트랜스코딩 프록시에서 세그먼트 기반 캐쉬 교체 정책

박 유 현[†] · 김 학 영[††] · 김 경 석[†††]

요 약

최근 스트리밍 미디어는 인터넷 상의 많은 트래픽을 유발하고 있다. 기존의 웹을 위한 객체와 마찬가지로 스트리밍 객체 또한 프록시 시스템을 사용하게 되면 여러 이점이 있긴 하지만, 스트리밍 미디어를 캐싱 하는 방법은 기존의 웹을 위한 객체보다 매우 크고 높은 대역폭을 요구하기 때문에 고려해야 할 점이 많다. 또한 많은 종류의 단말들의 다양한 대역폭을 적절히 서비스 하기 위해서 프록시 기능과 트랜스코딩 기능을 함께 할 수 있는 트랜스코딩 프록시가 유용하다. 기존의 프록시는 객체가 프록시 서버에 저장되어 있거나, 그렇지 않은 경우로만 고려되었으나, 트랜스코딩 프록시에서는 같은 객체의 다양한 버전에 대해서 종합적인 효과를 고려하여 최적의 버전 집합을 저장 해야 한다. 또한 최근의 멀티미디어 캐싱 시스템들은 접근 빈도가 높은 앞부분을 캐싱하여 지연 시간을 줄이고 높은 효율을 얻을 수 있도록 하는 방법을 사용한다. 한편 많은 연구에서 멀티미디어 데이터의 효율적인 저장 관리를 위해서 객체를 세그먼트로 나누어 관리하는 방법을 사용하고 있다. 본 논문에서는 부분 캐싱을 사용하는 트랜스코딩 프록시에서 사용자 요청에 따른 후속 작업을 결정하기 위해 4개의 기본 이벤트를 이용하여 9개의 이벤트로 정의한다. 또한 트랜스코딩 프록시 시스템을 위하여 세그먼트 기반의 관리정책을 제안한다. 실험결과는 제안하는 방법이 사용자 지연시간, BHR와 트랜스코딩 데이터의 양이 적음을 보여준다.

키워드 : 트랜스코딩, 프록시 캐싱, 멀티미디어 캐싱, 세그먼트 기반 캐싱

Segment-based Cache Replacement Policy in Transcoding Proxy

Park Yoohyun[†] · Kim Hagyoung[††] · Kim Kyongsok[†††]

ABSTRACT

Streaming media has contributed to a significant amount of today's Internet Traffic. Like traditional web objects, rich media objects can benefit from proxy caching, but caching streaming media is more of challenging than caching simple web objects, because the streaming media have features such as huge size and high bandwidth. And to support various bandwidth requirements for the heterogeneous ubiquitous devices, a transcoding proxy is usually necessary to provide not only adapting multimedia streams to the client by transcoding, but also caching them for later use. The traditional proxy considers only a single version of the objects, whether they are to be cached or not. However the transcoding proxy has to evaluate the aggregate effect from caching multiple versions of the same object to determine an optimal set of cache objects. And recent researches about multimedia caching frequently store initial parts of videos on the proxy to reduce playback latency and archive better performance. Also lots of researches manage the contents with segments for efficient storage management. In this paper, we define the 9-events of transcoding proxy using 4-atomic events. According to these events, the transcoding proxy can define the next actions. Then, we also propose the segment-based caching policy for the transcoding proxy system. The performance results show that the proposing policy have a low delayed start time, high byte-hit ratio and less transcoding data.

Keyword : Transcoding, Proxy Caching, Multimedia Caching, Segment Based Caching

1. 서 론

프록시 캐싱 기법은 네트워크 및 서버의 부하를 줄여 전체 시스템의 성능을 향상시키기 위한 방법 중의 하나로써 시간적, 공간적 지역성이 높은 데이터를 사용자 측에 가까운 곳에 위치한 프록시 서버에 저장하여 서비스한다. 프록시 캐싱을 사용하면 사용자 요청을 항상 원본 객체가 저장되어 있는 서버가 서비스 하는 것이 아니라 프록시가 저장하고 있을 경우에는 원본 서버의 개입 없이 프록시가 대신 서비스를 제공해 줌으로써 네트워크 부하와 사용자 지연시간을 줄일 수 있다. 일반적인 프록시 서버는 프록시의 저장 공간이 부족할 때 LRU 등의 정책을 사용하여 객체 단위로 교체 (replacement) 한다. 하지만, 스트리밍 데이터에는 용량이 매우 크기 때문에 멀티미디어 데이터 전체를 캐싱하게 되면 프록시의 저장 공간이 빨리 소모되기 때문에 많은 종

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 IT 신성장동력핵심기술개발 사업의 일환으로 수행하였음.[2007-S-016-01, 저비용 대규모 글로벌 인터넷 서비스 솔류션 개발]
† 정 회 원 : 한국전자통신연구원 디지털홈연구단 인터넷서버그룹 선임연구원
†† 정 회 원 : 한국전자통신연구원 디지털홈연구단 인터넷서버그룹 책임연구원 (팀장)
††† 정 회 원 : 부산대학교 정보컴퓨터공학부 교수
논문접수 : 2007년 6월 29일, 심사완료 : 2007년 12월 12일

DOI: 10.3745/KIPSTA.2008.15-A.1.53

류의 데이터를 캐싱할 수 없다. 따라서 전통적인 프록시 캐싱에서 사용하는 방법과는 다른 방법의 캐쉬 교체 정책에 관한 연구들이 계속 진행되고 있다. [1]에서는 크게 같은 종류의 단말을 위한 스트리밍 프록시와 다른 종류의 단말을 위한 스트리밍 프록시 방법을 정리하여 설명하였다. 다른 종류의 단말을 위한 스트리밍 프록시 방법은 다양한 단말 및 다양한 네트워크 환경에 적응할 수 있는 방법들인데, 계층형 코딩, FGS(Fine Grained Scalable), MDC(Multiple Description Coding) 등을 사용하는 방법들이 있다. 이러한 방법 이외에 트랜스코딩(transcoding)을 이용하여 사용자 단말 및 네트워크에 적응하는 방법도 연구되고 있다.

트랜스코딩은 코딩된 신호를 다른 신호로 변환하는 것을 의미하며 동영상의 경우, 압축된 동영상을 다른 규격의 동영상으로 변환하거나 비트율, 공간 해상도, 프레임율, 오류 탄력성 등을 조절하는 기술을 의미한다. 현재의 기술로 동시에 실시간 트랜스코딩 할 수 있는 수는 비교적 적으나 보다 효율이 높은 트랜스코딩 기법들이 제안되고 있기 때문에 상용화된 제품들이 등장할 것으로 기대되고 있다. 트랜스코딩은 수행 위치에 따라 서버측, 클라이언트측, 프록시측으로 구분되고 프록시측에서 트랜스코딩을 수행하는 것이 가장 유리한 것으로 알려져 있다[2].

트랜스코딩 프록시도 프록시 서버이기 때문에 프록시의 용량이 부족하게 되면 특정 데이터를 버리고 새로운 데이터를 저장해야 한다. 이러한 교체 정책을 수행할 때 기존의 웹 문서용으로 많이 사용하는 LRU와 같은 방법과 기존의 멀티미디어용 교체 정책을 그대로 사용해서는 효율이 높지 않다. 왜냐하면 웹 프록시에서 다루는 데이터에 비해서 트랜스코딩 프록시에서 다루는 데이터는 멀티미디어 데이터이기 때문에 크기가 매우 크고, 기존의 멀티미디어용 프록시는 다중 버전에 대한 고려가 없기 때문이다.

본 논문에서는 네트워크 효율을 위해 멀티미디어 프록시를 사용하는 시스템에서 트랜스코딩을 수행 할 때 효율적인 프록시 관리 정책을 제안한다. 제안하는 방법은 기존의 멀티미디어 프록시에서 많이 사용하는 세그먼트 기반의 멀티미디어 프록시 기법과 트랜스코딩 프록시에서 다중 버전을 고려한 교체 정책을 결합한 방법이다. 제안하는 방법을 사용하면 사용자 지연 시간, BHR(byte-hit ratio), 그리고 트랜스코딩 해야 할 데이터의 양을 줄일 수 있기 때문에 사용자 측면에서의 장점과 시스템측면에서의 장점을 동시에 가질 수 있다. 본 논문의 구성은 다음과 같다. 2장에서는 기존의 트랜스코딩 프록시 기법에 대해서 설명하고 3장에서는 멀티미디어 데이터를 위한 부분 캐싱을 수행하는 트랜스코딩 프록시에서 발생하는 이벤트를 정의하고, 제안하는 세그먼트 기반의 트랜스코딩 프록시에서의 교체 정책에 대해 설명한다. 4장에서는 시뮬레이션 결과를 살펴보고 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

기존의 멀티미디어 스트리밍을 위한 프록시 서버는 크게 동일한 종류의 클라이언트를 다루는 방법과 다른 종류의 클라이언트까지 고려한 방법이 있다[1]. 전자는 시간간격 이동 캐싱(sliding-interval caching), 서두 캐싱(prefix caching) [3], 서두 캐싱을 확장한 세그먼트 캐싱(Segment Caching) [4], 비율 분할 캐싱(Rate-Split Caching, Video Staging) 등이 있으며, 후자에는 계층 코딩(Layered Coding), FGS, MDC 등이 포함된다.

트랜스코딩을 통한 사용자 단말 적응 방법은 웹 컨텐츠를 대상으로 한 InfoPyramid[5]가 대표적인 예이다. 이 방법은 사용자가 요청한 컨텐츠의 종류와 품질을 선택할 수 있는 방안을 제공하고 있으나, 오프라인 트랜스코딩을 통하여 텍스트, 이미지, 비디오와 오디오 데이터에 대한 다양한 단말에 적응하지만 캐싱 기법은 적용되지 않았다[3]. TransSquid[6]은 서버에서 직접 트랜스코딩을 지원하는 방법을 사용하였고 miss, partial hit, hit라는 이벤트를 정의하였다. 사용자가 요청한 버전이 서버에는 존재하지 않으나 트랜스코딩 가능한 버전이 존재할 경우 partial hit라고 정의하였다.

트랜스코딩과 프록시를 접목시킨 연구로는 FVO/TVO[7], TeC[8]등이 있다. FVO(Full Version Only)는 트랜스코딩 원본 버전만을 캐싱하는 방법이고, TVO(Transcoded Version Only)는 트랜스코딩 한 결과 버전만을 캐싱하는 방법이다. 또한 TeC(Transcoding enable Caching)은 LRU 방법을 사용하여 특정 시간에는 하나의 버전만을 캐싱하는 TeC11, TeC12와 동시에 여러 개의 버전을 캐싱할 수 있는 TeC2를 제안하였다. [9]는 LRU를 사용하고 객체를 반씩 나누어 관리하는 방법을 제안하였다. 즉, 희생 데이터로 선택이 되면 먼저 전체 데이터의 뒤에 있는 반을 버리고, 다음번 희생 데이터로 선택이 되었을 때 앞의 반을 버리는 정책이다.

[2]에서는 웹 데이터를 위한 프록시에서 같은 컨텐츠의 다중 버전 사이의 관계를 트랜스코딩 그래프로 표현하고 트랜스코딩 그래프를 이용하여 이득값(profit)를 계산하고 이득값을 기반으로 교체정책을 수행하였다. 이 방법은 기존의 LRU를 사용하는 방법에 비해 우수한 성능을 보인다. [10]은 [2]의 방법을 멀티미디어 데이터에 적용하기 위해서 평균 참조 길이와 트랜스코딩 길이를 이용하여 이득값 계산 수식을 개선하였다.

[11][12]에서는 트랜스코딩 프록시의 교체 함수로 [2], [10]과 유사한 종합 이득함수(aggregated profit function)를 사용하고 각 버전을 세그먼트별로 관리하는 방법을 제안하였다. 이 방법을 세그먼트를 나눌 때 지수 크기로 증가하는 지수 크기 세그먼트 방법을 사용하였으나, FVO, TVO, TEC 방법만을 비교하였다.

본 논문은 [10]에서의 이득함수와 세그먼트 캐싱 방법을 결합하여 트랜스코딩 프록시에서 효율적인 교체 정책을 제안한다. 제안하는 방법은 교체 함수는 [10]의 VPF(video profit function)를 사용하며, 세그먼트를 나누는 방법을 모든 버전에 대해서 동일한 크기의 세그먼트를 가지도록 하였다.

3. 트랜스코딩 프록시에서 세그먼트 기반 캐쉬 교체 정책

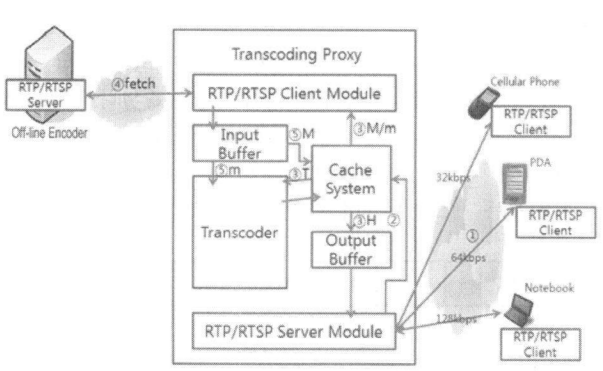
3.1 시스템 구조

트랜스코딩 프록시 사용하는 컨텐츠는 일반 프록시에서 사용하는 컨텐츠와는 다른 특징을 가진다. 즉, 하나의 컨텐츠에 대해서도 다양한 전송율(bitrate)를 가지는 여러 버전(version)이 존재할 수 있는데, 높은 전송율을 가지는 버전을 이용하여 낮은 전송율의 버전을 트랜스코딩 작업으로 만들 수 있다.

일반적인 트랜스코딩 프록시의 구조는 (그림 1)과 같으며, 이 구조는 일반적인 스트리밍 프록시 구조에서 트랜스코딩 모듈이 추가된 모습이다. 즉, RTP/RTSP 서버 모듈이 사용자 단말로부터 서비스 요청을 받고 Cache System에서 요청된 데이터의 저장 여부를 판단하여 저장 되어 있지 않다면 프록시의 RTP/RTSP 클라이언트 모듈을 통해 원본 서버에게 데이터를 요청하여 사용자 단말에게 전달하고, 이미 저장되어 있는 데이터인 경우에는 원본 서버의 개입 없이 프록시가 직접 사용자 단말에게 서비스를 제공한다.

트랜스코딩 프록시는 이러한 경우 외에 사용자는 낮은 전송율을 가지는 버전을 요청했을 때, 프록시에는 해당 버전은 저장 하고 있지 않으나, 높은 전송율을 가지는 버전을 저장하고 있어서 이를 통하여 요청한 버전을 만들 수 있는 경우에 일반 프록시와는 다른 동작을 하게 된다. 이 경우에 cache system은 트랜스코더에게 트랜스코딩 작업을 요청하고 이의 결과를 사용자에게 전송하여 서비스를 제공하게 된다. 이러한 경우를 기존의 프록시에서 사용하는 hit, miss와는 구별되도록 transcoding hit라고 정의한다[8][10].

만일 트랜스코딩 프록시가 버전 단위로 캐싱 하게 되면 hit, miss, transcoding hit의 세 가지 이벤트로 모든 경우를 처리할 수도 있지만, 본 논문에서 사용하는 부분 캐싱을 적용하게 되면 더 많은 이벤트가 필요하며, 이에 따라 트랜스코딩 프록시는 각기 다른 후속 작업을 하게 된다. 이에 대한 자세한 내용은 3.2에서 설명한다.



(그림 1) 트랜스코딩 프록시의 구조

3.2 부분 캐싱을 사용하는 트랜스코딩 프록시에서의 이벤트

프록시에서는 사용자의 요청에 대해 해당 컨텐츠를 이미 캐쉬하고 있는지 유무에 따라 각기 다른 행동을 취하게 된다. 즉, 전통적인 프록시에서 사용자가 요청한 컨텐츠가 프록시에 이미 캐싱되어 있을 때는 캐쉬 히트(cache hit), 그렇지 않을 경우에는 캐쉬 미스(cache miss)라고 정의하고, 캐쉬 히트일 경우에는 원본 서버의 개입 없이 프록시가 직접 사용자에게 서비스를 하고, 캐쉬 미스일 경우에는 원본 서버로부터 데이터를 전송 받고 이를 사용자에게 전달하게 된다. 몇몇 트랜스코딩 프록시 연구에서 기존 프록시의 이벤트 외에 요청한 버전을 생성할 수 있는 버전을 가지고 있는 경우를 트랜스코딩 가능한 히트(Transcodable hit)라고 정의하고 이를 통하여 시스템을 설계 하였다[8][10]. 하지만, 트랜스코딩 프록시 역시 멀티미디어 프록시의 일부분이기 때문에 컨텐츠 전체 단위로 캐싱하는데는 문제점이 있고, 따라서 컨텐츠의 일부분을 캐싱 할 수 있도록 하게 되면, 보다 많은 이벤트가 발생하게 된다. 이러한 이벤트를 정의하기 위해서 본 논문에서는 기본 이벤트(atomic event)로 H, T, M, m를 정의하였다. H(hit)와 M(Miss)는 일반적인 프록시에서 사용하는 이벤트와 동일한 개념이고, T(transcoding hit)는 기존의 트랜스코딩 프록시에서 새로 정의한 이벤트이다. m 이벤트는 캐싱 여부의 관점에서는 M과 동일하나 후속작업이 다른 이벤트이다. 즉, 프록시가 가장 높은 전송율을 가지는 버전을 캐싱하고 있지 않을 때 사용자가 이 버전에 대한 요청을 하면, 프록시는 원본 서버로부터 이 버전을 전송 받아 사용자에게 바로 전달하지만(M), 사용자가 낮은 전송율을 가지는 버전을 요청하게 되면 프록시는 원본 서버로부터 전달받은 높은 전송율의 버전을 트랜스코딩 작업을 거쳐 낮은 전송율의 버전으로 만든 후 사용자에게 전달하게 된다. 후자의 경우가 m 이벤트이다. 즉, M 이벤트의 경우 프록시의 경우 후속작업으로 fetch 작업만 수행하면 되지만, m 이벤트의 경우는 fetch후에 트랜스코딩 작업을 수행해야 한다. 이러한 4가지 기본 이벤트를 이용하여 부분 캐싱하는 트랜스코딩 프록시의 이벤트를 정의 하면 <표 1>과 같이 9개의 이벤트가 존재하게 된다. 즉, 4개의 기본 이벤트와 5개의 복합 이벤트가 존재한다. 표에서 첫번째 필드는 이벤트의 이름을, 두번째 필드는 버전에서의 기본 이벤트의 구성을, 세번째 필드는 후속 작업을 나타낸다.

〈표 1〉 부분 캐싱을 이용하는 트랜스코딩 프록시의 이벤트

Event name	Atomic events in each range	Next action
H	H	No act
M	M	Fetch
m	m	Fetch & Transcoding
T	T	Transcoding
HM	H M	Fetch
Hm	H m	Fetch & Transcoding
HT	H T	Transcoding
HTm	H T m	transcoding/ Fetch & Transcoding
Tm	T m	transcoding/ Fetch & Transcoding

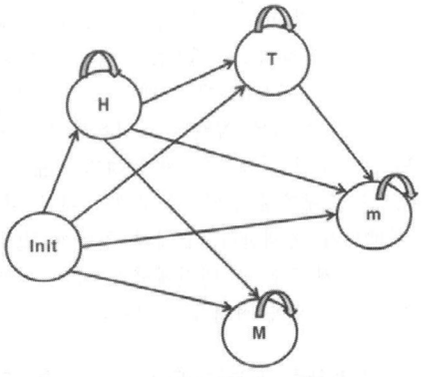
HM과 Hm은 앞부분은 프록시에 캐싱되어 있지만, 뒷부분은 캐싱되어 있지 않은 경우를 나타낸다. 하지만, 요청된 버전이 최상위 전송율인지의 여부에 따라 HM과 Hm으로 구분되며, HM의 경우는 뒷부분을 원본서버로부터 전송받기만 하면 되지만, Hm의 경우는 원본 서버로부터 전송받은 버전을 낮은 전송율로 트랜스코딩 해야 한다.

HT는 앞부분은 캐싱되어 있고, 뒷부분은 캐싱되어 있지 않지만, 이를 만들 수 있는 높은 전송율의 버전을 캐싱하고 있는 경우를 나타낸다. 따라서 후속작업은 뒷부분을 트랜스코딩 하면 된다.

HTm은 요청된 버전이 세 부분으로 나뉘어 지고, 앞부분은 캐싱되어 있고, 가운데 부분은 트랜스코딩 작업을 통해 생성할 수 있으나, 끝부분은 전혀 캐싱되어 있지 않은 경우이다. 따라서 후속작업은 가운데 부분을 트랜스코딩 하고, 끝부분은 원본 서버로부터 전송받아 트랜스코딩 해야 한다.

Tm은 앞부분은 이미 캐싱되어 있는 높은 전송율의 버전을 통해 만들고, 뒷부분은 전혀 캐싱되어 있지 않은 경우로, 후속작업으로 앞부분을 위해서는 트랜스코딩을, 뒷부분을 위해서는 원본서버로부터 높은 전송율의 버전을 가져온 후, 트랜스코딩 해야 한다.

이러한 다양한 이벤트를 그림으로 표현하면 (그림 2)와 같다. 즉, 특정 버전에 대한 요청이 들어오면 트랜스코딩 프록시는 이벤트 전이 그래프 상에서 적절한 이벤트를 생성하게 된다. 이벤트 전이 그래프에서 T➔H, T➔M, M➔H, M➔T, M➔m, m➔H, m➔T, m➔M의 전이는 발생하지 않는데, 이는 캐싱할 때 연속적으로만 저장하는 경우에 한정된다.



(그림 2) 이벤트 전이 그래프

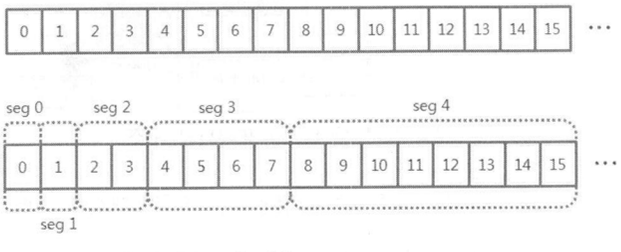
3.3 캐쉬 교체 함수

본 논문에서는 캐쉬 교체 함수로 [10]에서 제안한 VPF를 사용한다. 이 함수들은 [2]에서 제안한 함수인 PF를 비디오 데이터에 적용한 것으로 LRU와 같은 전통적인 프록시 교체 함수와는 달리 같은 컨텐츠에 대해서 존재하는 다양한 버전들 사이의 관계와 트랜스코딩 하게 되는 범위, 해당 버전에 대한 접근율과 해당 버전의 크기 등을 고려 하였다. PF에 기반한 많은 연구들은 트랜스코딩 그래프(transcoding graph)로 불리는 각 버전간의 관계를 나타낸 그래프를 기반으로 한다. 이 그래프는 노드(node)들과 이들을 연결한 링크(link)들로 나타내지고, 노드간 링크가 존재하는 경우는 이 노드간 트랜스코딩을 할 수 있는 관계를 나타낸다. 각 링크는 비용값을 가지는데, 이 비용값은 트랜스코딩에 소요되는 비용을 나타내며 PF 기반의 많은 연구들은 주로 트랜스코딩 지연 시간을 사용하였다[2] [10]. 트랜스코딩 지연시간은 버전 i에서 버전 j로 트랜스코딩 연산을 수행할 때, 버전 j를 스트리밍 할 수 있을 정도의 데이터가 만들어지는데까지 걸리는 시간을 나타낸다. 이러한 트랜스코딩 그래프를 기반으로 특정 시간에 트랜스코딩 프록시에 캐싱되어 있는 상태에 따라 최소비용을 가지는 서브 트랜스코딩 그래프(sub-transcoding graph)를 찾고, 이를 기반으로 어떤 버전으로 사용자가 요청한 버전을 만드는 것이 최소의 비용을 가지는지를 판단하게 된다. 또한, 프록시의 저장공간이 부족하여 원본 서버로부터 가져오는 데이터, 또는 트랜스코딩을 통해 만들어지는 데이터를 저장할 공간이 없을 때, 기존에 저장되어 있는 데이터 중에서 삭제할 대상을 찾기 위해서 PF, VPF 등의 함수를 통해 가장 효율이 적은 컨텐츠의 버전을 선택하게 된다.

3.4 세그먼트 기법

스트리밍 데이터를 캐싱 할 때, 전체 데이터를 대상으로 캐싱하게 되면 프록시 서버의 저장 공간의 문제 때문에 다양한 종류의 스트리밍 서비스를 제공하는데 한계가 있다. 따라서, 기존의 스트리밍 데이터를 위한 프록시에 대한 연구들에서 스트리밍 데이터를 세그먼트로 나누어 관리하는 방법들에 제안되었다. 대표적인 방법이 동일 크기의 세그먼트로 나누는 방법과 지수 크기의 세그먼트로 나누는 방법이 있다. 균등 크기의 세그먼트로 나누는 방법은 모든 컨텐츠에 대해서 같은 크기의 세그먼트로 관리하고, 지수 크기의 세그먼트로 나누는 방법은 특정 컨텐츠의 i번째 세그먼트는 i-1번째 세그먼트의 두 배가 되도록 관리하는 방법이다.

[11][12]의 연구 이외에 트랜스코딩 프록시에 세그먼트 기법을 적용한 예는 없으나, [11][12]의 연구에서는 지수크기의 세그먼트 기법을 적용하고 VPF와 유사한 교체 함수를 사용하여, 이전의 트랜스코딩 프록시 기법과 비교하였다. 본 논문에서는 균등 크기의 세그먼트 기법을 사용하여 트랜스코딩 프록시에 적용한다. 또한 본 논문에서 사용하는 균등 크기 세그먼트 기법은 연속적인 형태로만 저장된다. 즉, 저장된 세그먼트들 간에 의미 없는 세그먼트(garbage segment)는 존재하지 않고, 첫 세그먼트부터 저장된 마지막 세그먼트까지 모든 세그먼트들이 의미 있는 세그먼트이다.



(그림 3) 균등 및 지수 크기의 세그먼트 기법

3.5 세그먼트에 기반한 교체 정책

본 절에서는 기존의 VPF 캐쉬 교체 함수를 사용한 트랜스코딩 프록시에서 프록시 교체 정책에 대하여 설명한다. 교체는 트랜스코딩이나 페칭(fetching)을 통하여 새로운 세그먼트를 프록시에 저장할 때 프록시의 공간이 부족한 경우에 수행한다. 새로운 세그먼트를 저장하기 위한 공간이 부족하면 VPF의 값이 가장 작은 버전을 선택하고 선택된 버전의 가장 마지막 세그먼트부터 삭제한다. 가장 마지막 세그먼트부터 삭제하는 이유는 사용자가 특정 서비스를 처음부터 받다가 임의의 위치에서 종료할 수 있다고 가정하게 되면, 앞부분이 뒷부분보다 접근 빈도가 높기 때문이다. 캐쉬하고자 하는 데이터를 위한 충분한 공간이 만들어 질 때까지 이러한 과정을 반복하게 된다. 이러한 방식은 계층형 코딩을 사용하는 프록시 관리에서도 이미 사용되었다[13].

〈표 2〉 세그먼트 기반 트랜스코딩 프록시에서 교체 정책

```
1:  replacement(cid, ver, size)
2:  {
3:      while(!is_room(cid, ver, size))  {
4:              vic_cid, vic_ver ← find_profit_victim
5:              delete_last_seg(vic_cid, vic_ver)
6:      }
7:      update_profit(cid)
8:  }
9:  is_room(cid, ver, size)
10: {
11:     bef_seg ← get current segment no.
12:     aft_seg ← get after segment no.
13:     if(bef_seg == aft_seg)
14:             return TRUE
15:     else {
16:             if(capacity+seg_size*(aft_seg-bef_seg)>capacity)
17:                     return FALSE
18:             else
19:                     return TRUE
20:     }
21: }
```

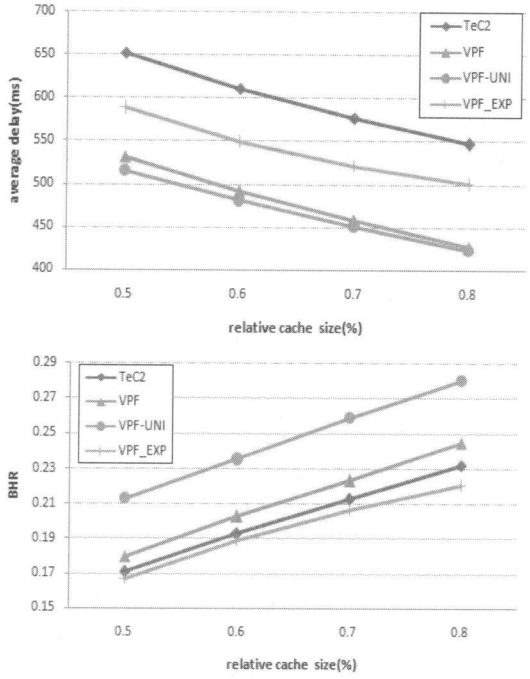
4. 성능 평가

본 논문에서는 균등 크기의 세그먼트와 VPF를 사용한 방법의 성능을 측정하기 위해 [10]에서와 유사한 시뮬레이션 환경을 이용하였다. 사용자 단말은 5개의 서로 다른 종류로 가정하였고 각각 15%, 20%, 30%, 20%, 15%의 비율로 서비스를 요청하며, 이때 단말은 각각 512, 256, 128, 64, 32Kbps의 상영속도(bitrate)를 지원한다. 사용자가 요청한 버전이 프록시에 저장되어 있지 않다면 이벤트의 종류에 따라 페칭 또는 트랜스코딩 연산을 통하여 사용자가 원하는 데이터를 전송한다. 트랜스코딩 지연시간은 트랜스코딩 연산이 발생할 때 연산이 시작된 이후에 처음으로 트랜스코딩 결과가 나오기 시작하기까지의 시간이며, H.264컨텐츠의 실측치를 이용하여 값을 설정하였다. 또한 각 객체의 인기도는 Zipf 분포를 따르며 skew factor는 0.47로 가정하였다. 컨텐츠는 30초에서 12분 사이의 길이를 가지며, 시뮬레이션은 100,000개의 요청을 400 시간 동안 진행하도록 하였다. 원본서버로부터 트랜스코딩 프록시까지 데이터를 가져올 때의 지연시간은 지수분포를 따르며(μ = 1.5), 사용자 모두가 컨텐츠의 끝까지 보는 것은 아니기 때문에 지수분포로 종료 시간을 설정하였다(μ = 0.5). 프록시 서버의 저장공간은 128Kbps 컨텐츠의 전체 크기 합의 50~80%의 크기를 사용하였다. 또한 세그먼트의 크기는 64Kbyte, 트랜스코딩 속도는 512Kbps로 설정하였다.

본 논문에서 제안하는 방법과 성능 비교를 위해 트랜스코딩 시스템인TeC2[7], VPF[10]를 사용하였다. TeC2는 교체함수로 LRU를 사용하고 버전 전체 단위로 캐쉉 하는 방법이다. VPF는 교체함수로 VPF를 사용하고 캐쉉 단위를 평균 접근 시간을 고려하여 파일을 두개의 단위로 나누어 관리한다. VPF_EXP[11][12]와 VPF_UNI는 교체함수로 VPF를 사용하되, 세그먼트를 나누는 방법에 있어서 지수크기와 균등크기를 나타낸다.

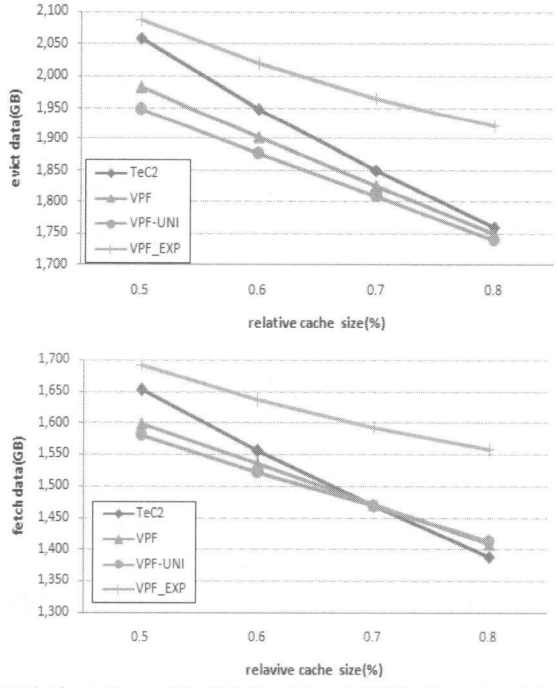
본 논문에서는 성능 분석의 요소는 일반적인 프록시 시스템에서 많이 사용하는 평균지연시간, BHR(byte-hit ratio)와 트랜스코딩 프록시에서 실제 트랜스코딩 해야 하는 데이터 양을 사용하였다. 평균 지연 시간은 사용자가 요청한 이후에 실제 서비스를 받기까지의 지연시간을 나타내며, BHR는 사용자가 요청한 버전을 트랜스코딩 프록시가 저장하고 있는 비율을 나타낸다. 또한 트랜스코딩 데이터 양은 트랜스코딩 작업을 수행해야 하는 데이터 양을 나타낸다. 트랜스코딩 작업은 CPU 자원을 많이 소모 하기 때문에[14] 트랜스코딩 작업을 적게 할수록 시스템 자원을 효율적으로 사용할 수 있다.

(그림 4~6)은 트랜스코딩 프록시 서버 용량에 따른 성능을 나타내는 그래프이다. 먼저 (그림 4)는 일반적인 프록시 서버와 같이 평균 지연시간과 BHR를 나타낸다. 그림과 같이 VPF_UNI는 평균 지연시간이 TeC2에 비해서 26~29%, VPF에 비해서 1~3%, VPF_EXP에 비해서 14~18% 더 우수하다. 또한 BHR측면에서는 TeC2에 비해서 17~20%, VPF에 비해서 13~16%, VPF_EXP에 비해서 20~22% 더 우수하다. 그림과 같이 TeC2의 성능이 가장 좋지 않은데, 이는 TeC2의 경우 LRU를 사용하며 LRU는 사용자의 요청 빈도수나 데이터의 크기, 지연 시간 등을 고려하지 않기 때문이다. 또한, VPF_EXP의 경우는 TeC2의 경우보다는 성능이 우수하지만 VPF나 VPF_UNI에 비해서는 성능이 떨어짐을 알 수 있다. [11]에서 저자들은 TeC2에 비해 지연시간은 우수하나 BHR의 경우 거의 유사하다고 밝힌 바 있다. VPF_UNI의 경우는 VPF에 비해 지연시간은 거의 유사하지만 BHR 측면에서는 우수하다. 이는 각 버전을 작게 나누어 관리하기 때문에 프록시 서버에 더 많은 종류의 버전을 저장할 수 있기 때문이다.

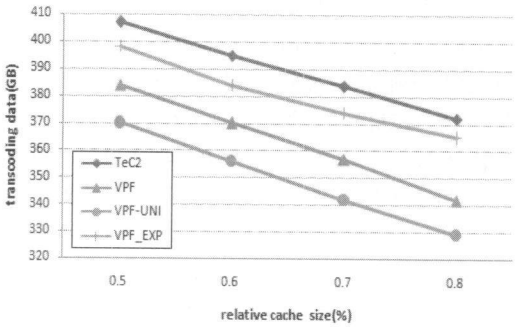


(그림 4) 프록시 서버 용량에 따른 사용자 지연시간(위) 및 BHR(아래)

(그림 5)와 (그림 6)은 트랜스코딩 프록시에서 새로운 데이터의 저장을 위해서 버려지는 데이터의 양(그림 5의 위 그래프)와 원본 서버로부터 가져오는 데이터 양(그림 5의 아래 그래프), 그리고 트랜스코딩 하는 데이터의 양(그림 6)을 나타낸다. 트랜스코딩 프록시에서 새로운 데이터가 캐싱되는 경우는 원본 서버로부터 최상위 전송속도를 가지는 버전을 가져오는 경우와 상위 전송속도를 가지는 버전을 이



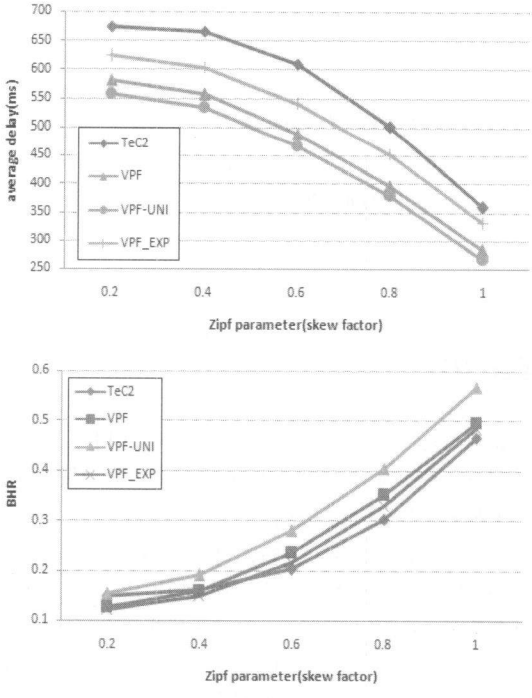
(그림 5) 프록시 서버 용량에 따른 evict(위) 및 fetch 데이터 양(아래)



(그림 6) 프록시 서버 용량에 따른 transcoding 데이터 양

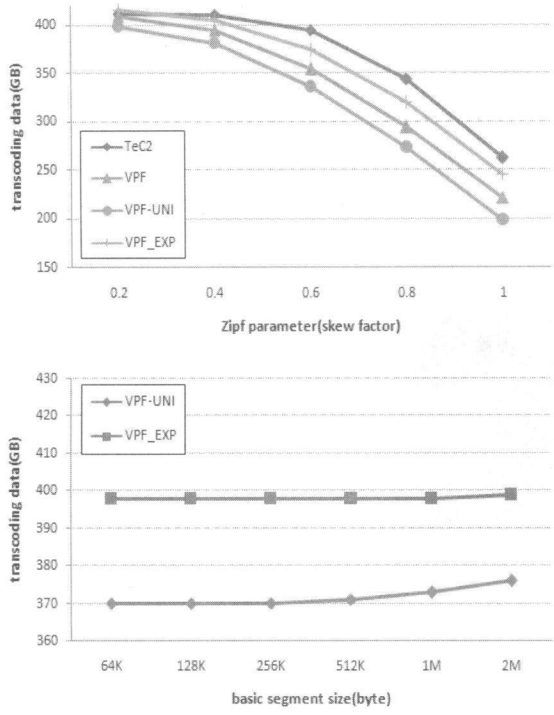
용하여 사용자가 요청한 버전을 생성해 내는 경우이다. 따라서 버려지는 데이터 양(evict)은 이 두 가지 경우의 데이터 양의 합과 같다. 즉, (그림 5)의 위에 있는 각각의 데이터 양(evict data)은 (그림 5)의 아래 데이터 양(fetch data)과 (그림 6)의 데이터 양(transcoding data)의 합과 일치한다. 그림과 같이 VPF_UNI의 경우, 다른 방법들과 비교할 때, 적은 양의 데이터를 버리고 적은 양의 데이터를 트랜스코딩 하기 때문에 시스템 자원을 좀 더 효율적으로 사용할 수 있다.

(그림 7)과 (그림 8)의 위 그래프는 Zipf 분포의 변수인 α 값의 변화에 따른 성능 변화를 나타낸다. Zipf 분포의 α는 사용자의 접근 패턴을 나타내는 변수이다. 이 값이 클수록 사용자들은 특정 컨텐츠의 특정 버전에 대한 접근 비율이 편향되게 집중되고, 이 값이 작을수록 집중도가 떨어진다. 전체적으로 볼 때 VPF_UNI가 가장 우수한 성능을 가지며, TeC2의 성능이 가장 떨어짐을 알 수 있다.

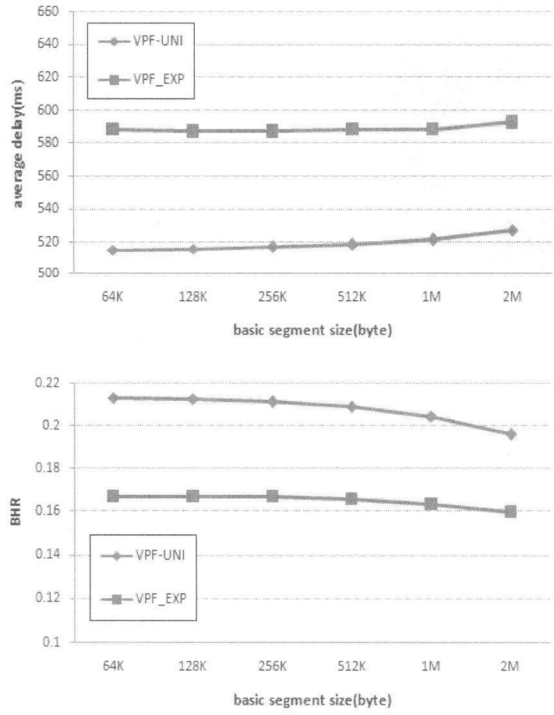


(그림 7) Skew factor 변화에 따른 사용자 지연시간(위) 및 BHR(아래)

(그림 8)의 아래 그래프와 (그림 9)는 기본 세그먼트 크기의 변화에 따른 VPF_UNI와 VPF_EXP의 성능을 보여준다. 기본 세그먼트의 크기는 VPF_UNI의 경우, 모든 세그먼트의 크기와 동일하며, VPF_EXP의 경우 첫번째 세그먼트의 크기이다. VPF_EXP의 경우, i 번째 세그먼트의 크기는 i-1번째 세그먼트의 크기의 두 배가 된다. 그림과 같이 기본 세그먼트의 크기가 커 질수록 트랜스코딩 프록시의 성능이 떨어지나 VPF_UNI는 VPF_EXP에 비해서 우수한 성능을 갖게 된다.



(그림 8) Skew factor 변화(위) 및 기본 세그먼트 크기 변화(아래)에 따른 트랜스코딩 데이터 양



(그림 9) 세그먼트 크기 변화에 따른 evict 및 fetch 데이터 양

VPF_UNI의 경우, 다른 방법들에 비해서 교체 수는 월등히 많다. 이는 각 버전을 작은 단위로 관리하기 때문인데, 이에 대한 연산은 트랜스코딩 연산에 비하면 미미한 성능 저하로 판단된다.

5. 결론 및 향후 연구

유비쿼터스 환경에서는 사용자 단말과 단말이 연결되어 있는 네트워크 상태가 매우 다양하기 때문에 같은 컨텐츠에 대한 요청에 대해서도 사용자 적응적으로 서비스를 제공해 줄 수 있는 방법이 필요하다. 또한 많은 사용자가 동일한 컨텐츠에 대해 집중적으로 서비스 요청할 수 있기 때문에 사용자 지연 시간 감소와 네트워크 대역폭 절약 등의 이유로 프록시 서버를 많이 사용하고 있다. 하지만 멀티미디어 데이터는 기존의 웹 문서에서 많이 사용하는 데이터에 비해 크기가 매우 크기 때문에 기존의 웹 문서를 위한 프록시를 그대로 멀티미디어 스트리밍 데이터를 위한 프록시에 적용하는데는 많은 한계가 있다. 본 논문에서는 사용자 적응 서비스를 제공하기 위해 트랜스코딩을 프록시에서 적용할 때의 효과적인 교체 정책을 제안하였다. 제안하는 방법은 기존의 멀티미디어 프록시에서 많이 사용하고 있는 세그먼트 기반의 교체 정책과 트랜스코딩 프록시에서 다중 버전을 캐싱할 때 상호간의 관계를 이용한 방법을 결합하였다. 제안하는 방법을 위해 멀티미디어 데이터를 위한 부분 캐싱에서의 이벤트들을 정의하고, 이러한 이벤트에 따라 적절히 캐싱 및 교체를 수행할 수 있는 방법이다. 실험결과 사용자 지연시간, BHR, 트랜스코딩 해야 할 데이터의 양 등에서 다른 방법들에 비해 우수한 성능을 보였다. 하지만, 세그먼트 기반으로 교체를 수행하기 때문에 교체 횟수는 다른 방법들에 비해 단점을 가지지만 트랜스코딩 연산에 드는 자원 소모에 비해서는 미미하기 때문에 트랜스코딩 프록시를 위한 정책으로 적절하게 보인다.

참 고 문 헌

[1] Jiangchuan Liu, Jianliang Xu, "Proxy Caching for Media Streaming Over the Internet", IEEE Communications Magazine, Aug., 2004.

[2] Cheng-Yue Chang, Ming-Syan Chen, "On Exploring Aggregate Effect for Efficient Cache Replacement in Transcoding Proxies", IEEE Transactions on Parallel and Distributed Systems, Vol.14, No.6, Jun., 2003.

[3] S.Sen, J.Rexford, D.Towsley, "Proxy prefix caching for multimedia streams", in Proc. IEEE INFOCOM'99, New York, NY, Mar., 1999.

[4] Songqing Chen, Bo Shen, Susie Wee, Xiaodong Zhang, "Segment-Based Streaming Media Proxy : Modeling and Optimization", IEEE Transactions on Multimedia, Vol.8, No.2, Apr., 2006.

[5] J.R. Smith, R. Mohan, and C-S. Li, "Scalable multimedia delivery for pervasive computing", Proc. ACM Multimedia '99, Orlando, Florida, Oct., 1999.

[6] A. Maheshwari, A. Sharma, K. Ramamrithan, P. Shenoy, "TransSquid: Transcoding and caching proxy for heterogeneous e-commerce environments", Proceeding of IEEE RIDE 2002, San Jose, CA, USA, 2002.2.

[7] Xueyan Tang, Fan Zhang, Samuel T. Chanson, "Streaming Media Caching Algorithms for Transcoding Proxies", Proceedings of the International Conference on Parallel Processing(ICPP02), 2002.

[8] Bo Shen, Sung-Ju Lee, Sujoy Basu, "Caching Strategies in Transcoding-Enabled Proxy Systems for Streaming Media Distribution Networks", IEEE Transactions on Multimedia, Vol.6, No.2, Apr., 2004.

[9] YongJu Lee, YuHyeon Bak, OkGee Min, HagYoung Kim, CheolHoon Lee, "The PT-2 Caching Algorithm in the Transcoding Proxy Cluster to Facilitate Adaptive Content Delivery", International Workshop on Multimedia Content Analysis and Mining 2007 (MCAM01), WeiHai, China, 2007.6.

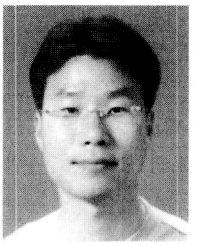
[10] Chi-Feng Kao, Chung-Nam Lee, "Aggregrate Profit-Based Caching Replacement Algorithms for Streaming Media Transcoding Proxy Systems", IEEE Transactions on Multimedia, Vol.9, No.2, Feb., 2007.

[11] Kuei-Chung Chang, Tien-Fu Chen, "Efficient segment-based video transcoding proxy for mobile multimedia services", Journal of Systems Architecture, ELSEVIER, Nov., 2007.

[12] Kuei-Chung Chang, Ren-Yo Wu, Tien-Fu Chen, "Efficient segment-based video transcoding proxy for mobile multimedia services", Proceedings of International Conference of Multimedia and Expo(ICME 2005), Jul., 2005.

[13] R.Rejaie, H.Yu, M.Handley, D.estrin, "*Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet*", in Proc. IEEE INFOCOM'00, Tel Aviv, Israel, Mar., 2000.

[14] Dongyu Liu, Songqing Chen, Bo Shen, "AMTrac: Adaptive Meta-caching for Transcoding", Proceedings of ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2006), Newport, Rhode Island, May, 22-23, 2006.



박 유 현

e-mail : bakyh@etri.re.kr

1996년 부산대학교 전자계산학과(학사)

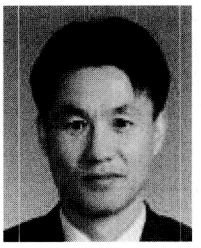
1998년 부산대학교 대학원 전자계산학과 (이학석사)

2000년 부산대학교 대학원 전자계산학과 박사수료

2000년 한국국방연구원(KIDA) 자원관리연구부 연구원

2001년~현 재 한국전자통신연구원 디지털홈연구단 인터넷서버그룹 선임연구원

관심분야 : 트랜스코딩 프록시, 대규모 클러스터 시스템, 시스템 소프트웨어, VoD



김 학 영

e-mail : h0kim@etri.re.kr

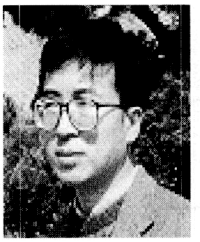
1983년 경북대학교 전자공학과 전자계산 전공 학사

1985년 경북대학교 대학원 전자공학과 전자계산전공 공학석사

2003년 충남대학교 대학원 컴퓨터공학과 공학박사

1988년~현 재 한국전자통신연구원 디지털홈연구단 인터넷서버그룹 책임연구원(팀장)

관심분야 : 대규모 클러스터, 인터넷 서버, 컨텐츠 스트리밍, 컨텐츠 분배, 시스템소프트웨어, 분산파일시스템, GRID



김 경 석

e-mail : gimgs@asadal.cs.pusan.ac.kr

1977년 서울대학교 무역학과(경제학사)

1979년 서울대학교 전자계산학과(이학석사)

1988년 일리노이 주립대(어바나-샴페인) 전자계산학 박사

1988년~1992년 미국 노스다코타 주립대학교 전자계산학과 조교수

1992년~현 재 부산대학교 정보컴퓨터공학부 교수

1993년~현 재 국내 문자 코드 전문위원회 위원장(Korea JTC1/SC2)

1997년~현 재 국내 문헌 정보 및 로마자 전문 위원회(Korea TC46) 위원

1999년~현 재 국내 인터넷 네임 위원회 및 한글 네임 WG 위원

관심분야 : 데이터베이스, 멀티미디어, 한글/한말 정보처리, 인터넷 컴퓨팅