1994

[2] M. Abrams, C. Standridge, G. Abdulla " Caching Proxies: Limitations and Potentials" , Proc. Fourth International World Wide Web Conference, Boston, 1995

[3] Kien A. Hua Ying Cai Simon Sheu, " Patching : A Multicast Technique for True Video-on-Demand" , ACM Multimedia Bristol, 1997, UK 191 - 200

[4] Wessels, D., Claffy, K " ICP and the Squid web cache" , Selected Areas in Communications, IEEE Journal on Volume: 16 3 , April 1998 , Page(s): 345 - 357

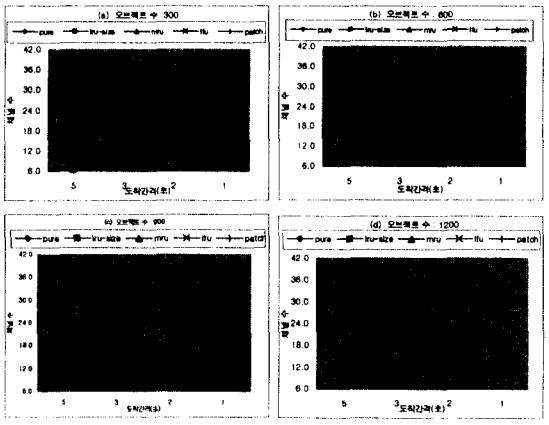
[5] Zhi-Li Zhang; Du, D.H.C., Dongli S, Yuewei Wang, "A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers" , INFOCOM '98. IEEE Computer and Communications Societies. Proceedings. IEEE

[6] Reza Rejaie, Mark Handley, Haobo Yu, Deborah Estrin, " Proxy Caching Mechanism for Multimedia Playback Streams in the Internet" ,1999.

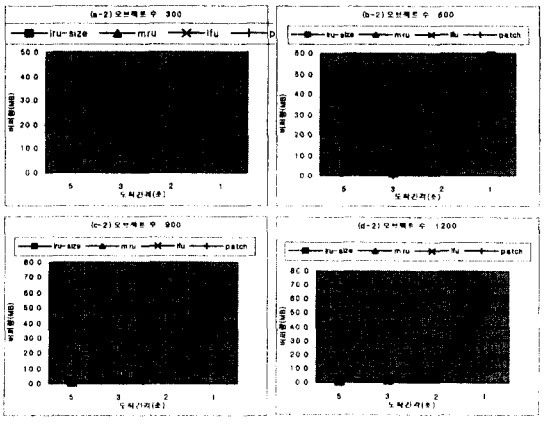
[7] Rexford S., J. Towsley, " Proxy Prefix Caching for Multimedia Streams" , INFOCOM '99. IEEE Computer and Communications Societies. Proc. IEEE Volume: 3 , 1999 , Page(s): 1310 -1319 vol.3

[8] Jia Wang, " A Survey of Web Caching Schemes for the Internet" , Cornell Network Research Group (C/NRG) Department of Computer Science, TR99-1747, May 12, 1999.

[9] H. Yu, D. Estrin, R. Govindan, " A Hierarchical Architecture for Internet-scale Event Services" , Technical Report of Dept. of CS, Univ. of Southern California, 1999 V. 2 , 1998 , Page(s): 660 -667 vol.2

[그림 3.1] 채널 사용도의 변화



[그림 3.2] 서버의 디스크 스트림과 선반입 버퍼량 변화