웹 프락시 서버를 위한 적응형 캐시 교체 정책
(An Adaptive Cache Replacement Policy for Web Proxy Servers)

최 승 락 † 김 미 영 †† 박 창 섭 ††† 조 대 현 † 이 윤 준 ††††
(Seung Lak Choi) (Mi Young Kim) (Chang Sup Park) (Dae Hyun Cho) (Yoon Joon Lee)

요 약 월드 와이드 웹 사용의 폭발적인 증가는 네트워크 트래픽과 서버 부하의 급격한 증가를 초래하였다. 이러한 문제를 해결하기 위해 웹 프락시 캐싱 기술은 빈번히 요청되는 웹 문서를 사용자와 인접한 위치에 설치된 프락시(proxy)에 저장한다. 캐시 성능을 결정짓는 가장 중요한 요소는 캐시 교체 정책으로서, 가까운 미래에 빈번히 요청될 문서들을 저장하기 위해 사용된다.

캐시 교체 정책이 문서의 인기도를 정확히 예측하기 위해서는 웹 프락시 워크로드의 특성을 반영하는 것이 중요하다. 시간 지역성과 Zipf 빈도 분포는 웹 프락시 워크로드에서 빈번히 관찰되는 특성으로서 문서의 인기도를 예측하기 위한 중요한 속성들이다. 본 논문은 1) LFU를 기반으로 하여 Zipf 빈도 분포를 반영하며, 2) 문서들의 시간에 따른 인기도 감소를 효율적으로 측정하여 시간 지역성을 적응적으로 반영하는 적응형 LFU(ALFU) 캐시 교체 정책을 제안한다. 트레이스 기반의 모의 실험을 통해 다른 교체 정책들과 ALFU를 비교 분석한다. 실험 결과, ALFU는 다른 교체 정책보다 우수한 성능을 보였다.

키워드 : 월드 와이드 웹, 웹 캐시, 캐시 교체, 프락시

Abstract *The explosive increase of World Wide Web usage has incurred significant amount of* network traffic and server load. To overcome these problems, web proxy caching replicates frequently requested documents in the web proxy closer to the users. Cache utilization depends on the replacement policy which tries to store frequently requested documents in near future.

Temporal locality and Zipf frequency distribution, which are commonly observed in web proxy workloads, are considered as the important properties to predict the popularity of documents. In this paper, we propose a novel cache replacement policy, called Adaptive LFU (ALFU), which incorporates 1) Zipf frequency distribution by utilizing LFU and 2) temporal locality adaptively by measuring the amount of the popularity reduction of documents as time passed efficiently. We evaluate the performance of ALFU by comparing it to other policies via trace-driven simulation. Experimental results show that ALFU outperforms other policies.

Key words : world wide web, web cache, cache replacement, proxy

1. 서 론

· 본 논문은 (주)아라기술과의 산학연구과제 GB00630의 지원에 의하여 연구되었음.
† 비 회 원 : 한국과학기술원 전산학과
slchoi@dbserver.kaist.ac.kr
dhcho@dbserver.kaist.ac.kr
†† 비 회 원 : NOA ATS
mykim@noaats.com
††† 비 회 원 : KT 멀티미디어연구소 플랫폼연구팀
cspark0@kt.co.kr
†††† 종신회원 : 한국과학기술원 전산학과 교수
yjlee@cs.kaist.ac.kr
논문접수 : 2001년 6월 28일
심사완료 : 2002년 3월 18일

월드 와이드 웹 사용의 폭발적인 증가는 네트워크 트래픽과 서버 부하의 급격한 증가를 초래하였다. 그 결과 대부분의 사용자들은 빈약한 품질의 웹 서비스를 경험하게 되었다. 이러한 문제를 해결하기 위해 웹 프락시 캐싱 기술은 자주 요청되는 웹 문서를 사용자와 인접한 위치에 설치된 프락시(proxy)에 저장한다. 많은 인기 있는 웹 문서들이 프락시에서 서비스되므로 응답 시간, 네트워크 트래픽, 웹 서버 부하를 감소시킨다.

클라이언트로부터 요청되는 전체 웹 데이타는 프락시의 저장 공간보다 매우 크기 때문에 프락시는 모든 웹 문서를 저장할 수 없으며, 따라서 미래에 요청될 확률이 큰 웹

문서들만 저장해야 한다. 캐시 교체 정책(cache replacement policy)은 확률이 큰 웹 문서들만 캐시에 저장되도록 함으로써 웹 캐시의 효율을 높이는 것이 목적이다.

컴퓨터 시스템 워크로드(workload)의 대표적인 특성은 시간 지역성(temporal locality)과 참조 빈도(access frequency)이다[15]. 운영 체제, 컴퓨터 구조, 분산 파일 시스템 등 전통적 캐싱에서 시간 지역성과 참조 빈도는 미래 참조 가능성을 판단하기 위한 유용한 단서를 제공하였으며, 우수한 캐시 교체 정책을 위해 활발히 연구되어 왔다. 시간 지역성과 참조 빈도는 웹 프락시 워크로드에서도 발견되는 특성으로 우수한 캐시 교체 정책을 위해 반드시 고려되어야 한다[5, 6, 7, 8, 9, 11, 13].

전통적인 캐시 교체 정책 중의 하나인 LRU(Least-Recently-Used)는 가장 오래 전에 접근된 객체를 삭제함으로써 시간 지역성을 이용한다. LRU는 웹 프락시 워크로드에서 빈번히 관찰되는 인기 있는 객체들의 집합, 즉 핫 집합의 변화*(hot-set drift)*[11]를 반영한다. 웹 프락시 워크로드에서 한 번만 접근되는 객체는 매우 자주 발생된다[3]. LRU는 가장 최근에 요청된 객체가 가장 가치 있다고 판단한다. 한 번만 접근되는 객체는 캐시에 저장될 필요가 없음에도 불구하고 LRU에서는 이들이 오래 동안 캐시에 남아 있는다. 이는 LRU가 시간 지역성만을 반영하고 접근 회수를 반영하지 않기 때문에 발생하는 문제이다.

다른 대표적인 캐시 교체 정책으로 LFU(Least-Frequently-Used)가 있다. LFU는 참조 빈도가 높은 객체들만을 저장함으로써 LRU의 단점을 방지한다. 그러나 LFU는 워크로드 이동을 고려하지 않기 때문에 캐시 *오염 현상(cache pollution)*을 초래할 수 있다. 캐시 오염은 과거에 인기 있었던 객체들 때문에 새로 인기 있는 객체를 캐시에 저장하지 못하는 현상이다.

본 논문은 시간 지역성과 참조 빈도를 동시에 고려하고, 핫 집합 변화를 능동적으로 반영하는 적응형(Adaptive) LFU 캐시 교체 정책인 ALFU를 제시한다. ALFU는 LFU와 마찬가지로 참조 빈도를 기반으로 하나, 핫 집합 변화에 따라 시간 지역성에 대한 고려 정도를 조정함으로써 핫 집합의 변화가 다양한 워크로드에서 보다 나은 성능을 발휘한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 살펴본다. 3장에서 웹 프락시 워크로드의 특성과 핫 집합의 변화를 설명한다. 4장에서 핫 집합의 변화를 고려한 ALFU 캐시 교체 정책을 제안한다. 5장에서 실험을 통해 ALFU와 다른 캐시 교체 정책들을 비교한다. 마지막으로 6장에서 요약 및 결론을 내린다.

2. 관련 연구

기존 연구들은 웹 프락시 워크로드에 시간 지역성과 Zipf 빈도 분포가 존재함을 제시하였다. 이 장에서는 이 두 가지 특성을 중심으로 이전 연구들에 대해 논의한다.

여러 연구들은 웹 프락시 워크로드가 시간 지역성을 따른다고 제시하였다[7, 9, 11, 13]. 이러한 이유로 몇 가지 교체 정책[1, 2, 12]들은 시간 지역성을 이용하는 LRU 정책을 기반으로 하고 있다. LRU의 가장 심각한 문제는 매우 드물게 접근되는 객체들은 캐싱 할 필요가 없음에도 불구하고 이들에 의해 캐시 저장 공간이 소모된다는 점이다. 웹 프락시 워크로드에서 한 번 접근되는 객체는 전체 객체의 60% 이상을 차지하므로[3] 이 문제는 매우 심각하다.

이 문제를 해결하기 위해 Aggarwal 등[2]은 승인 컨트롤 정책(admission control policy) 을 제안하였다. 승인 컨트롤 정책은 요청된 객체가 캐시에 저장될 만한 가치가 있는지 판단한 후 저장한다. Johnson과 Shasha [10]는 2Q 교체 알고리즘을 제안하였다. 2Q 교체 알고리즘은 두 번 이상 접근된 객체들만 캐시에 저장된다. 이 정책들은 LRU처럼 새로 접근된 객체를 무조건 캐시에 저장하지 않는다. 그러나 이 정책들은 캐시에 저장될 가능성이 있는 객체들을 위해 추가적인 메모리와 관리를 필요로 한다.

몇몇 연구들[5, 6, 8]은 웹 프락시 워크로드가 Zipf 분포를 따른다고 주장한다. Zipf 분포에서 i번째로 참조 회수가 높은 객체에 대한 상대적 요청 확률은 $1/i^{\alpha}$ ($0<\alpha<1$)에 비례한다. 이는 높은 접근 빈도를 가지는 객체가 다시 요청될 확률도 커짐을 암시한다. Zipf 분포에 의하면 LFU 정책이 매우 효과적일 것으로 보인다. 그러나 LFU는 웹 접근 패턴에서 빈번히 발생하는 핫 집합 변화[11]에 매우 둔감하여 캐시 오염 현상, 즉 과거에 인기가 있었던 객체들에 의해 새로 인기를 얻은 객체들이 캐시에 저장되지 못하는 현상을 초래한다.

Robinson과 Devarakonda[14]는 캐시 오염 현상을 피하는 FBR(Frequency-based Replacement) 캐시 교체 정책을 제안하였다. FBR은 평균 접근 회수가 사전에 정의된 임계치를 초과하면 캐시된 객체들의 접근 회수를 절반으로 낮춘다. 또한 높은 접근 수를 가지는 객체가 캐시에 오래 동안 남아 있는 것을 방지하기 위해 최대 접근 회수를 정하여 객체의 접근 회수가 이 수치를 초과하지 못하도록 한다. FBR은 웹 프락시에서 비교적 우수한 성능을 보인다[4]. 그러나 FBR의 성능은 사전에 결정되어야 하는 파라미터인 두 가지 임계값에

의존적이다. 이는 FBR을 실제 시스템에 적용하기 어렵게 만든다.

Arlitt 등[3]은 LFU-DA 알고리즘을 제안하였다. LFU-DA는 교체가 발생할 때마다 캐시에 있는 모든 객체들의 접근 회수를 희생자(victim)의 접근 회수만큼 삭감하였다. 그러나 LFA-DA는 FBR처럼 수동으로 정의해야 하는 튜닝 파라미터가 없지만 희생자(victim)의 접근 회수가 어떠한 의미를 지니며 어떻게 캐시 교체 정책에 영향을 미치는지 밝히지 않고 있다.

3. 웹 프락시 워크로드

2장에서 언급하였듯이 기존 많은 연구 결과는 웹 프락시 워크로드가 시간 지역성과 Zipf 빈도 분포 특성을 지니고 있음을 밝혔다. 이 장에서는 시간 지역성과 Zipf 빈도 분포 특성을 핫 집합의 변화율과 관련하여 고찰한다.

시간 지역성은 최근에 참조된 객체가 다시 참조될 가능성이 높은 성질을 가리킨다. 시간 지역성에 따르면, 특정 객체 d가 참조된 이후로, 다른 객체에 대한 k번의 참조가 발생했다면, 객체 d의 참조 가능성은 $1/k$에 비례한다[6, 7]. 즉 객체를 참조한 후 시간이 지날수록 해당 객체의 미래 참조 가능성은 현격히 떨어진다고 볼 수 있다. 전통적인 캐싱 기법의 대표적 알고리즘인 LRU가 프락시 캐싱에서도 널리 사용되는 것은 바로 프락시 워크로드의 시간 지역성 때문이다.

다양한 웹 프락시의 통계 결과에 따르면 60~70%의 객체는 오직 한 번만 요청되며 있으며 25~40%의 객체가 전체 웹 참조 요청의 70%을 차지하는 경향을 가진다고 한다[6, 13]. 이 결과를 통해 참조 회수가 높은 객체일수록 미래 참조 가능성이 높다고 판단할 수 있다. 참조 회수가 높은 객체를 캐시에 저장하는 LFU가 웹 캐싱에서도 사용되는 것은 이러한 성질 때문이다.

시간 지역성과 Zipf 빈도 분포는 객체들의 미래 참조 예측을 위한 유용한 특성들이다. 그러나 인기 있는 객체들, 즉 핫 집합의 변화 정도에 따라서 두 특성의 유용성은 차별화된다. 핫 집합의 이동이 느리다면 Zipf 빈도 분포를 고려하는 것이 유용하다. 이전에 자주 요청된 객체들이 이후에도 요청될 가능성이 높기 때문에 과거에 접근 회수가 높은 객체들을 캐시에 저장하면 캐시 히트가 발생할 가능이 높아진다.

반면에 핫 집합의 이동이 빠른 경우 Zipf 빈도 분포는 이를 적절히 반영하지 못한다. 핫 집합의 빠른 이동으로 인해 새로 요청되는 객체가 인기가 높음에도 불구하고, Zipf 빈도 분포는 이미 캐시에 저장된 접근 회수가 높은 객체들이 보다 인기가 있다고 오판한다. 이에 반해, 시간 지역성은 새로 요청되는 객체가 인기가 높다고 판단하여 과거에 인기 있던 객체를 캐시에서 삭제하고 새로 인기있는 객체를 캐시에 저장한다.

[11]의 연구에서는 핫 집합의 크기를 사용자가 요구한 모든 객체 집합의 0.01%, 0.05%, 0.10%로 하여, 하루 단위로 변화율을 분석하였다. 분석 결과에 의하면 트레이스(trace)에 따라 변화율이 다양하며, 한 트레이스 내에서도 시간에 따라 변화율이 다양하였다. 따라서 시간 지역성과 Zipf 빈도 분포를 동시에 고려하되 핫 집합의 변화율에 따라 적응적으로 두 특성의 가중치를 다르게 둘 필요가 있다.

4. ALFU 캐시 교체 알고리즘

본 논문은 객체의 미래 참조 가능성을 보다 정확히 예측하기 위해 웹 프락시 워크로드의 특성을 고려하는 ALFU(Adaptive LFU) 캐시 교체 정책을 제안한다. 기존 연구에서는 미래 참조 가능성 예측을 위해 시간 지역성과 Zipf 빈도 분포의 반영 정도가 정적으로 결정되었다. ALFU는 핫 집합의 변화율을 탐지하여 변화율이 높은 경우 시간 지역성을 반영하는 정도를 높이고, 변화율이 낮은 경우 Zipf 빈도 분포를 반영하는 정도를 높임으로써 캐시 히트 비율을 높인다.

ALFU에서 각 객체는 미래 참조 가능성을 나타내는 *가치* f를 부여 받는다. 가장 낮은 f값을 가지는 객체가 캐시 교체시 희생(victim) 객체 v로 선정된다. 희생 객체는 캐시 공간 확보를 위해 캐시 교체시 방출되는 객체이다. 캐시에 저장되어 있지 않은 객체의 가치는 항상 0이며, 캐시에 새로 저장되는 객체의 가치는 1이다(수식 1 참조). D는 캐시에 저장될 수 있는 모든 객체의 집합이며, $C(t)$는 t 시점에 캐시에 저장되어 있는 모든 객체의 집합이다.

ALFU는 LFU와 마찬가지로 객체들의 인기도를 측정하기 위한 척도로서 접근 회수를 사용한다. LRU가 한 번의 요청을 받은 객체를 가장 인기 있는 객체로 간주하여 인기도를 잘 못 판단하는 경우가 많은 반면 LFU는 객체의 참조 회수를 이용함으로써 LRU보다 정확하게 객체의 인기도를 측정한다. 특히 웹 프락시 워크로드는 Zipf 빈도 분포를 보이기 때문에 접근 회수는 미래 참조 예측을 위한 훌륭한 척도로 사용될 수 있다.

LFU는 핫 집합의 변화율을 적절히 반영하지 못함으로 ALFU는 Zipf 빈도 분포와 더불어 희생 객체의 생명 기간(life time)을 이용한다. 희생 객체 v의 생명 기간 l_v는 희생 객체 v가 캐시에 저장된 기간을 의미한다. 기간은 희생 객체 v가 캐시에 저장되어 있는 동안 객체

들에 대한 접근 회수이다. LFU 정책을 사용할 때 생명 기간은 핫 집합의 변화율과 반비례하는 특성을 가진다. 만약 핫 집합의 변화율이 급격하다면 많은 요청에서 캐시 미스가 발생한다. 따라서 기존에 캐시에 저장되어 있던 객체들은 보다 빠르게 방출된다. 반대로, 핫 집합의 변화율이 느리다면 많은 요청에서 캐시 히트가 발생하며 따라서 캐시된 객체들은 오래 동안 캐시에 남아 있는다.

캐시 교체 발생시 캐시에 저장된 모든 객체의 f값에서 희생 객체들의 생명 기간과 반비례하는 값을 뺀다. 이렇게 함으로써 핫 집합의 변화율에 따라 캐시에 저장되어 있는 객체들의 가치를 조정한다. 핫 집합의 변화가 심하다면 객체들의 가치를 많이 떨어뜨리고, 반대로 핫 집합의 변화가 적다면 객체들의 가치를 적게 떨어뜨린다.

$t+1$번째 참조 요청이 발생했을 때, t 시점의 캐시 $C(t)$에 저장된 모든 객체들의 가치 값은 다음과 같이 정의된다. $r_{d,t}$는 객체 d가 t 시점에 참조되었는지를 나타내는 변수로써 참조되었다면 값이 1로 설정된다. V_{t+1}는 $t+1$번째 참조 요청에서 선택된 희생 객체들의 집합으로서 히트가 발생했을 때에는 공집합이다.

$$\forall d \in D - C(t),\ f_{t+1}(d) = r_{d,t+1} \tag{1}$$

$$\forall d \in C(t),\ f_{t+1}(d) = f_t(d) + r_{d,t+1} - \sum_{v \in V_{t+1}} p\, l_v^{-q} \tag{2}$$

$$\text{where } r_{d,t+1} = \begin{cases} 0 & d \text{ is not referenced at } t+1 \\ 1 & d \text{ is referenced at } t+1 \end{cases}$$

$$p, q \geq 0$$

p, q는 핫 집합 변화율을 반영하는 정도를 결정하는 값이다. p는 시간 지역성을 반영하는 가중치로써 0에서 p사이에서 시간 지역성을 반영하는 정도가 적응적으로 변화한다. p가 0일 때에는 Zipf 접근 빈도만을 고려하여 LFU와 동일하게 동작한다.

q는 희생 객체의 생명 주기 l_v를 시간 지역성 반영에 얼만큼의 영향을 줄 것인지 결정하는 적응도 값(adaptiveness value)이다. q가 작을 수록 f 값에 대한 l_v의 영향력은 감소하며, q가 0이면 희생 객체의 생명 주기는 f 값에 전혀 영향을 주지 않는다.

희생 객체의 생명 주기 분포에 따른 p, q값의 변경은 시간 지역성을 보다 적절히 반영하게 할 수 있다. 가령, 요청 개수가 증가하면 캐시 교체가 빈번해 지며, 결과적으로 l_v가 전반적으로 작아진다. 이 경우 l_v의 변화가 작아지기 때문에 시간 지역성에 대한 적응도가 감소한다. 따라서 시간 지역성에 대한 적응도를 유지하기 위해 q 값의 조정이 필요하다. q값의 조정은 전체적인 시간 지역성을 반영하는 정도를 변화시킴으로 p값의 조정도 아울러 필요하다. p, q값에 대한 보다 분석적인 고찰은 향후 연구 과제이다.

ALFU는 LFU 기반의 적응적 에이징(aging) 기법으로 해석될 수 있다. 에이징은 LFU 기반 캐시 교체 정책에서 캐시 오염 현상을 막기 위한 유용한 방법이다. 그러나 기존에 제안된 FBR[14]은 핫 집합 변화율에 따라 적응적으로 동작하지 못하고 수동으로 튜닝 파라미터의 값을 지정해야 한다. 이에 반해 ALFU는 희생 객체의 생명 기간을 통해 현재 핫 집합의 변화 정도를 측정하고 이를 고려하여 에이징을 한다.

ALFU는 객체의 인기도 변화를 매우 정확히 측정한다. 일반적으로 객체의 인기도는 일정 기간동안 상승하였다고 하락한다. ALFU는 LFU를 기반으로 함으로써 LRU에 비해 객체의 인기도 상승을 보다 정확히 측정한다. 그러나 LFU는 인기도 하락을 반영하지 못하여 캐시 오염 현상을 초래한다. ALFU는 희생 객체의 생명 기간에 따라 캐시에 저장되어 있던 객체들의 가치를 감소시킴으로써 객체의 인기도 하락을 적절히 측정한다.

```
L ← 0.0
for each request for object d do
  if d is in cache
    then f(d) ← f(d) + 1
    else
      while there is not enough free space in cache
      for d
        g ← min(f(e) | e is in cache)
        evict e such that f(e) = g
        L ← L + p/(l_e)^q
      Bring d into cache
      f(d) ← L + 1
```

그림 1 ALFU 알고리즘

이상 설명한 ALFU 캐시 교체 정책의 알고리즘은 그림 1과 같다. 캐시 교체시 캐시에 저장되어 있는 모든 객체들의 f 값을 줄이는 것은 매우 높은 비용을 필요로 한다. 그러므로 ALFU 교체 정책을 구현할 때는 모든 객체에 대해 f 값을 줄이는 것이기 보다는 새로 캐시에 저장되는 객체의 f 값을 증가시키는 방식을 취한다. 즉, 캐시에서 객체가 삭제될 때마다 변수 L에 희생 객체 v의 pl_v^{-q}를 더해준다. 캐시에 새로 추가되는 객체에 L 값을 더해 줌으로써 상대적으로 캐시 내의 객체들의 f 값을 줄여주는 것과 동일한 결과를 얻는다.

신속한 캐시 교체를 위해 f 값에 따라 객체들을 정렬시켜야 한다. 이를 위해 힙(heap) 구조를 이용한다. f 값은 객체가 참조되는 경우와 객체가 삭제되는 경우에 값

이 변경되며 이 때 해당 객체는 정렬 상태를 유지하기 힙 구조에서 위치를 변경해야 한다. 참조된 객체가 힙 구조에서 위치를 변경할 때의 시간 복잡도는 O(log$N$) 이다. 캐시에서 객체가 삭제되는 경우, 캐시에 저장되어 있는 모든 객체에서 동일한 값을 빼주기 때문에 객체들 간에는 f값의 대소 관계가 유지된다. 따라서 별도의 처리가 필요하지 않다.

고성능의 프락시 캐싱을 위해 일반적으로 객체들에 대한 사전 정보를 주 메모리에 상주시키는 것이 일반적이다. 프락시 캐싱은 주 메모리 역시 캐싱을 위한 공간으로 사용하기 때문에 딕셔너리 정보는 가급적 작아야 한다. 따라서, 프락시 캐시 교체 정책을 수행하기 위해 지나치게 많은 공간을 필요로 하는 교체 정책은 비효율적이다. 프락시 캐시 교체 알고리즘을 수행하기 위해 프락시 캐시 내의 객체 당 부가적으로 저장되는 정보의 양이 고정되어 있는 것이 바람직하다. 프락시 캐시 교체 알고리즘은 시간 복잡도가 O(log N)이하이며, 공간 복잡도(space complexity)는 O(N)을 만족해야 한다고 알려져 있다[7]. ALFU은 시간 복잡도가 O(log N)이며, 공간 복잡도(space complexity)는 O(N)로써 실제 구현 시 효율적으로 동작한다.

5. 성능 평가

본 장에서는 트레이스 기반의 모의 실험을 통해 4장에서 제안한 ALFU와 기존의 프락시 캐시 교체 정책인 LRU, LFU, SIZE[16], GD-Size[7]를 비교 분석한다. 본 논문에서는 한국과학기술원 프락시 캐시에서 추출한 트레이스와 NLANR (National Laboratory for Applied Network Research)에서 학술용으로 제공하는 프락시 캐시 트레이스를 사용하여 실험을 하였다.

NLANR 트레이스는 NLANR (National Laboratory for Applied Network Research)에서 학술용으로 제공하는 웹 프락시 캐시의 트레이스로서 웹 캐싱 알고리즘의 성능 측정에 사용되는 대표적인 트레이스이다. 실험은 일주일 동안의 트레이스를 이용하여 진행하였다. 표 1은 본 실험에서 사용한 NLANR 트레이스 특성에 대한 표이다.

KAIST 트레이스는 한국과학기술원에 설치되어 있는 프락시 캐시에서 얻은 트레이스이다. 실험은 하루치 트레이스를 가지고 진행되었다. 표 2는 KAIST 트레이스의 특성이다. NLANR 트레이스의 기간이 일주일인데 반해 KAIST 트레이스는 하루 동안이므로 KAIST 트레이스의 핫 집합 변화율이 NLANR 트레이스보다 상대적으로 작다고 판단할 수 있다.

표 1 NLANR 트레이스

Duration	2000.10.09 ~ 2000.10.15
Total Request	4167692
Total MBytes	52120.05MB
Total unique MBytes	32000.35MB
Number of Hits	1901854
HR_∞	45.63%
BHR_∞	38.60%

표 2 KAIST 트레이스

Duration	2000.2.19
Total Request	1920279
Total MBytes	20173.09MB
Total unique MBytes	10864.27MB
Number of Hits	1178458
HR_∞	61.40%
BHR_∞	46.10%

실험은 본 논문이 제안한 ALFU 기법과 기존의 프락시 캐시 교체 정책인 LRU, LFU, SIZE, GD-Size에 대해 프락시 캐시 크기를 달리하면서 진행되었다. 프락시 캐시 크기는 유일한 객체들의 사이즈 합계(Total unique size)를 기준으로 결정하였다. NLANR 트레이스에 대해서는 사이즈 합계의 1%인 320Mb부터 10%인 3.2Gb까지 프락시 캐시 크기를 조정하였고, KAIST 트레이스는 1%인 109Mb부터 10%인 1.1Gb까지 달리하면서 실험하였다.

ALFU는 캐시 교체 시에 희생 객체의 생명 주기의 비중을 결정하는 p, q 값을 모두 1로 하여 실험하였다. ALFU의 핫 집합 변화율에 대한 적응력으로 인해 실험 결과는 p, q 값이 모두 1인 경우에도 기존 다른 알고리즘에 비해 우수한 성능을 보이고 있다.

GD-Size는 GD-Size(1)과 GD-Size(packets) 두 가지 버전이 존재한다. GD-Size(1)은 각 객체을 웹 서버로부터 가지고 오기 위한 네트워크 비용을 모두 동일하게 1로 설정한 경우이다. GD-Size(packets)은 네트워크 비용을 고려한 경우이다. GD-Size(1)은 미스 비율을 줄이는데 주목적을 두며, GD-Size(packets)은 네트워크 트래픽을 줄이는데 주목적을 둔다. 본 실험에서는 GD-Size(packets)을 사용하였다.

SIZE는 크기가 가장 큰 객체를 삭제하는 캐시 교체 정책이다. 웹 객체들의 크기는 매우 다양하기 때문에 가급적 많은 수의 작은 객체를 저장하여 HR(hit ratio)를 높이고자 하는 정책이다.

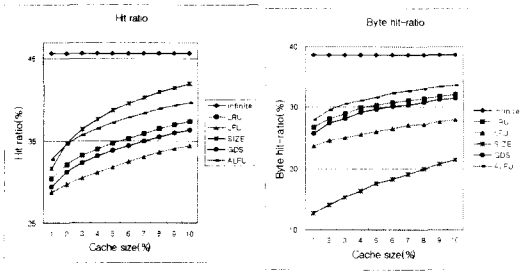


그림 2 NLANR 트레이스에서 HR, BHR

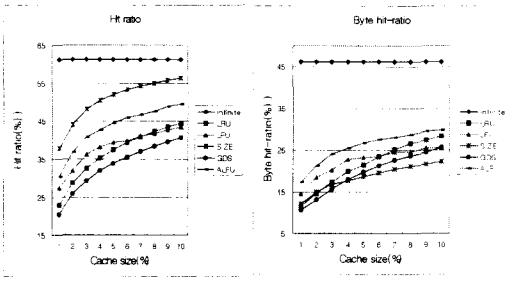


그림 3 KAIST 트레이스에서 HR, BHR

그림 2, 3은 실험 결과를 보여주고 있다. 기간이 일주일인 NLANR 트레이스는 기간이 하루인 KAIST 트레이스에 비해 핫 집합의 변화율이 높다. 그러한 이유로 시간 지역성을 고려하지 않은 LFU, SIZE의 경우 NLANR 트레이스에서의 성능이 KAIST 트레이스에서보다 비교적 좋지 않게 나왔다. 이는 핫 집합의 변화을 고려하지 않음으로 인한 캐쉬 오염 현상을 잘 보여주고 있다. LFU가 시간 지역성을 전혀 고려하지 않은 프락시 캐쉬 교체 정책임에도 불구하고 KAIST 트레이스 실험에서 LRU에 비해 HR, BHR(byte hit ratio)에서 모두 성능이 좋은 결과를 보인 것은 KAIST 트레이스의 핫 집합 변화율이 비교적 낮아 시간 지역성보다는 Zipf 빈도 분포가 보다 크게 캐시 교체 정책에 영향을 끼치기 때문이다.

두 실험에서 SIZE는 HR은 좋으나 BHR은 매우 좋지 않은 성능을 보였다. SIZE는 미래 참조 가능성을 고려하지 않고 단순히 작은 크기의 객체를 저장함으로써 많은 객체를 저장하여 HR을 높이는 교체 정책이기 때문이다. 시간 지역성을 고려하지 않는 교체 정책이기 때문에 KAIST 트레이스 실험보다 NLANR 트레이스 실험에서의 성능이 비교적 좋지 않다. LRU와 GD-Size는 두 실험에서 비슷한 성능을 보였다. 이는 LRU, GD-Size가 모두 시간 지역성을 고려하고 인기도를 고려하지 않은 교체 정책이라는 공통점 때문이다. 이러한 이유로 두 교체 정책 모두 KAIST 트레이스 실험 보다 NLANR 트레이스 실험에서 좋은 성능을 보였다.

ALFU는 시간 지역성과 Zipf 빈도 분포를 핫 집합의 변화율에 따라 적절히 고려하는 교체 정책이기 때문에 NLANR 트레이스 실험과 KAIST 트레이스 실험에서 모두 좋은 성능을 보였다. 즉 워크로드가 보이는 시간 지역성과 Zipf 빈도 분포의 강도에 따라 적응적으로 핫 집합의 변화를 고려함으로 다른 특성을 가지는 두 트레이스에서 모두 좋은 성능을 보였다. 시간 지역성이 지배적으로 영향을 미치는 NLANR 트레이스 실험에서 시간 지역성만을 고려한 LRU 보다 좋은 성능을 보였으며, Zipf 빈도 분포가 영향을 많이 미치는 KAIST 트레이스 실험에서는 Zipf 빈도 분포만을 고려한 LFU 보다 좋은 성능을 보였다.

6. 결 론

본 논문에서는 프락시 캐시 교체 기법이 고려해야 할 웹 프락시 워크로드의 특성을 알아보고, 기존 연구가 지니는 한계를 극복하는 새로운 프락시 캐시 교체 기법인 ALFU를 제안하였으며, 그 우수성을 트레이스 기반 실험을 통해 검증하였다.

기존 프락시 캐시 교체 정책은 대부분 웹 프락시 워크로드의 시간 지역성이나 Zipf 빈도 분포 하나만을 고려하였다. 두 특성을 모두 고려한 교체 정책의 경우도 핫 집합 변화율과는 상관없이 교체 정책을 설계할 때 고려 정도가 사전에 결정된다. 본 논문은 핫 집합 변화율이 희생 객체의 생명 기간과 반비례하는 특성을 제시한다. ALFU는 희생 객체의 생명 기간을 사용하여 시간 지역성과 Zipf 빈도 분포의 고려 정도를 적응적으로 변화하였다. 따라서 워크로드가 다양한 핫 집합의 변화율을 보일지라도 ALFU는 항상 우수한 성능을 유지한다.

ALFU는 시간 복잡도 및 공간 복잡도 측면에서도 매우 현실적이다. ALFU는 프락시 캐시 운영의 시간 복잡도가 $O(\log N)$으로 매우 효율적이며, 객체 당 부가적으로 저장되는 정보 역시 $O(1)$로서 전체 공간 복잡도는 $O(N)$이다. ALFU는 시간 복잡도, 공간 복잡도 측면에서 매우 효율적이라고 할 수 있다.

본 논문에서는 NLANR의 공개 트레이스와 한국과학기술원 프락시 캐쉬의 트레이스를 이용한 트레이스 기반 모의 실험을 통해 ALFU의 성능을 평가하였다. 이 실험을 통해 ALFU가 LRU, LFU, SIZE, GD-Size 보다 우수한 성능을 나타낸다는 것을 검증할 수 있었다.

차후 연구에서는 핫 집합 변화율과 객체의 생명 주기

의 관계를 분석하고, 핫 집합 변화율이 시간 지역성 및 Zipf 빈도 분포의 고려 정도에 어떠한 영향을 미치는지 정량적으로 분석할 예정이다.

참 조 문 헌

[1] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching proxies: limitations and potentials. In *Proceedings of the 4th International WWW Conference*, December 1995.

[2] C. Aggarwal, J. Wolf, and P. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94-107, 1999.

[3] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for Web proxy caches. In *Proceedings of the Workshop on Internet Server Performance (WISP99)*, may 1999.

[4] M. Arlitt, R. Friedrich and T. Jin. Performance evaluation of web proxy cache replacement policies. In *Proceedings of the 10th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Palma de Mallora, Spain, September 1998.

[5] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web (special issue on Characterization and Performance Evaluation)*, 1999.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the INFOCOM '99 conference*, March 1999.

[7] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. *In Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, December 1997.

[8] S. Glassman. A caching relay for the World Wide Web. In *Proceedings of the 1st International WWW Conference*, Geneva, Switzerland, May 1994.

[9] S. Jin and A. Bestavros. Sources and characteristics of web temporal locality. In *Proceedings of Mascots'2000: The IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Fransisco, CA, 2000.

[10] T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[11] A. Mahanti, D. Eager, and C. Williamson. Temporal locality and its impact on web proxy cache performance. *Performance Evaluation*, 42:149-164, 2000.

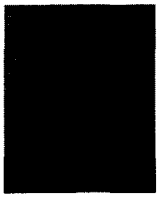
[12] J. E. Pitkow and M. M. Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *Proceedings of the 3rd International WWW Conference*, October 1994.

[13] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking*, 8(2), 2000.

[14] J. Robinson and M. Devarakonda. Data cache management using frequency-based replacement. In *Proceedings of the 1990 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*

[15] J. Spirn. Models and Measurements. Elsevier North-Holland, 1977.

[16] S. Williams, M. Abrams, C. Standridge, G. Abdulla, and E. Fox. Removal policies in network caches for world-wide web documents. In Proceedings of ACM SIGCOMM, 1996.



최 승 락

1993년 3월 ~ 1997년 2월 울산대학교 전자계산학과 학부 졸업. 1997년 3월 ~ 1999년 2월 한국과학기술원 전산학과 석사 졸업. 1999년 3월 ~ 현재 한국과학기술원 전산학과 박사 과정 재학중

김 미 영

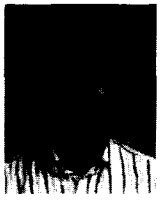
1994년 3월 ~ 1998년 2월 이화여자대학교 전자계산학과 학사. 1997년 12월 ~ 1998년 10월 동양시스템즈 근무. 1999년 3월 ~ 2001년 2월 한국과학기술원 전산학과 데이터베이스연구실 석사. 2000년 12월 ~ 2001년 9월 Bell Labs. 근무. 2001년 10월 ~ 현재 NOA ATS

박 창 섭

1991년 3월 ~ 1995년 2월 한국과학기술원 전산학과 학사 졸업. 1995년 3월 ~ 1997년 2월 한국과학기술원 전산학과 석사 졸업. 1997년 3월 ~ 2002년 2월 한국과학기술원 전산학과 박사 졸업. 2002년 2월 ~ 현재 KT 멀티미디어연구소 플랫폼연구팀

조 대 현

1994년 3월 ~ 1998년 2월 경북대학교 컴퓨터공학과 학사 졸업. 1998년 3월 ~ 2000년 2월 한국과학기술원 전산학과 석사 졸업. 2000년 3월 ~ 현재 한국과학기술원 전산학과 박사 과정 재학중



이 윤 준

1977년 서울대학교 계산통계학과 졸업. 1979년 한국과학기술원 전산학과에서 석사학위 취득. 1983년 France, INPGEN-SIMAG에서 박사학위 취득. 1983년 ~ 1984년 France, IMAG 연구원. 1984년 ~ 현재 한국과학기술원 전산학과 부교수. 1989년 MCC(미) 초빙연구원. 1990년 CRIN(불) 객원교수. 관심분야는 데이타베이스 시스템, 정보검색, 실시간 데이타베이스 등