이동환경에서 연속미디어 서비스를 위한 협력적인 프록시 캐싱

이 승 원[†] · 이 화 세[††] · 박 성 호[†††] · 정 기 동[††††]

본 연구는 이동환경에서 연속 미디어 객체에 대한 사용자 요구들에 대하여 효율적으로 자원을 관리하기 위하여, 사용자의 이동성 정보를 반영하는 협력적 프록시 캐싱 정책을 제안한다. 제안된 정책은 담당영역 내에서 사용자의 요구들을 참조함으로서 캐싱하는 기존의 프록시 캐싱 정책과는 다르게 인접한 영역의 많은 사용자 요구 정보들을 이용하여 캐싱 객체를 선택함으로써 이동환경에서 발생하는 사용자들의 랜덤 액세스 요구들을 충분히 반영할 수 있도록 한다. 그리고 제안된 캐싱 정책을 위해서 각 프록시에서 발생한 요구 정보의 가중치와 재생량을 기반으로 하는 재배치 기법을 제안한다. 그래서 객체단위와 세그먼트 단위로 재배치 기법을 수행하여 결과를 평가하고 캐싱 정책의 성능을 분석하였다. 그 결과, 이동환경에서 적절한 사용자의 이동성 비율이 협력적 캐싱의 성능에 영향을 미치는 중요한 요인이 된다는 것을 알았다.

A Cooperative Proxy Caching for Continuous Media Services in Mobile Environments

Seung-Won Lee[†] · Hwa-Sei Lee[††] · Seong-Ho Park[†††] · Ki-Dong Chung[††††]

ABSTRACT

This paper proposes a user's mobility based cooperative proxy caching policy for effective resource management of continuous media objects in mobile environments. This policy is different from the existing proxy caching policies in terms of how to exploit users' mobility. In other words, existing caching policies work based on the information about objects by referring to user's requests within a specified domain whereas the proposed caching policy runs by utilizing a number of user's requests across several domains. So, the proposed policy is applicable to random requests in mobile environments Moreover, we also propose a replacement policy based on weights and playback time. To check the efficiency of the proposed caching policy, the proposed replacement policy is run with different size of caching unit : object or segment. The result of performance analyze tells what a ratio of user's mobility is are major factors for the efficient operation of the cooperative caching.

키워드 : 이동환경(Mobile Environment), 프록시 캐싱(Proxy Caching), 연속미디어(Continuous Media)

1. 서 론

최근 인터넷에서 스트리밍 서비스는 데이터의 속성상 많은 양을 처리하고 전송하기 위하여 시스템의 높은 계산 능력과 네트워크의 고속 전송 능력을 요구하게 된다. 그러나 멀티미디어 데이터를 위한 스트리밍 서비스의 지원은 코어 네트워크의 교통량에 심각한 문제를 발생시킨다. 그래서 네트워크상의 교통량과 서버의 오버헤드를 줄여 전체 시스템의 성능을 향상시키고 네트워크의 자원을 효율적으로 관리하는 방법으로서 프록시 서버를 사용한다.

이동환경을 지원하는 프록시는 원격의 중앙서버 자료를 저장하는 기능, 이동 노드의 이동성에 관한 정보, 이동 노드의 요구를 서비스하기 위해 제공된 네트워크의 대역폭, 이동 노드의 사용가능한 자원 등을 고려하여야 한다. 이에 대한 연구로, 프록시가 서버의 데이터를 변환하여 전달해주는 프록시 트랜스 코딩 기법에 대한 연구들과 이동 노드의 요구에 대한 지연시간을 줄이기 위한 버퍼링, 선반입(Prefetching) 등에 관한 연구들이 주를 이루고 있다[1-4]. 그리고 [5-7]는 이동환경에서 단말 노드나 하나의 프록시에서 재배치 정책에 대한 연구들로 단말 노드가 여러 셀로 이동하는 경우에 발생하는 문제는 고려하고 있지 않으며, 특히 연속미디어 서비스를 위해서 사용자의 이동성 정보를 반영한 캐싱 정책에 대한 연구들은 아니다. 이동환경에서의 프록시 캐싱은 유선망에서 캐싱 기법과 마찬가지로 객체(Object)의 서로 다른 접근 빈도인 인기도와 같은 미디어 데이터의 특성을 고려하여야 하며 사용자들의 이동에 따른 이동성 정보를 적응성 있게 적용할 수 있는 캐싱 구조와 캐싱 정책의 연구가 필요하다.

† 준 회 원 : 부산대학교 대학원 전자계산학과
†† 정 회 원 : 밀양대학교 컴퓨터공학부 교수
††† 정 회 원 : 부산대학교 정보전산원 교수
†††† 종신회원 : 부산대학교 전자전기정보컴퓨터공학부 교수
논문접수 : 2004년 7월 27일, 심사완료 : 2004년 9월 17일

본 연구에서는 이동환경에서 사용자의 이동에 따른 객체의 요구 정보를 프록시들 간에 공유하여 캐싱에 반영하는 협력적 프록시 캐싱(Cooperative Proxy Caching) 정책을 제안한다. 제안된 정책은 하나의 프록시의 위치를 기준으로 인접한 주위 프록시들에서 발생한 사용자들의 요구 정보를 서로 공유하여 캐싱 정책에 반영하는 구조이다. 그래서 프록시는 현재 사용자들의 서비스 요구에 대한 정보뿐만 아니라 향후 이동해 올 사용자들의 서비스 요구에 대한 정보도 반영한다. 그리고 객체단위와 세그먼트(Segment) 단위의 재배치(Replacement) 기법으로 캐싱 정책을 수행하여 사용자의 이동성 비율에 따른 성능을 분석하였다.

본 연구의 구성은 다음과 같다. 2장에서는 이동환경에서 연속미디어 서비스와 네트워크 캐싱, 캐쉬 재배치 기법에 대한 관련연구를 살펴보고, 3장에서는 이동환경에서 연속미디어 서비스를 위한 협력적 프록시 캐싱 구조와 재배치 기법에 대해서 설명한다. 그리고 4장에서는 성능 평가와 분석을 수행하고 5장에서는 결론 및 향후과제에 대해서 기술한다.

2. 관련 연구

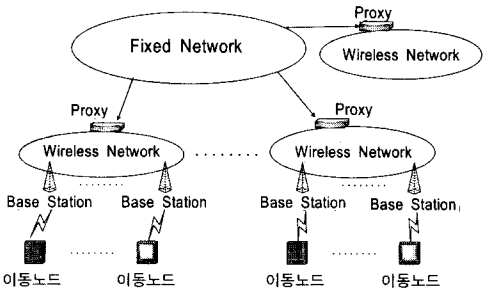
2.1 이동환경에서 연속미디어 서비스

무선 접속 방식의 패킷 데이터 서비스를 기반으로 하는 이동환경은 케이블이나 초고속 인터넷 등과 같은 유선망 환경과는 다르게 연속미디어 서비스를 위해서는 고려해야 할 많은 문제점들이 존재한다. 연속미디어 서비스는 특성상 많은 양의 멀티미디어 데이터를 처리하기 위해서 시스템의 처리능력이 뛰어나야 하고 또한 처리된 미디어 데이터를 전송하기 위해서는 네트워크 상의 전송 대역폭이 커야 한다. 그래서 이동환경에서 미디어 서비스는 유선망에서와 마찬가지로 미디어 데이터의 특성과 사용자의 이동성에 따른 지터나 끊김 없는 서비스, 빠른 응답시간 등을 고려하여야 한다.

2.2 이동환경에서 네트워크 캐싱

이동환경에서는 사용자가 이동하는 동안에도 데이터 전송과 필요한 자원에 대한 접근이 가능해야 한다. 그리고 네트워크 전체의 오버헤드를 줄이고 성능을 향상시키면서 이러한 기능을 지원하기 위해서는 이동 노드에 필요한 정보 또는 객체들을 캐싱하는 것이 필요하다. (그림 1)과 같이 고정망과 연결되는 무선 접근 망의 게이트웨이나 베이스 스테이션에 무선 인터넷 멀티미디어 프록시를 두어 네트워크 자원 비용의 감소, 미디어의 품질 향상, 단말 노드의 초기 지연시간 감소와 같은 이점을 가질 수 있다[8].

그러나 이동환경에서 사용자의 이동성 정보를 반영하는 네트워크 캐싱은 유선망 환경에서와는 다르게 사용자의 작업 패턴이나 공간적 시간적 이동성, 캐쉬 관리 기법, 적응성 기법, 이동성관리 기법, 효율적인 경로 설정, 낮은 전송 속도 등에 대한 고려가 필요하다. 그래서 주로 인터넷상의 웹 서버 또는 HTTP 프록시 서버와 이동 노드(클라이언트)사이의 MSS에 주로 프록시 서버를 두어 이동 노드에 적합한 형태로 객체를 전달한다. 이러한 연구들은 웹 서버나 기존 프록시 캐쉬의 부하를 완화시키는 목적으로 하는 트랜스 코딩 프록시에 관한 것들이 많다[1]. 그러나 이것은 이동환경에서의 프록시 정책이나 재배치 기법에 대한 연구는 고려하고 있지 않으며 또한 사용자들의 이동성 정보를 제대로 반영하지 않는다.



(그림 1) 고정 망과 무선 접근 망으로 구성된 이동환경

2.3 이동환경에서 캐쉬 재배치 기법

다른 관점의 연구로 [5]는 사용자의 로밍(Roaming)을 예측하여 가장 이동가능성이 높은 셀을 선택하여 현재의 캐쉬 내용을 재배치(Relocation)하는 방법을 제공하고 있다. [6]은 단말 노드의 이동성이 발생할 때, 분산으로 연결되어 서로 협력하는 캐쉬들 간에 연결된 동적인 위상을 면밀히 추적하여 객체가 위치한 가까운 연결부분의 캐쉬와 단말 노드를 연결시킨다. 그래서 QoS를 향상시키고 오버헤드를 줄여 서버와 네트워크 자원의 소모를 줄일 수 있는 방법을 제안하고 있다. [7]은 데이터 검색 지연, 데이터 크기, 액세스 확률, 갱신 빈도를 고려한 SAIU(Stretch : Access-rate Inverse Update-frequency)라는 이득(Gain) 함수를 사용한 캐쉬 재배치 정책을 제안 하였다. [8]은 무선망 환경에서 멀티미디어 프록시를 베이스 스테이션이나 게이트웨이에 두고 비디오 스트리밍을 지원하기 위한 Cost-Based 재배치 기법을 제안하였다. 이 기법은 프록시에서 네트워크 대역폭을 향상시키고 초기 지연 시간을 줄이며 미디어의 품질을 향상시키기 위해서 네트워크 비용, 지연(Latency) 비용, 그리고 미디어 왜곡(Distort)비용을 결합한 비용 측도(Cost Metric)를 재배치 측도로 사용하였다.

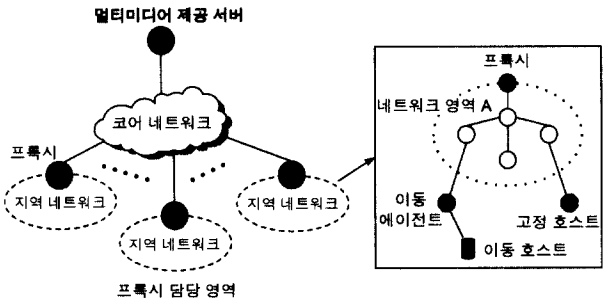
3. 이동환경에서 협력적 프록시 캐싱

본 연구는 이동환경에서 프록시가 자신의 담당 영역에서 발생하는 사용자 객체 요구에 관한 정보뿐만 아니라 이웃하

는 영역에서도 발생하는 잠재적인 사용자들의 객체 요구에 관한 정보도 고려하여 캐싱 정책을 수행하도록 하는 협력적 프록시 캐싱 구조를 성능 측정을 위한 모델로 제안한다. 협력적 프록시 캐싱 구조에서 프록시 간에 서로 협력하는 클러스터 프록시와 클러스터 그룹 내의 프록시들 간의 가중치를 정의한다. 그리고 클러스터 그룹 내의 각 프록시들에서 발생하는 사용자 요구들의 가중치를 부여하고, 이 가중치와 객체의 데이터 재생량(Amount of Playback)을 이용하여 **가중치를 적용한 객체 데이터의 재생량**[1]을 계산한다. 그래서 계산된 값을 협력적 프록시 캐싱을 위한 재배치 척도로 사용한다. 그리고 이러한 재배치 척도 값을 기준으로 하는 재배치 객체의 선택 기법에 대하여 제안한다.

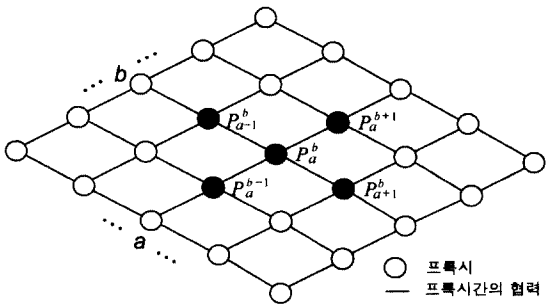
3.1 협력적 프록시 캐싱의 구조

(그림 2)는 본 연구에서 고려하는 네트워크의 전체 구조로 기존 유선망의 코어 네트워크와 무선망을 지원하는 지역 네트워크로 구성된다. 각 프록시는 한 영역 내에서 이동 호스트와 고정 호스트가 혼재되어 있는 지역 네트워크를 담당한다.



(그림 2) 전체 네트워크 구조

협력적 프록시 캐싱 구조에서 협력관계에 있는 각 프록시들의 논리적 연결은 (그림 3)과 같은 $n \times n$ 구조로 간주하며, 이들 간의 관계는 정의 1과 같다. 그리고 수식에서 사용되는 기호는 <표 1>과 같이 정의한다.



(그림 3) 협력적 프록시 캐싱의 구조

1) 본 연구에서는 **'가중치_적용_재생량'**이라고 간단하게 표현한다.

〈표 1〉 수식에 사용되는 기호

기 호	설 명
P_a^b	임의의 한 프록시, $1 \le a,\ b \le n$
$C(P_a^b)$	P_a^b의 인접한 프록시들의 집합
$CP(P_a^b)$	P_a^b를 포함한 인접한 프록시들의 전체의 집합

[정의 1] 협력적 프록시 캐쉬 구조

(그림 3)과 같이 프록시들 간에 서로 협력하는 전체 연결 구조는 다음과 같다.

① $$\exists P_a^b,\ \forall a, b \in Z,\ 1 \le a,\ b \le n \qquad (1)$$

② P_a^b를 중심으로 바로 인접한 프록시는 P_{a-1}^b, P_{a+1}^b, P_a^{b-1}, P_a^{b+1}가 되어

$$C(P_a^b) = \{P_{a-p}^{b-q} \mid |p|+|q|=1,\ \forall p, q \in Z, -1 \le p, q \le 1\} \qquad (2)$$

가 된다.

3.2 클러스터 프록시

논리적으로 연결된 프록시 구조에서 공간적으로 인접한 위치에 있는 프록시들을 같은 클러스터 프록시에 속한다고 본다. (그림 3)은 프록시 P_a^b를 중심으로 인접하는 프록시들과 서로 협력하는 프록시 P_a^b의 클러스터 구조를 나타낸다.

[정의 2] 프록시 P_a^b의 클러스터

식 (2)에서 P_a^b의 인접한 프록시들의 집합이 $C(P_a^b)$이므로, P_a^b의 프록시의 클러스터는 자신을 포함하여 바로 인접한 프록시들로

$$CP(P_a^b) = \{P_a^b\} \cup \{P_a^{b-1}, P_a^{b+1}, P_{a-1}^b, P_{a+1}^b\} \qquad (3)$$

로 정의한다.

이동환경에서는 한 프록시가 담당하는 영역외의 사용자들도 영역내로 이동하여 영역내의 사용자가 될 가능성이 있다. 그래서 클러스터 프록시 내의 각 프록시는 자신의 영역 내의 객체 요구에 관한 정보를 클러스터 내의 다른 프록시에게 전파하여 자신의 캐싱 객체 요구에 관한 정보를 다른 프록시도 유지하도록 한다. 따라서 클러스터 프록시 내의 모든 프록시들은 서로 캐싱 객체 요구에 관한 정보를 유지 관리한다. 그리고 인접 영역의 사용자 요구는 해당 프록시에 대한 서비스 요청이 아닌 단지 가능성이 있는 잠재적인 서비스 요청이기 때문에 가중치를 차등적으로 적용한다.

[정의 3] 프록시에서 발생하는 요구들의 가중치 값

① $C(P_a^b)$에 속하는 프록시들의 개수를 $Num(C(P_a^b))$라 하면, 프록시 P_a^b의 가중치는 1로 두고, 인접한 프록시들의 집합 $C(P_a^b)$에 속하는 각 프록시 P_{a-p}^{b-q}의 가중치는

$$w(P_{a-p}^{b-q}) = \frac{1}{Num(C(P_a^b))},\ where\ \forall p, q \in Z, -1 \le p, q \le 1 \quad (4)$$

라 한다.

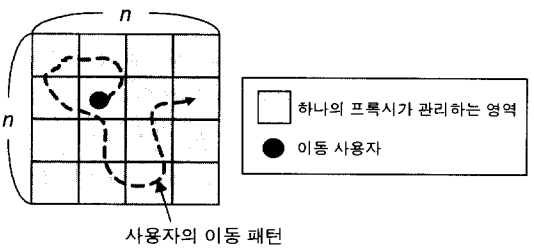
② 각 프록시에서 발생하는 사용자 요구들의 가중치는 담당 프록시의 가중치를 따른다.

3.3 이동 사용자 모델

사용자들이 이동환경에서 여러 영역 사이를 빠르게 혹은 느리게 이동하면서 연속미디어 서버에 대한 여러 가지 요구들을 발생시킬 때, 사용자들의 이동 패턴에 대한 모델링을 위해서 고려해야할 여러 가지 요인들을 전체적으로 반영하기란 현실적으로 매우 어렵다. 본 연구에서는 사용자의 이동 영역과 이동 방향을 정의된 범위 내에서 랜덤하게 선택하고 과거의 사용자 위치와 현재의 위치는 독립적인 환경으로 가정하여 사용자의 이동성을 모델링 한다. 이때 모델을 단순화시키기 위하여 공간적(Spatial) 이동성 정보[2]를 반영하는 이동 노드의 가변적인 속도는 고려하지 않고, 이동 노드의 방향은 이동할 인접 영역을 선택하는 확률 값을 사용한다[9]. 그리고 시간적(Temporal) 이동성 정보[3]를 반영하는 것으로 이동 노드들이 위치하는 영역의 초기 위치, 한 영역에서 상주 시간(Residence Time), 단위 시간당 사용자들의 서비스 요구 수, 연속미디어들의 특성을 다음과 같이 가정한다.

① (그림 2)와 (그림 3)에서 제시된 협력적 프록시 캐쉬 구조에서 각 프록시는 하나의 영역을 담당하는 구조이므로, 사용자가 이동하는 전체 영역들을 (그림 4)와 같이 $n \times n$ 형태의 정방형 구조로 간주한다. 이때 하나의 영역은 하나의 프록시가 담당하는 이동환경을 고려한다.

② 사용자들은 ①과 같은 환경에서 연속미디어를 하나의 프록시 서버에서 서비스 받고, 각 이동 사용자들의 초기 위치는 임의의 한 영역으로 설정된다. 일반적으로 연속미디어 응용에서 단위 시간당 사용자들의 서비스 요구 수[4]는 포아송 분포(Poisson Distribution)를 따르고, 서비스 요구 시간 간격(Inter-Arrival Time)은 포아송 과정을 따른다[8, 10]. 그리고 일반적인 연속미디어 응용에서 연속미디어들의 인기도에 따른 접근 확률은 *Zipf* 분포를 따르고, 연속미디어를 서비스 받고 있는 사용자가 각 영역에서 머무는 시간은 지수 분포(Exponential Distribution)를 따르게 되어 사용자가 한 영역에서 상주하는 시간을 추출해 낼 수 있다[10-12]. 이상과 같은 환경으로 연속미디어를 요구하는 각 사용자들의 이동 패턴과 전체 영역에서 사용자들의 패턴을 추출하여 사용자들의 이동성을 모델링 한다.



(그림 4) 정방형으로 구성한 프록시 영역들

3.4 협력적 프록시 캐싱

클러스터 프록시 그룹 내에서 각 프록시 간에 서로 주고받는 객체 요구에 관한 정보는 사용자의 요구 인덱스, 해당 객체의 인덱스, 객체의 시작 위치, 영역 진입 시간, 영역 진출 시간, 해당영역 가중치의 값을 나타내는 자료 구조이다. (그림 3)과 같은 구조에서 임의의 한 프록시 P_a^b에서 객체 O_k에 대한 사용자 요구 i가 발생할 때, 객체 O_k가 P_a^b에 존재하면 P_a^b의 캐쉬로부터 O_k를 서비스하고 그렇지 않으면 원격지 서버로부터 O_k를 서비스 한다. 이때 서비스되는 O_k를 해당 캐쉬에 캐싱하기 위해서 캐쉬의 잔여공간(Remained Cache Space)을 검사하여 재배치 여부를 결정한다. 만약 잔여공간이 충분하지 않으면 해당 캐쉬에 점유하고 있는 객체를 선택하는 재배치 객체 선택이 이루어져야 한다. 재배치 객체 선택은 해당 프록시의 객체 요구에 관한 정보와 인접한 프록시에서 발생한 객체 요구에 관한 정보를 고려하여 이루어진다. 이와 같은 절차로 이루어지는 협력적 프록시 캐싱은 (그림 5)와 같다.

$f_{LFU}(O_k, t_n)$는 t_n 시간에서의 재배치 척도 값이고 $Size(O_k)$는 객체 O_k의 크기를 나타낸다. 그리고 (그림 5)에서 재배치 객체 선택에 대한 알고리즘은 (그림 8)에서 나타내었다.

2) 사용자가 이동하는 패턴에 따르는 지식정보나 예측이 고려되어 이동할 방향을 예측하는데 사용된다.

3) 사용자가 영역(셀)을 방문하여 다른 영역(셀)로 이동할 때까지의 시간으로 영역(셀)의 상주시간을 구하는데 사용된다.

4) 사용자들의 요구들을 생성하기 위해서 포아송 분포를 사용하여 요구들의 도착 시간 간격을 계산한다.

① 요구 i의 시작에 따른 P_a^b의 객체 정보 DB의 서비스 정보 수정.
② 협력 프록시에게 객체 O_k에 대한 요구 i시작을 알림.
③ **while**(! 요구 i에 대한 서비스가 종료) {
  if ($\exists O_k \in \{P_a^b$캐쉬$\}$) {
    P_a^b의 캐쉬로부터 O_k를 서비스
  } **else** {
  서버로부터 O_k를 서비스 ;
  if *Size*(O_k) < (*Size*(P_a^b의 *Remained Cache Space*) {
    O_k를 P_a^b의 캐쉬에 캐슁
    } **else** {
    P_a^b의 객체 정보 DB로부터 협력 프록시들의 O_k에 대한 정보를 검색 ;
    재배치 척도 값 $f_{LFU}(O_k, t_n)$를 계산 ;
    P_a^b에서 재배치 척도 값으로 재배치 객체 선택 ;
    O_k를 P_a^b의 캐쉬에 캐슁
    }
  }
}
④ 요구 i의 종료에 따른 P_a^b의 객체 정보 DB의 서비스 정보 수정.
⑤ 협력 프록시에게 객체 O_k에 대한 요구 i의 종료를 알림.

(그림 5) 협력적인 프록시 캐슁

3.5 재배치 객체 선택 기법

클러스터 그룹 내의 각 프록시들은 가중치를 가지며, 각 프록시에서 발생하는 연속미디어 객체에 대한 사용자 요구를 서비스하기 위해서는 클러스터 그룹 내의 프록시들로부터 전달받은 객체 요구 정보 리스트를 검색한다. 그래서 현재 클러스터 그룹 내의 프록시들에서 서비스 중인 객체 스트림의 재생량과 객체의 가중치 값을 적용하여 재배치 객체를 선택한다. 재배치 객체 선택 기법으로서 기존의 캐슁 기법들 중에 연속미디어 서비스에서 객체의 인기도를 잘 반영할 수 있고, 다른 기법에 비해 일반적으로 좋은 성능을 보이며, 비교적 알고리즘이 간단한 LFU기법을 본 연구에서는 사용한다[13].

3.5.1 가중치_적용_재생량을 이용한 재배치 척도값

현재시간 t_n에서 $\Delta t_n (= | t_n - t_{n-1} |)$시간 동안 발생한 요구에 대한 가중치 값과 객체의 **재생량**을 곱한 값을 **가중치_적용_재생량(Weight & Amount)**이라 하고, 이를 W_A로 표현한다. 그래서 Δt_n동안 발생한 r개의 요구들의 W_A값들을 더한 값($Sum\_W\_A_r$)을 LFU 알고리즘의 재배치 척도값으로 사용하며 이를 f_{LFU}로 표시하고, 식 (5)와 같이 표현한다.

$$f_{LFU}(O_k, t_n) = Sum\_W\_A_r(S_i(O_k), \Delta t_n) \quad (5)$$

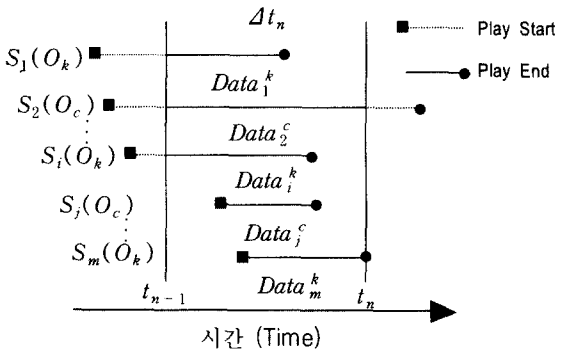
$f_{LFU}(O_k, t_n)$는 Δt_n시간 동안 연속미디어 객체 O_k를 참조하는 모든 스트림들의 **가중치_적용_재생량의 총합으로** t_n시간에서 LFU 알고리즘의 재배치 척도를 표시하며 $S_i(O_k)$를 요구 i에 대한 객체 O_k를 전송하는 스트림이라 하고, $Sum_W_A_r(S_i(O_k), \Delta t_n)$는 Δt_n동안 연속미디어 객체 O_k를 참조 중인 r개의 W_A들의 총합를 표시한다. **가중치_적용_재생량**을 구하기 위해서 스트림의 가중치를 다음과 같이 정의한다.

[정의 4] 스트림의 가중치

[정의 3]에서 각 프록시에서 발생한 요구 i의 스트림이 $S_i(O_k)$이면 P_a^b에서 발생한 스트림의 가중치는 1이고, P_a^b의 인접한 프록시(P_{a-p}^{b-q})에서 발생한 스트림의 가중치는

$$w(P_{a-p}^{b-q}(S_i(O_k))) = \frac{1}{Num(C(P_a^b))}, \quad where\ \forall p, q \in Z, -1 \le p, q \le 1 \quad (6)$$

와 같이 표현한다.



(그림 6) Δt_n에서 연속미디어 객체의 재생량

식 (5)에서 **가중치_적용_재생량의 총합** $Sum_W_A_r(S_i(O_k), \Delta t_n)$은 다음과 같은 ①, ②단계로 구한다.

① (그림 6)과 같이 이동 노드의 요구 i가 Δt_n동안 연속미디어 객체 O_k를 참조 중인 스트림 $S_i(O_k)$를 재생한 데이터 량을 $Data_i^k$라 하고, 식 (6)에서 객체 O_k에 대한 요구 i의 가중치가 $w(P_{a-p}^{b-q}(S_i(O_k)))$이므로, 요구 i의 $W_A(S_i(O_k), \Delta t_n)$는 재생한 데이터 량과 스트림의 가중치를 곱한 값으로 식 (7)과 같이 표현된다.

$$W\_A(S_i(O_k), \Delta t_n) = Data_i^k \times w(P_{a-p}^{b-q}(S_i(O_k))) \quad (7)$$

② 연속미디어 객체의 재생량이 (그림 6)과 같을 때 Δt_n 동안 클러스터 프록시에서 객체 O_k를 참조하는 스트림 $S_i(O_k)$에 대한 사용자들의 요구수를 r이라 가정하면, 식 (7)에서 나타낸 각 요구에 대한 **가중치_적용_재생량** 값을 모두 더한 값이 객체 O_k에 대한 **전체 가중치_적용_재생량**으로 식 (8)과 같이 표현된다.

$$\Sigma\ W\_A_r(S_i(O_k), \Delta t_n)$$
$$= W\_A(S_1(O_k), \Delta t_n) + \cdots + W\_A(S_i(O_k) + \cdots + (S_r(O_k)\Delta t_n)$$
$$= \Sigma W\_A(S_i(O_k), \Delta t_n),\ where\ r = |i|,\ i = \text{사용자 요구 인덱스} \quad (8)$$

3.5.2 *LFU* 알고리즘

LFU 알고리즘은 다음과 같은 2단계 구조로 수행된다.

① 1단계 : 가중치와 재생량을 고려한 $f_{LFU}(O_k)$값 계산

$f_{LFU}(O_k)$를 계산하기 위해서 클러스터 프록시 그룹 내의 프록시에서 객체 O_k의 요구 정보가 추가될 때, 식 (6)과 식 (7)으로부터 $W_A(S_i(O_k), \Delta t_n)$을 계산한다. 그리고 이 값을 현재값과 누적하는 모듈과 객체 O_k 요구의 서비스가 끝났을 때 지원된 스트림의 $W_A(S_i(O_k), \Delta t_n)$값만큼 감소시키는 모듈, 그리고 지원되고 있는 스트림의 $W\_A(S_i(O_k), \Delta t_n)$ 값을 반환하는 모듈은 (그림 7)과 같다.

```
Procedure Weight & Amount_add
input : 요구 i, k(스트림 객체의 인덱스), W_A(i) (요구 i의 가중치 적용재생량)
output : none
begin
    incrementWeight & Amount ( O_k, W_A(i))
    { O_k . CurrentWeight & Amount + = W_A(i) } ;
      // 현재 지원되고 있는 요구 i가 참조하는 객체 O_k의
         현재 가중치_적용_재생량 값에 요구 i의 값을 누적한다.
end

Procedure Weight & Amount_delete
input : 요구 i, k(스트림 객체의 인덱스), W_A(i) (요구 i의 가중치 적용재생량)
output : none
begin
    decrementWeight & Amount ( O_k, W_A(i))
    { O_k .CurrentWeight & Amount - = W_A(i) } ;
      // 현재 지원되고 있는 요구 i가 참조하는 객체 O_k의
         현재 가중치_적용_재생량 값에서 요구 i의 값 만큼 감소시킨다.
end

Procedure get_Weight & Amount
input : 요구 O_k (스트림 객체)
output : f_LFU 값
begin
return O_k .CurrentWeight & Amount ;
    // 객체 O_k의 지원되고 있는 가중치_적용_재생량 값을 반환한다.
end
```

(그림 7) 가중치 값 조정 및 반환 모듈

② 2단계 : 재배치 객체 선택

입력값 R_{CS}는 클러스터 프록시 그룹의 한 프록시에서 객체에 할당되지 않은 캐쉬의 잔여 공간을 표시하며, C_O는 클러스터 프록시 그룹의 한 프록시에서 담당하는 캐싱된 객체들의 집합을 나타낸다. 한 프록시에서 캐싱된 객체들 중 객체 j를 선택하여 $f_{LFU}(O_j)$값이 요구한 객체의 $f_{LFU}(O_k)$보다 작은 객체를 검색하여 없으면 재배치하지 않고, 있으면 객체 j를 해당 캐쉬의 잔여공간에 더한다. 이러한 과정을 $Size(O_k) < R_{CS}$가 될 때까지 반복하여 $Size(O_k)$를 재배치할 잔여공간을 확보하여 재배치한다. 재배치 객체 선택 알고리즘은 (그림 8)과 같다.

```
Procedure 재배치 객체 선택
입력 : f_LFU(O_k), O_k, R_CS
출력 : 재배치 여부
Begin
  V ← φ
  while(1) {
    if (Size(O_k) < R_CS) return 재배치함.
    if ({j | ∀j ∈ C_0, f_LFU(O_j) < f_LFU(O_k),
            j ∉ V인 j를 선택 } = φ
       return 재배치하지 않음.
    R_CS ← R_CS + Size(O_j)
    V ← V ∪ {j}
  }
End
```

(그림 8) 재배치 객체 선택 알고리즘

4. 실험 및 성능 평가

제안한 협력적 프록시 캐싱 정책의 성능을 분석하기 위해서 이동 사용자의 이동 비율을 기준으로 하여 객체 단위와 세그먼트 단위로 재배치 기법을 적용하여 StandAlone[5] 프록시 캐싱과 협력적[6] 프록시 캐싱을 수행하였다.

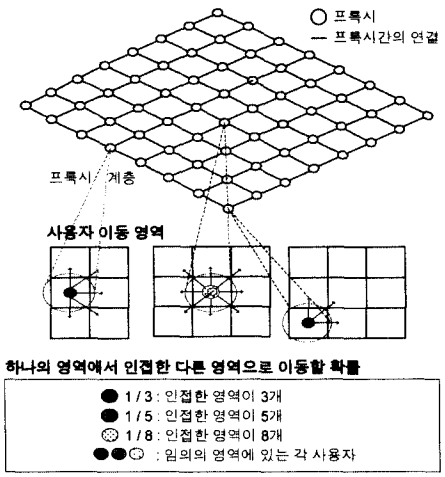
4.1 실험 환경

4.1.1 협력적 프록시 캐싱의 환경

한 영역에서 발생하는 요구들을 하나의 프록시가 서비스하는 StandAlone 프록시 캐싱과 제안한 협력적 프록시 캐싱을 (그림 9)와 같은 구조의 모델에서 실험하였다. 그리고 다음과 같은 환경을 기본적으로 가정하여 이동 사용자의 트레이스 데이터를 생성하고 본 실험을 수행한다.

① 각 프록시의 CPU 자원, 메모리 자원의 대역폭은 충분하고 메모리의 크기만을 고려한다.
② 원격지 서버와 각 프록시간의 네트워크 대역폭, 각 프록시들 간의 네트워크 대역폭, 각 프록시와 이동 노드들 간의 네트워크 대역폭은 충분하다고 본다.
③ 이웃하는 프록시들과의 거리는 일정한 거리로 논리적으로 연결되어 있다.
④ 하나의 프록시는 하나의 영역에서 발생하는 이동 사용자들의 요구를 담당한다.
⑤ 한 영역에서 이동 노드의 이동 방향은 이동할 인접 영역을 선택하는 확률 값을 사용하고, 이동 노드가 요구한 미디어 객체는 처음부터 끝까지 본다고 가정한다.

5) 기존의 단일 프록시에서 '단일' 대신에 'StandAlone'으로 표시한다.
6) 실험의 결과에서는 'Cooperate'로 표시한다.



(그림 7) 프록시 계층과 사용자 이동 영역

4.1.2 환경 변수

실험의 구현에 사용된 환경 변수는 <표 2>~<표 4>와 같다.

〈표 2〉 실험 환경 변수

환 경 변 수	고 정 값
모의실험 시간	5시간
연속미디어 개수	1000개
CBR 연속미디어 길이	20Mbyte
미디어의 요구 비트율	128Kbps
시간당 전체 서비스 요구 도착율	1000(명/초)
프록시 영역의 대역폭	제한 없음
프록시 영역의 개수	64(8×8)

〈표 3〉 캐싱 정책 환경 변수

환 경 변 수	고 정 값
재배치 기법	LFU
프록시 메모리 크기	512Mbyte
Time Window Size	5분

〈표 4〉 사용자 이동 환경 변수

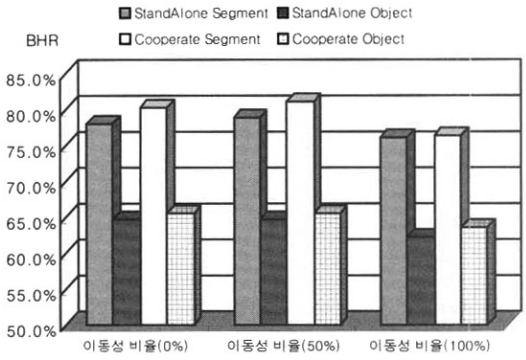
환 경 변 수	값 의 범 위
이동성향의 구분점[7]	10분
이동성이 큰 사용자 비율[8]	0%, 50%, 100%
이동할 영역의 선택 확률	1/3, 1/5, 1/8

4.2 성능분석

제안한 협력적 프록시 캐쉬 구조에서 StandAlone 캐싱과 협력적 캐싱을 객체 단위와 세그먼트 단위의 재배치 기법으로 나누어 이동성 비율에 따라 수행하고, 그 결과에 대해서 성능을 분석한다. 그리고 협력적 프록시 캐싱의 성능을 평가하기 위해서 프록시 캐쉬에서의 ① BHR(Byte Hit Ratio), ② BRR(Byte Replacement Ratio), ③ 광대역 요구 대역폭[9]을 기준으로 평가한다.

4.2.1 BHR 비교

(그림 10)은 전체적으로 세그먼트 단위의 기법(평균 78.5%)이 객체 단위의 기법(64.5%)보다 평균 14%가량 BHR 값이 높다는 것을 보여준다. 그리고 이동성 비율이 50%일 경우가 다른 경우보다 전체적으로 1~2% 정도 높으며, 특히 세그먼트 단위로 협력적 캐싱하는 것이 81.2%로 가장 높다.



(그림 10) 이동성 비율에 따른 BHR

그런데 협력적 캐싱의 세그먼트 단위 기법에서 이동성 비율이 100%인 경우는 이동성 비율이 0%나 50%인 경우보다 BHR 값이 3~4% 낮다. 이것은 협력적 프록시 구조에서 각 프록시는 이웃하는 프록시들과 서로 협력하여 캐싱정보를 공유하는데, 이동성 비율이 낮으면(0%인 경우) 해당 영역내의 사용자 요구만 처리하고 인접한 영역의 사용자 요구는 반영하지 않기 때문이며, 이동성 비율이 높으면(100%인 경우) 이동하는 사용자가 많아서 사용자 중 일부가 협력하는 프록시들의 영역(클러스터) 밖으로 이동하는 경우가 발생하고 또한 클러스터 밖의 사용자가 이동해올 수도 있기 때문이다.

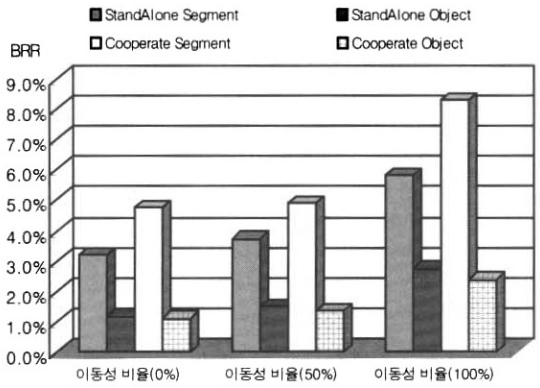
4.2.2 BRR 비교

(그림 11)은 전체적으로 세그먼트 단위의 기법(Standalone : 4.2%, Cooperate : 5.9%)이 객체 단위의 기법(Standalone : 1.6%, Cooperate : 1.8%)보다 평균 2.6%와 4.1% 정도로 BRR 값이 높은 것을 나타낸다. 협력적 캐싱에서 이동성 비율(100%)이 높을수록 세그먼트 단위 기법의 BRR 값(8.3%)은 증가하여 조금 더 자주 재배치가 발생한다는 것을 의미하며, 이것은 재배치하는데 부하가 큼을 알 수 있다.

7) 이동사용자가 하나의 영역 내에서 10분 이내에 다른 영역으로 이동할 경우 이동성향이 크다고 보고, 10분 이상 영역 내에서 머무를 때 이동성향이 적다고 본다. 이동 사용자의 비율을 측정하는데 사용된다.

8) 간단하게 '이동성 비율'이라고 표현한다.

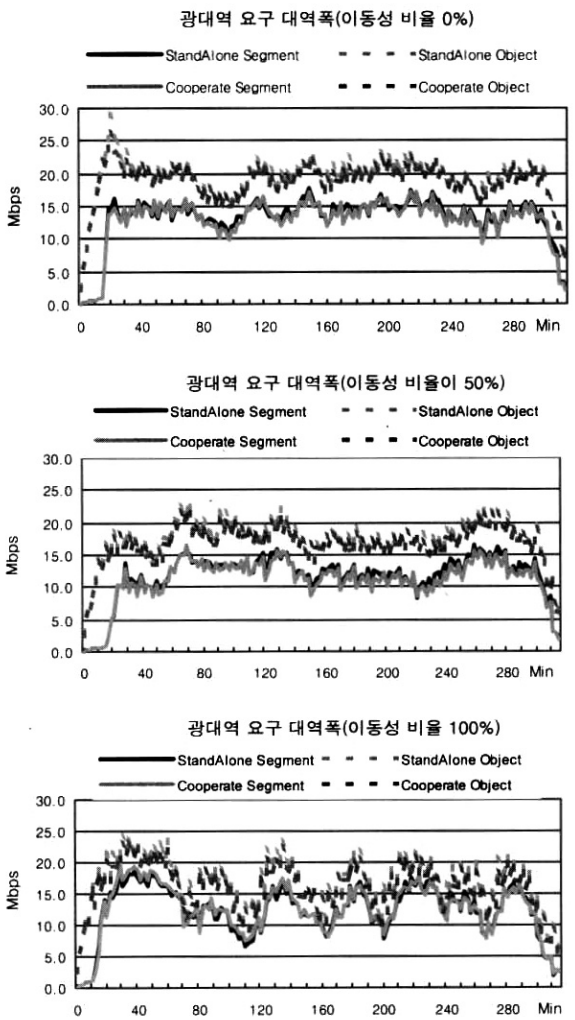
9) 프록시가 서버에 요구한 스트림의 량



(그림 11) 이동성 비율에 따른 BRR

4.2.3 광대역 요구 대역폭 비교

(그림 12)는 이동성 비율이 0%, 50%, 100%인 경우의 광대역 요구 대역폭을 나타낸다.

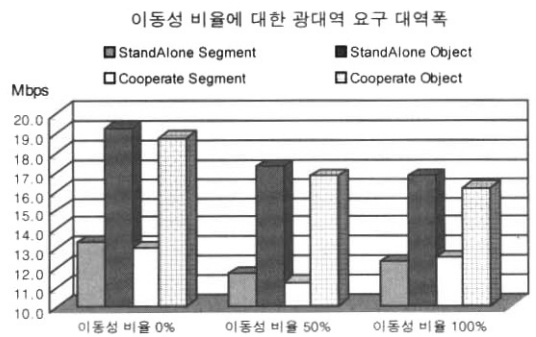


(그림 12) 광대역 요구 대역폭

이동성 비율이 0%일 때 객체 단위의 재배치 기법일 경우 처음 20초 동안은 캐쉬 안정화 작업을 위해 객체의 캐싱이 많이 발생하여 서버로의 광대역 요구 대역폭이 높아진다. (그림 12)에서 이동성 비율에 따른 광대역 요구 대역폭의 평균값을 <표 5>에 나타내고 (그림 13)으로 표시하였다. 협력적 세그먼트 단위의 재배치 기법에서 이동성 비율 50% 일 때 11.2Mbps로 낮게 나와 캐싱의 효과가 좋음을 보인다. 광대역 요구 대역폭은 이동성 비율의 변화에 따라 세그먼트 단위의 재배치 기법일 경우에는 StandAlone 캐싱보다 협력적 캐싱이 0.3~0.4Mbps 작으며, 객체 단위의 재배치 기법일 경우에는 StandAlone 캐싱보다 협력적 캐싱이 0.5~0.6Mbps 작게 나타난다. 이는 협력적 프록시 캐싱이 캐쉬에서 히트율이 높아서 사용자의 요구에 대해 좀더 효과적으로 서비스할 수 있다는 것을 의미하며 원격지 서버로의 요구가 더 작게 발생한다는 의미이다. 그리고 전체적으로는 세그먼트 단위의 재배치 기법이 객체단위의 재배치 기법보다 4~6Mbps 정도 작게 나타난다. 이것은 세그먼트 단위의 재배치 기법이 객체단위의 재배치 기법보다 효율적인 것을 보여준다. 그리고 협력적 캐싱 적용 세그먼트 단위 재배치 기법의 경우에는 사용자 이동성 비율이 50%일 때 광대역 요구 대역폭이 11.2Mbps 로 가장 작게 나타난다. 이는 이동성 비율이 적당할 경우에 캐싱의 효과가 좋음을 보여준다.

〈표 5〉 이동성 비율에 따른 광대역 요구 대역폭의 평균값

구분 \ 이동성 비율	0%	50%	100%
StandAlone Segment	13.3	11.7	12.3
StandAlone Object	19.3	17.3	16.8
Cooperate Segment	13.0	11.2	12.5
Cooperate Object	18.8	16.8	16.1



(그림 13) 이동성 비율에 따른 광대역 요구 대역폭의 평균값

5. 결론 및 향후과제

본 연구의 주된 연구내용은 담당 영역 내의 사용자 요구 객체만을 참조하여 캐싱 대상을 선정하는 기존의 프록시 캐싱 정책과 달리 이동환경에서 인접한 영역에서 발생한 많은 사용자 요구 정보들도 활용하여 캐싱하는 협력적 프록시 캐

싱 정책을 제안하였다. 그래서 세그먼트 단위와 객체 단위로 재배치 기법을 적용하여 실험한 결과, 협력적 캐싱이 StandAlone 캐싱보다 효율적임을 알 수 있었고 재배치 기법의 적용에서 세그먼트 단위가 객체 단위보다 성능이 우수함을 알 수 있었다. 즉 (그림 10)과 (그림 13)에서 이동성 비율이 작을 때(0%)와 클 때(100%)보다 이동성 비율이 적당할 때(50%)에 협력적 프록시 캐싱의 BHR값이 더 높고, 또한 광대역 요구 대역폭이 더 낮아 더 좋은 캐싱의 효과를 나타내고 있다. 따라서 본 연구에서는 사용자의 이동성 비율과 객체 단위보다는 세그먼트 단위의 재배치 수행이 협력적 프록시 캐싱의 성능에 영향을 미치는 중요한 요인이 됨을 알 수 있었다.

향후 연구되어야 할 부분으로는 다음과 같다.

- 사용자 환경을 구분하여 사용자의 이동성향에 따른 이동성 예측 모델의 연구가 필요하며, 이 연구를 기반으로 하여 프록시에서의 선반입, 재배치 기법 등의 성능 향상이 필요하다.
- 다양한 이동 노드의 하드웨어 제약사항에 따라 프록시 캐싱만으로 연속미디어 객체의 서비스에 대한 품질(지터나 전송지연, 전송에러)을 보장할 수 없으므로 프록시에서의 트랜스 코딩 문제와 같이 연구할 필요가 있다.
- 사용자의 이동성에 따른 핸드오프 발생시 Mobile IP를 이용한 프록시 간의 경로 재설정 및 캐싱 정보의 전달 기법에 대한 연구가 필요하다.

참 고 문 헌

[1] J. Jing, A. Helal and A. Elmagarmid, "Client-Server Computing in Mobile Environments," ACM Computing Surveys, Vol.31, No.2, pp.117-157, June, 1999.

[2] M. Satyanarayanan, "Fundamental Challenges in Mobile Computing," 5th ACM Symposium on Principles of Distributed Computing, pp.1-7, May, 1996.

[3] Adam Wolisz, "Mobility in Multimedia Communication," http://www.tkn.tu-berlin.de/~wolisz/wolisz.html.

[4] 박재원, 김문정, 엄영익, "이동 컴퓨팅 환경의 분산 파일 시스템", 한국정보과학회지, 제16권 제1호, pp.30-37, 1998년 1월.

[5] S. Hadjiefthymiades and L. Merakos, "Proxy+Path Prediction : Improving Web Service Provision in Wireless-Mobile Communications," Mobile Networks and Applications, Kluwer Academic Publishers, Vol.8, pp. 389-399, 2003.

[6] W. H. O. Lau, M. Kumar and Svetha Venkatesh, "A Cooperative Cache Architecture in Support of Caching Multimedia Objects in MANETs," WoWMoM'02, pp. 56-63, Sept., 2002.

[7] J. Xu, Q. Hu, W.-C. Lee and D. L. Lee, "Performance Evaluation of an Optimal Cache Replacement Policy for Wireless Data Dissemination," IEEE Transaction on Knowledge and Data Engineering (TKDE), Vol.16, No. 1, pp.125-139, Jan., 2004.

[8] Z. Xiang, Q. Zhang, W. Zhu and Y. Zhong, "Cost-based Replacement Policy for Multimedia Proxy across Wireless Internet," IEEE Globecom'01, Vol.3, pp.2009-2013, Nov., 2001.

[9] 이화세, 이승원, 박성호, 정기동, "이동환경에서 영역기반의 네트워크 캐싱 효용성 분석", 한국멀티미디어학회논문지, 제7권 제5호, pp.668-679, May, 2004.

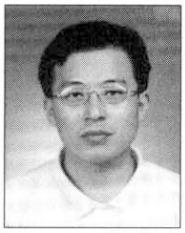
[10] L. Breslau, P. Cao, L. Fan, G. Philips and S. Shenker, "Web Caching and Zipf-like Distributions : Evidence and Implications," IEEE INFOCOM'99, Vol.1, pp.126-134, 1999.

[11] C. Cunha, et al., "Characteristics of WWW Client-based Traces," Technical Report BU-CS-95-010, Dept., of Computer Science, Boston University, July, 1995.

[12] 박성호, "프록시 캐쉬에서의 연속미디어 데이터 재배치 기법", 박사학위 논문, 부산대학교, Feb., 2002.

[13] Luigi Rizzo and Lorenzo Vicisano, "Replacement Policies for a Proxy Cache," IEEE/ACM Transaction on Networking, Vol.8, No.2, pp.158-170, 2000.

[14] Martin Arlitt and Carey Williamson, "Web Server Workload Characterization and Performance Implications," IEEE/ACM Transactions on Networking, Vol.5, No.5, pp.631-444, Oct., 1997.



이 승 원

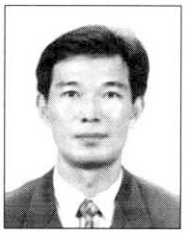
e-mail : swlee@pusan.ac.kr

1997년 부산대학교 전자계산학과(학사)

1999년 부산대학교 대학원 전자계산학과 (이학석사)

1999년~현재 부산대학교 대학원 전자계산학과 박사과정

관심분야 : 멀티미디어, 이동통신, VOD



이 화 세

e-mail : hslee@mnu.ac.kr

1985년 부산대학교 계산통계학과(학사)

1987년 부산대학교 대학원 계산통계학과 (이학석사)

2004년 부산대학교 대학원 전자계산학과 (이학박사)

1995년~1997년 밀양대학교 전자계산소 소장

1991년~현재 밀양대학교 컴퓨터공학부 교수

관심분야 : 인터넷 캐싱, 멀티미디어

박 성 호

e-mail : shpark@pusan.ac.kr
1996년 부산대학교 전자계산학과(학사)
1998년 부산대학교 대학원 전자계산학과 (이학석사)
2002년 부산대학교 대학원 전자계산학과 (이학박사)
2002년~현재 부산대학교 정보전산원 조교수
관심분야 : VOD 시스템, 인터넷 캐싱, 멀티미디어 이동통신

정 기 동

e-mail : kdchung@melon.cs.pusan.ac.kr
1973년 서울대학교(학사)
1975년 서울대학교 대학원(석사)
1986년 서울대학교 대학원 계산통계학과 (이학박사)
1990년~1991년 MIT, South Carolina대학 교환교수
1995년~1997년 부산대학교 전자계산소 소장
1978년~현재 부산대학교 전자전기정보컴퓨터공학부 교수
관심분야 : 병렬처리, 멀티미디어