계층적 프록시 캐쉬를 이용한 인터넷 성능 향상 기법*

Improving Performance of Internet by Using Hierarchical Proxy Cache

이효일**, 김종현***
Hyo Il Lee, Jong hyun Kim

Abstract

Recently, as construction of information infra including high-speed communication networks remarkably expands, more various information services have been provided. Thus the number of internet users rapidly increases, and it results in heavy load on Web server and higher traffics on networks. The phenomena cause longer response time that means worse quality of service. To solve such problems, much effort has been attempted to loosen bottleneck on Web server, reduce traffic on networks and shorten response times by caching informations being accessed more frequently at the proxy server that is located near to clients. And it is also possible to improve internet performance further by allowing clients to share informations stored in proxy caches.

In this paper, we perform simulations of hierarchical proxy caches with the 3-level 4-ary tree structure by using real web traces, and analyze cache hit ratio for various cache replacement policies and cache sizes when the delayed-store scheme is applied. According to simulation results, the delayed-store scheme increases the remote cache hit ratio, that improves quality of service by shortening the service response time.

* 이 연구는 1999년도 연세대학교 학술연구비 지원에 의한 것임.
** 삼성SDS
*** 연세대학교 전산학과 교수

1. 서 론

초고속 정보통신망을 비롯한 정보 인프라의 구축이 확대되면서 다양한 종류의 서비스들이 출현하고 있다. 특히, 그 중에서 가장 널리 사용되고 있는 것은 인터넷을 통한 World Wide Web(WWW; 이하 웹이라 함) 서비스이다. 최근 웹의 사용자 수가 크게 증가함에 따라 웹 서버에 걸리는 부하(load)와 통신망의 트래픽이 급증하고 있다. 특히, 유용한(인기 있는) 정보를 가진 웹 서버는 하루에 수십만 이상의 액세스 요구들을 받고 있으며, 그와 같은 높은 부하는 응답 시간이 길어지게 하는 주요 요인이 되고 있다. 이와 같은 현상은 앞으로 인터넷 서비스가 더욱 활성화되고 사용자가 늘어날 것으로 예상됨에 따라 시급히 해결해야 할 문제로 대두되고 있다.

웹 서버는 사용자로부터 어떤 웹 문서에 대한 액세스 요구가 들어오면 그 문서를 디스크로부터 읽어서 사용자 컴퓨터로 보내게 된다. 이 과정에서 웹 서버와 사용자 컴퓨터가 지리적으로 상당히 멀리 떨어져 있는 경우에는 전송 시간이 길어지고, 경로상에 있는 통신망에 트래픽을 유발하게 된다. 그런데 한 번 액세스된 문서는 그 사용자에 의해 다시 액세스될 가능성이 있으며, (인기도가 높은 문서라면) 그 사용자 가까이 위치한 다른 사용자들에 의해 액세스될 가능성도 있다. 그런 경우에는 매 액세스 때마다 문서를 웹 서버로부터 읽어오기 보다는 그 문서를 사용자 컴퓨터 혹은 그와 가까이 위치한 프록시 서버(proxy server)에 저장해둔다면 다음 액세스에서는 응답 시간이 크게 단축될 것이며, 통신망의 트래픽도 줄일 수 있을 것이다. 이러한 기술을 웹 캐싱(web caching)이라고 부르며, 인터넷 서비스와 사용자들이 계속 증가함에 따라 필수적으로 사용되어야 할 기술이다. 그러나 인터넷 정보들은 종류와 크기가 다양한 반면에, 사용자 컴퓨터들이나 프록시 서버의 저장장치 용량에는 한계가 있을 수밖에 없기 때문에, 인터넷 서비스 향상을 위해서는 효율적인 캐싱 방식이 반드시 설계되어야 한다.

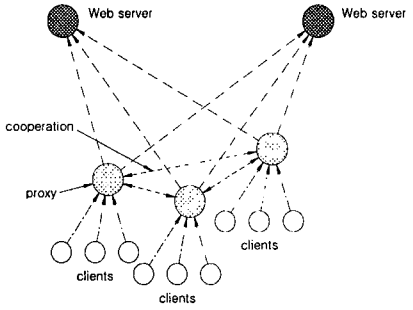
인터넷 정보를 액세스하는 과정에서 웹 서버와 통신망 트래픽을 줄임으로써 서비스의 처리 속도를 높이기 위한 웹 캐싱은 일반적으로 다음과 같은 세 레벨들이 고려되어 왔다. 먼저 1차 캐쉬(first-level cache)는 사용자 컴퓨터, 즉 웹 브라우저에 위치하며, 여기에는 빈번히 액세스되는 문서들을 저장하는 적은 양의 기억장치와 디스크 공간을 가지고 있다. 2차 캐쉬(second-level cache)는 일반적으로 프록시 서버에 위치한다. 프록시 서버에는 많은 수의 클라이언트들이 접속되기 때문에 제한된 캐쉬 크기로 적중률(hit ratio)을 높이는 방법이 필요하다. 마지막으로, 3차 캐쉬(third-level cache)는 웹 서버의 주기억장치를 이용하여 캐싱하는 방법이다. 이 경우에도 캐쉬에 적중되면 웹 서버의 디스크를 액세스하는 동작이 필요 없게 된다.

Chankhunthod와 Schwartz[1] 및 Wessel et al.[2]은 사용자 컴퓨터와 웹 서버 사이에 존재하는 여러 프록시 서버들에 위치한 캐쉬들을 이용하여 계층적 캐쉬(hierarchical cache)를 구성하는 것을 제안하였다. 각 캐싱 서버는 자신의 주위에 있는 부모 노드(parents node), 자식 노드(children node) 및 형제 노드(siblings node)들과 상호 협력하면서, 원하는 문서를 가진 가장 가까운 노드로부터 전송 받도록 하고 있다. 그러나 이 방식에서는 캐싱 서버가 문서를 찾기 위하여 모든 주변 노드들로 요구 패킷을 보내야 하기 때문에 네트워크 트래픽을 증가시키는 단점이 있다. CRISP[3]는 그러한 문제를 보완한 방법으로서, 통합 디렉토리(global directory)를 두고 있다. 이 경우에 캐싱 서버는 그 디렉토리를 가진 서버와 한 개의 패킷만 교환하면 문서가 있는 위치를 찾을 수 있다. 그러나 이 방식에서는 통합 디렉토리를 가진 서버가 병목이 될 수 있고, 사용자 컴퓨터와의 거리가 먼 경우에는 지연이 길어진다. 따라서 Gadde et al.[4]은 디렉토리를 분할하여 여러 개의 서버들에 분산 저장하는 방법과 디렉토리의 주요 부분은 여러 서버들에 중복 저장하는 방법으로 그러한 문제에 대한 해결책을 제시하였다. Texas 대학 연구팀[5]은 액세스하려는

문서가 저장된 가장 가까운 캐쉬에 대한 정보를 저장하는 힌트 캐쉬(hint cache)를 이용하는 방법과 푸쉬(push) 동작을 통해 상위 계층에서의 캐쉬 적중을 하위 계층의 프록시 서버에서의 캐쉬 적중으로 바꾸어줌으로써 응답 시간을 줄이는 방법을 제안하였다. 그런데 이 방법은 상위 계층의 캐쉬에 있던 정보가 다른 정보의 유입 시에 용량의 한계로 인해 지워지는 경우에는 빈번히 액세스되는 정보라도 하위 계층 캐쉬 미스 발생에 대해 보완해주지 못한다.

E. P. Markatos [6]는 단일 웹 서버에서 캐쉬 크기와 캐싱 가능한 문서 크기에 따른 성능 향상을 분석하였다. North Dakota 주립대학의 연구팀[7]은 웹 문서들에 대한 액세스 패턴을 이용한 정적 캐싱(static caching) 방식을 제안하였는데, 이 방식은 일정 기간 동안 수집한 요구 로그 파일(request log file)을 분석하여 캐쉬에 교체 저장할 문서들을 결정한다. 그리고, UCLA 연구팀[8]은 액세스 빈도와 확률에 근거하여 동적으로 멀티캐스팅 그룹을 생성함으로써 캐싱된 웹 문서를 서비스 받는 동적 캐싱(dynamic caching) 방식을 제안하였다. 이 방법은 확률 정보를 계속 수집해야 한다는 점과 실제 상황에 대처하는데 한계가 있다는 문제점이 있다.

웹 캐싱은 광대역 통신망의 트래픽을 감소시키고, 통신망 혼잡(congestion)을 줄임으로써 통신망 대역폭 요구를 줄이고, 빈번히 액세스되는 문서를 사용자 컴퓨터에 가깝게 캐싱함으로써 웹 서버의 부하를 감소시키며 액세스 지연을 최소화시킨다. 인터넷상의 프록시 캐쉬들은 <그림 1>과 같이 상호 협력함으로써 그러한 효과를 더욱 배가시킬 수 있다[1, 2, 9]. 그런데 인터넷을 구성하는 통신망(LAN 혹은 WAN)들의 대역폭과 그들을 링크 해주는 프록시 서버들의 저장 장치 용량은 한정되어 있다. 따라서 효율적인 교체 알고리즘을 이용하여 프록시 캐쉬들의 적중률을 높여 웹 서버의 부하와 통신망의 트래픽을 최소화시킴으로써 인터넷 서비스 응답 시간을 줄이는 것은 필수적으로 개발되어야 할 기술이다.



<그림 1> 프록시 캐쉬들 간의 상호협력 구조

본 연구에서는 앞에서 살펴보았던 관련 연구들에서도 고려한 상호 협력적인 계층적 프록시 캐쉬 구조에서 인터넷 서비스 향상을 위한 최적의 캐쉬 교체 방식을 제안하는 것을 목표로 하였다. 이를 위하여 본 연구에서는 계층적 프록시 캐싱 환경에서 하위 캐쉬에서 제거된 정보를 상위 캐쉬에 일단 저장함으로써 적중률을 향상시키는 방법으로서 지연 저장 방식(delayed-store scheme)을 제안하였다. 이 방식은 앞에서 살펴본 Texas 대학 연구팀[5]의 문제점을 보완하여 상위 계층의 캐쉬 적중률을 높여주게 되는데, 그 이유는 하위 계층에 저장되었던 정보들이 상위 계층의 캐쉬에서 교체되어 지워진 경우에 다시 적재될 수 있기 때문이다. 이 제안의 효과를 입증하기 위하여 본 연구에서는 실제 웹 트레이스(web trace)를 이용한 시뮬레이션을 통하여 캐쉬 적중률이 향상되는지 분석하였다. 또한 이 과정에서 계층적 프록시 캐쉬 구조에 다양한 캐쉬 교체 방식들이 사용되었을 때의 적중률을 분석하여 비교하였다.

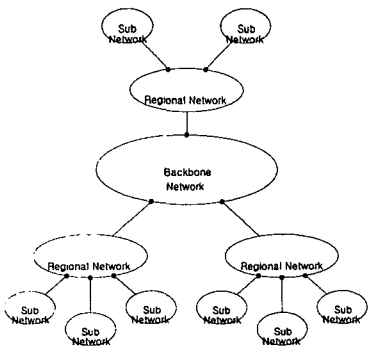
본 논문의 구성은 다음과 같다. 먼저 2장에서는 계층적 프록시 서버 구조에 대한 시스템 모델을 정의하고, 그에 따른 소프트웨어 시뮬레이터의 개발에 대하여 설명하였다. 3장에서는 시뮬레이션을 위한 작업부하의 생성에 대하여 기술하였고, 제4장에서는 지연 저장 방식에 대한 시뮬레이션 및 분석 결과를 살펴본 후에 마지막으로 제5장에서 결론을 맺었다.

2. 시뮬레이터 설계 및 구현

2.1 시스템 모델

본 연구에서는 계층적 분산 프록시 캐쉬 구조에서 캐쉬 교체 알고리즘의 성능을 분석하기 위한 소프트웨어 시뮬레이터(이하 시뮬레이터라 함)를 시뮬레이션 전용 언어인 AweSim v. 2.0 [10]을 이용하여 개발하였다. 본 연구의 목적은 인터넷 서비스 응답 시간의 개선을 위한 새로운 캐쉬 교체 알고리즘인 지연저장 방식을 제안하고, 그 효과를 캐쉬 적중률을 통하여 입증하는 것이다. 따라서 시뮬레이터 구현에서 프록시 서버에서의 캐쉬 동작과 각 셀의 흐름 및 통신 트래픽에 따른 지연 시간 등은 세부적으로 반영함으로써 캐쉬 적중률과 응답 시간을 정확히 분석할 수 있도록 시뮬레이션의 세부 정도(degree of detail)를 조정하였다.

인터넷을 지원하는 통신망의 계층적 구성을 개략적으로 표현하면 <그림 2>와 같다 [1]. 계층의 최상위 망을 백본망, 그 다음을 지역망, 그리고 사용자 컴퓨터들이 실제 접속되는 최하위 망을 서브망이라고 부른다. 서브망에 프록시 서버들이 접속되고, 사용자 컴퓨터들은 그 서버들 중의 하나를 통해 외부와 접속하게 된다. 그런데 프록시 서버가 내부 기억장치 혹은 디스크를 이용하여 캐싱 기능을 가지게 되면, 계층적 캐쉬 구조가 형성되는 것이다. 이러한 계층은 웹 서버와 사용자 사이가 아주 먼 경우에는 더 많은 단계의 통신망 계층들과 프록시 서버들로 이루어지게 될 것이다.

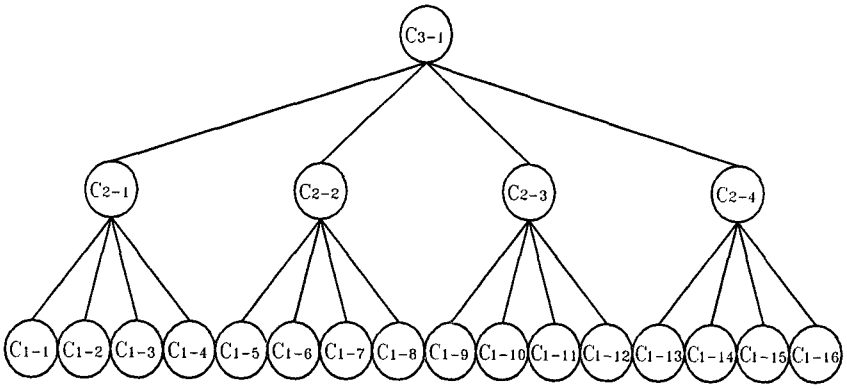


<그림 2> 통신망의 계층적 구성도

사용자 컴퓨터에서 어떤 웹 문서를 요구하면, 그 요구는 일단 서브망의 프록시 서버로 보내진다. 만약 그 프록시 서버에서 문서가 발견되지 않는다면, 상위 계층인 지역망에 접속된 프록시 서버로 요구를 보낸다. 그 문서가 거기서도 발견되지 않는다면 최상위의 백본망에 접속된 서버로 요구를 보내는데, 여기에도 그 문서가 없다면 원래의 웹 서버로 요구를 보내게 된다. 요청된 문서가 프록시 캐쉬 혹은 웹 서버로부터 읽혀지면, 그것은 계층을 따라 전송되면서 사용자 컴퓨터로 보내며, 그 과정에서 각 프록시 캐쉬에 그 문서의 복사본이 저장된다. 그렇게 하면, 그 이후에 그 문서에 대한 다른 요구가 발생하는 경우에 캐싱된 복사본을 읽어서 신속히 응답할 수 있게 된다.

2.2 시뮬레이터의 구현

본 연구에서 사용한 시뮬레이션 모델의 네트워크 구조는 유사한 다른 연구들[1, 4]에서도 사용된 <그림 3>과 같은 3-레벨 4진 트리(3-level 4-ary tree) 형태이다. 각 프록시 서버에는 네 개씩의 하위 프록시 서버들이 접속되며, 하위 프록시 캐쉬들에 캐싱된 웹 문서들에 대한 정보를 가지고 있는 디렉토리를 포함하고 있다. 그리고 모든 사용자 컴퓨터들은 최하위 레벨(레벨 1)에 연결되고, 각 노드의 스위치는 ATM 스위치, 스위치간 링크들은 모두 전송 속도 155Mbps의 STM-1 광케이블, 그리고 캐쉬 검사 시간은 10ms인 것으로 각각 가정하였으며, 디스크 액세스 시간은 고려하지 않았다. 최상위 레벨의 캐쉬 미스가 발생한 경우, 최상위 레벨의 프록시 서버에서 원래 웹 서버의 웹 문서 요구 전송 시간과 원래 웹 서버에서 최상위 레벨의 프록시 서버로의 웹 문서 응답 시간도 각각 10ms로 가정하였다. 본 연구에서는 캐쉬 적중률을 이용하여 성능

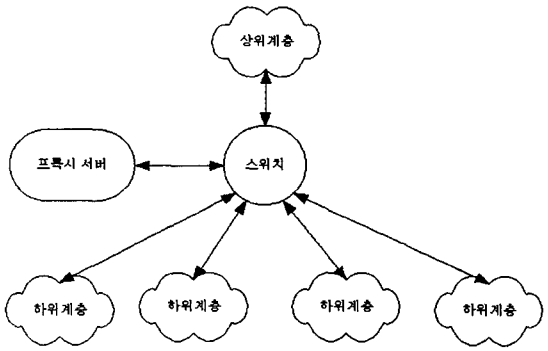


<그림 3> 3-레벨 4진 트리 구조로 표현된
계층적 인터넷망 모델

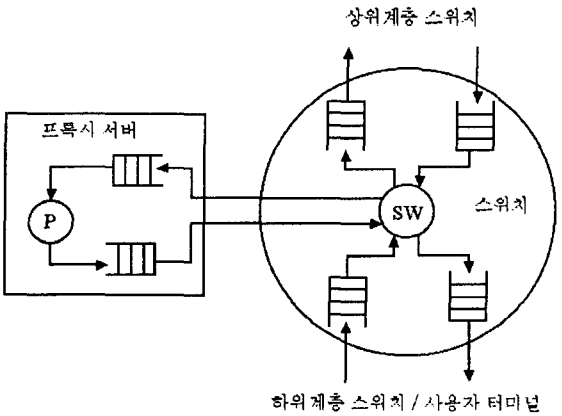
을 분석하기 때문에 이 시간들은 큰 의미를 가지지 않는다.

본 논문에서 사용한 시뮬레이션 모델의 네트워크 구조는 앞에서 언급한 바오 같이 3-레벨 4진 트리 구조인데, 이 구조의 각 노드들은 프록시 서버와 접속된 스위칭 노드들이다. <그림 4>와 같이 각 스위칭 노드에는 프록시 서버가 위치하고, 각 프록시 서버는 하위 노드들로부터 들어오는 요구에 대하여 캐쉬된 웹 문서들을 서비스한다. 사용자 컴퓨터들은 트리의 최하위 노드들에만 연결되어 있고, 중간 노드들에는 연결되지 않은 것으로 가정하였다.

시뮬레이터 개발을 위하여 <그림 4>의 구조를 큐잉 모델로 다시 표현하면 <그림 5>와 같아진다. 각 ATM 스위치의 내부는 입출력 버퍼들과 스위칭 회로만으로 간략히 모델링 하였으며, 이들이 <그림 3>에서와 같이 상호 접속되어 트



<그림 4> 스위칭 노드의 구조



<그림 5> ATM 스위치와 프록시 서버의
큐잉 모델

래픽 전송 동작을 시뮬레이션하게 된다.

고정 길이 셀을 라우팅하는 ATM 교환기는 스위칭 과정에서 소프트웨어가 개입되지 않는, 즉 순수한 하드웨어 교환 기술을 이용하고 있다. ATM 교환기는 입출력 회선 대응부, ATM 셀프 라우팅 스위치, 그리고 제어부로 구성되어 있다. 입력 회선 대응부는 입력되는 회선과 ATM 교환기 사이의 인터페이스 기능을 수행하며, 셀의 유량을 감시한다. 셀의 유량 감시는 UPC(Usage Parameter Control; 사용량 파라미터 제어)에서 행해진다. OAM(Operation, Admission and Maintenance; 보수 운용 관리)부는 전송로나 교환기가 정상적으로 작동하고 있는지를 검색하거나, 셀 손실 혹은 동작 오류 여부 등의 성능을 감시한다. 헤드 변환부(HCV; Head Converter)는 전송로 상에서 사용된 가상 채널 식별자를 스위치 부 및 출력 전송로에서 이용할 가상 채널 식별자로 변환하는 역할을 한다. ATM 셀프 라우팅 스위치는 셀의 헤드 정보를 바탕으로 수신처의 출력 전송로에 라우팅하고, 제어부에서는 헤드의 변환된 값을 보관하고 ATM 교환기의 고장이나 장치를 관리한다.

본 연구는 캐쉬 적중률을 중심으로 한 성능 분석에 초점을 맞추었기 때문에 ATM 스위치 내의 UPC, OAM, HCV의 모든 기능들이 정상적으로 동작한다고 가정하고, 셀이 입력 회선을 따라 ATM 스위치에 입력되어 정해진 스위칭 시간 후에 목적지 버퍼로 출력되도록 스위치 부에 대한 시뮬레이션은 간략화 시켰다. 본 연구에서는 스위치의 내부 회로 동작에 의한 지연 시간, 즉 스위칭 시간은 ForeRunner ATM Backbone switch인 ASX-200BX 모델[11]의 10μsec와 ASX-200BXE의 15μsec의 중간 값인 12μsec로 가정하였다.

3. 작업부하의 분석

시뮬레이션에서 작업 부하는 성능 측정 결과에 매우 중요한 영향을 미치기 때문에 각 실험에 대하여 적절하게 설정하여 사용해야 한다. 본 연구에서는 버클리 대학(UCB)에서 수집한 웹 트레이스들[12]을 이용하였다. 이 웹 트레이스들은 UCB에서 제공하는 Home IP service에서 수집된 HTTP 트레이스이다. Home IP는 2.4kb/s, 9.6kb/s, 28.8kb/s 유선 모뎀이나 Metricom Ricochet(약 20-30kb/s) 무선 모뎀을 이용한 PPP/SLIP IP 연결을 제공한다. 이들 트레이스는 다음과 같은 정보를 가지고 있다 : 수집 기간, 요구의 수 및 클라이언트의 수, 요구 생성 시간, 서버 응답의 첫 바이트가 도착한 시간, 서버 응답의 마지막 바이트가 도착한 시간, 클라이언트의 IP 주소, 클라이언트의 포트 번호, 서버의 IP 주소, 서버의 포트 번호, 응답 데이터의 길이, 그리고 요구의 URL. 본 연구에서는 이 정보들 중에서 요구 생성 시간, 서버의 IP 주소, URL, 그리고 응답 데이터의 길이만을 추출하여 사용하였다. 시뮬레이션에서는 그와 같이 수집된 트레이스에 근거하여 추정한 확률 분포 함수에 따라 요구를 발생하여 사용하였다. <표 1>의 첫 번째 트레이스는 시뮬레이터 검증을 위해 사용하였으며, 실제 연구에서는 두 번째 트레이스를 주로 사용하였다.

<표 2> UCB 트레이스들

	UCB(1)	UCB(2)
수집 기간	96.11.17 16:47:06 - 96.11.17 20:47:06	96.11.6 12:49:59 - 96.11.9 20:47:01
총 요구수	95,768 개	1,703,835 개
클라이언트수	916 개	5257 개
총전송바이트	640,238,759	12,573,824,026
참조 URL 수	61,354 개	697,139 개

이 트레이스를 분석하여 요구간 발생 시간 간격(inter-request time interval)을 구한 결과, 평균이 150.35ms 이고 표준 편차가 168.7 인 것으로 확인되었다. 이것을 이용하여 분산 계수를 구하면 1.12204이다. 따라서 요구 발생 시간 간격이 지수 분포(exponential distribution)를 가지는 것으로 가정할 수 있다.

UCB 트레이스에서 전체 참조된 문서의 10% 정도가 전체 요구의 51.63%를 차지하였다. 이 결과는 영어 텍스트에 출현한 단어들의 분포를 나타내는 Zipf의 확률 분포[14]와 유사하다. 또한 평균 전송 바이트는 약 6~8 Kbytes 이었으며, 그 중수(median)는 약 2 Kbytes 이었다. 이와 같은 특성은 다른 웹 트레이스 분석 연구들에서도 유사하였다[15, 16].

사용자들로부터 요구되는 문서들의 분포는 대부분 Zipf 분포를 따른다. 웹 트레이스 분석 연구에 따르면, 문서 요구의 90 % 정도가 웹 서버의 전체 문서 중 10~25 % 의 문서에 집중된다. 그리고 요구간 시간 간격은 지수 분포를 따른다. 문서 요구의 90 % 정도는 웹 서버에서 성공적으로 응답을 해준다. 또한 요구된 웹 문서는 대부분 HTML 문서이거나 이미지 문서이다. 웹 문서의 크기는 대부분 작아서 중수(median)가 2 Kbyte 정도이며, 평균 크기도 수 십 Kbyte를 넘지 않는다.

4. 시뮬레이션 및 결과 분석

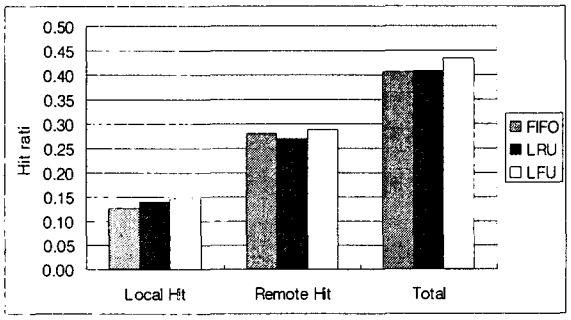
본 장에서는 제2장에서 설명한 소프트웨어 시뮬레이터를 이용하여 기존의 교체 방식들과 푸쉬 알고리즘이 적용된 경우의 캐쉬 적중률을 비교하였다. 그리고 본 논문에서 제안하는 지연 저장(delayed-store) 방식을 이용한 경우의 성능 향상을 캐쉬 적중률을 중심으로 분석하였다. 마지막으로 캐쉬 크기를 다양하게 변화시키면서 캐쉬 적중률의 변화를 분석하였다. 여기서 모든 시뮬레이션은 800만 시뮬레이션 시간까지 수행하였으며, 캐쉬 크기에 따라 100만 또는 500만 시뮬레이션 시간부터 각 프록시 서버에서의 캐쉬 적중과 캐쉬 미스의 비율을 수집하였다. 시뮬레이션 시작 시간부터 100만 또는 500만 시뮬레이션 시간까지는 캐쉬를 warm state로 만드는데 필요한 시간에 해당하므로 그 시간 동안의 데이터는 수집하지 않았다.

4.1 캐쉬 교체 방식에 따른 캐쉬 적중률

캐쉬의 용량에는 한계가 있기 때문에 캐쉬가 이미 채워져 있을 때는 새로운 데이터를 저장하기 전에 교체될 데이터를 선택해야 한다. 교체되는 데이터는 캐쉬 교체 알고리즘에 의해 선택된다. 대표적인 알고리즘으로는 LFU(Least Frequently Used), LRU(Least Recently Used), FIFO(First-In First-Out) 등이 있다. 이 알고리즘들은 파일 시스템에서는 좋은 성능을 발휘하지만, 웹 환경에는 적합하지 않다. 그 주요 이유는 웹 문서들의 크기가 매우 다양하며, 웹 문서를 요구하는 사용자들은 항상 요구한 문서 전체를 원하기 때문이다. 반면에, 파일 시스템은 고정된 크기의 데이터 블록들을 다룬다. 또한 파일 시스템은 읽기와 쓰기 요구 모두를 다루지만, 웹 문서는 대부분 읽기 전용이며 사용자들에 의해 수정되지는 않는 특징을 가지고 있다. 또한, 사용자들이 요구하는 웹 문서들 중에는 캐쉬이 가능하지 않은 문서들도 있다.

그와 같은 특성 때문에 웹 서버나 프록시 서버의 캐쉬에 적합한 교체 알고리즘을 개발하기 위한 연구들이 진행되었다[17, 18, 19, 20]. LRU-SIZE [17]는 캐쉬에 저장된 문서 중 크기가 가장 큰 문서를 먼저 교체한다. 만약 같은 크기의 문서들이 있으면 LRU를 적용한다. LRU-MIN [18]은 새로 캐쉬될 문서보다 더 크거나 같은 문서들을 LRU 방식으로 교체한다. LRU-threshold [18]는 캐쉬에 저장 가능한 문서의 크기를 제한하는 threshold를 두고, 교체 방식은 LRU 방식을 사용한다.

본 연구에서 첫 번째 시뮬레이션은 캐쉬 저장 가능한 한계 크기인 threshold를 64 KBytes로 하고, 새로 캐쉬에 저장될 문서보다 크거나 같은 문서들 중에서 교체 방식에 따라 교체하는 MIN 방식을 적용하여 LFU, LRU, 그리고 FIFO의 계층적 프록시 캐쉬 구조에서의 평균 캐쉬 적중률을 비교하였다. 그 시뮬레이션 결과는 <그림 6>과 같다.



<그림 6> 교체 방식에 따른 캐쉬 적중률

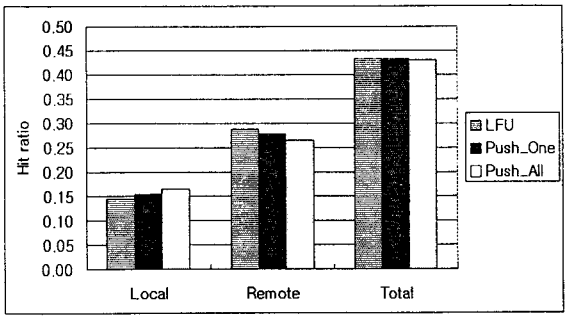
그림에서 사용자 컴퓨터와 직접 연결된 지역 프록시 캐쉬에서 캐쉬 적중이 일어났을 경우를 지역 캐쉬 적중(local cache hit)이라 하며, 그 외의 프록시 캐쉬에서 적중이 일어났을 경우는 원격 캐쉬 적중(remote cache hit)이라 한다. 결과에 따르면, 지역 캐쉬 적중률은 FIFO 방식이 0.13, LRU 방식이 0.14이었고, LFU 방식이 0.15로 가장 높았다. 원격 캐쉬 적중률(remote cache hit ratio)은 LRU 방식이 0.27, FIFO 방식이 0.28이었고, LFU 방식이 0.29로 가장 높았다. 결과적으로, 지역 캐쉬 적중률과 원격 캐쉬 적중률을 모두 포함하는 전체 캐쉬 적중률(total cache hit ratio)은 LRU와 FIFO 방식이 모두 0.41로 거의 같았고, LFU 방식을 사용하였을 경우의 전체 캐쉬 적중률이 0.43으로 가장 높은 것으로 나타났다. 즉, 캐쉬 교체 방식으로 LFU를 사용하면 지역 캐쉬 적중률뿐만 아니라 전체 캐쉬 적중률이 높아져 서비스 응답 시간이 가장 짧아진다는 것을 알 수 있다. 이것은 웹 서비스에서는 액세스의 시간적 패턴보다는 빈도가 적중률을 높이는데 더 큰 영향을 미친다는 일반적인 추측이 맞다는 것을 보여주는 결과이다.

4.2 푸쉬 알고리즘에 따른 캐쉬 적중률

Texas 대학 연구팀 [5]에서 제안한 푸쉬(push) 방식은 멀리 떨어져 있는 캐쉬에서보다는 가까이 있는 캐쉬에서 원하는 문서를 서비스 받는 것이 더 빠르다는 기본적인 개념에서 시작하였다. 푸쉬 알고리즘은 크게 두 가지로 나눌 수 있다. 첫째, push-on-update는 문서가 수정되었을 경우에 이전의 문서를 갖고 있던 프록시 캐쉬들에 새로운 문서를 푸쉬하는 기법이다. 두 번째는 계층적 프록시 캐쉬 구조에서 빈번히 액세스되는 문서를 동적으로 캐쉬에 분산시키는 push-shared 알고리즘이다. 이 방식은 계층적 프록시 캐쉬 구조에서 원격 캐쉬 적중이 일어났을 경우, 그 요구된 문서를 적당한 프록시 캐쉬들에 푸쉬한다. 이 방식은 푸쉬할 프록시 캐쉬를 선택하는 방법에 따라 다음과 같이 두 가지로 나눌 수 있다.

(1) Push-1 방식 : 원격 캐쉬 적중이 일어났을 때, 각 서브 트리의 임의의 한 노드에 그 웹 문서를 푸쉬.
(2) Push-all 방식 : 원격 캐쉬 적중이 일어났을 경우, 각 서브 트리의 모든 노드들로 푸쉬.

본 연구에서는 시뮬레이션을 이용하여 push-shared 알고리즘에서 위의 두 가지 방식을 각각 적용하였을 때의 평균 캐쉬 적중률을 분석하였다. 그 시뮬레이션 결과는 <그림 7>과 같다. 시뮬레이션에서 캐쉬 교체는 LFU 방식을 기반으로 하였으며, 각 프록시 캐쉬 크기는 각각 32 Mbytes이다. <그림 7>에서 보는 바와 같이 푸쉬 알고리즘에 따른 전체 캐쉬 적중률의 향상은 거의 없었다. 푸쉬를 적용하지 않은 경우와 적용한 경우 모두 0.43으로 캐쉬 적중률은 같았다. 그러나 push-1과 push-all 방식은 웹 문서를 사용자 컴퓨터와 가까운 최하위 프록시 서버들에 푸쉬함으로써 지역 캐쉬 적중률을 향상시켰다. Push-all의 경우 지역캐쉬 적중률이 0.17로, 푸쉬 알고리즘을 적용하지 않았을 경우 보다 약 13% 정도 향상되었다. Push-1의 경우에도 지역 캐쉬 적중률이 약간 향상되었다. 이렇게 지역 캐쉬 적중률이 높아지면 서비스 응답 시간이 짧아지게 되는 것이다.



<그림 7> 푸쉬 알고리즘에 따른 캐쉬 적중률

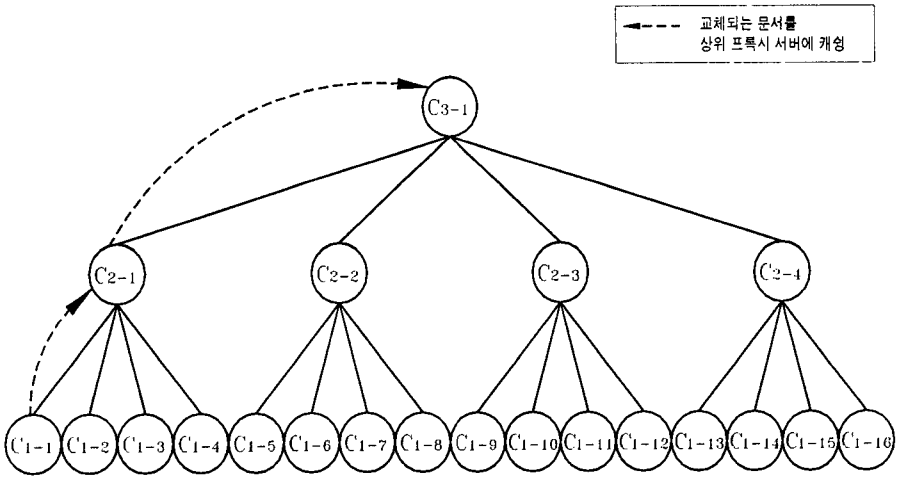
그러나, 푸쉬 알고리즘은 두 가지 문제점이 있다. 첫째, 다른 프록시 서버로 웹 문서를 푸쉬하기 위하여 통신망 대역폭을 소비하는 것이다. 둘째, 이렇게 푸쉬된 웹 문서가 다른 사용자들로부터 요구가 없을 지도 모르는 불필요한 문서일 수도 있다는 점이다.

4.3 지연 저장 방식을 이용한 성능 향상

본 논문에서 제안하는 지연 저장 방식(delayed-store scheme)은 계층적 분산 프록시 캐쉬 구조의 특성을 이용하는 것이다. 지연 저장 방식의 목적은 새로운 웹 문서에 의해 교체되는 이전의 문서를 다시 활용하여 캐쉬 적중률을 높이는데 있다. 이 방식은 <그림 8>에서와 같이 하위 프록시 서버에서 캐쉬 교체 알고리즘에 의하여 교체되는 웹 문서를 완전히 제거하지 않고 그 상위, 즉 부모 프록시 서버(parent proxy server)에 다시 캐싱하는 것이다. 이렇게 하면 일단 교체된 후에도 그 프록시 서버 혹은 같은 레벨의 형제 프록시 서버(sibling proxy server)가 그 문서를 다시 요구하는 경우에 원격 캐쉬 적중률을 높일 수 있게 되는 것이다. 1장에서 살펴본 다른 방식들에서는 부모 서버에 있던 문서가 교체되어 지워지는 경우에는 하위 서버에서 미스되는 요구는 모두 원래 서버로 보내져야 하는 점에서 본 연구의 제안과는 다르다. 이 절에서는 4.1절과 4.2절에서 분석한 캐쉬 교체 방식들과 푸싱 방식들에 지연 저장 방식을 추가하여 시뮬레이션한 결과들을 분석하였다.

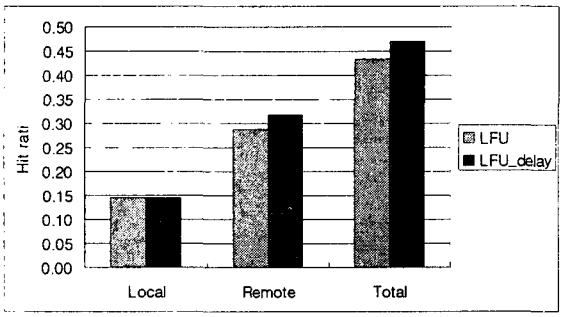
4.3.1 지연 저장 방식을 적용한 경우의 캐쉬 적중률

<그림 9>는 4.1절에서의 결과와 같이 캐쉬 교체 알고리즘들 중에서 가장 높은 적중률을 보여준 LFU 방식에 지연 저장 방식을 추가하였을 때의 성능을 보여 주고 있다. 시뮬레이션에서 모든 프록시 서버들의 캐쉬 크기는 32 Mbytes로



<그림 8> 계층적 프록시 캐쉬 구조에서의 지연 저장 방식

고정시켰다. 시뮬레이션 결과를 보면, 지역 캐쉬 적중률은 0.15로 변함이 없었다. 그러나 원격 캐쉬 적중률은 0.29에서 0.32로 약 10.34 % 향상되었다. 따라서 전체 캐쉬 적중률도 0.43에서 0.47로 약 9.3 % 상승하였다.

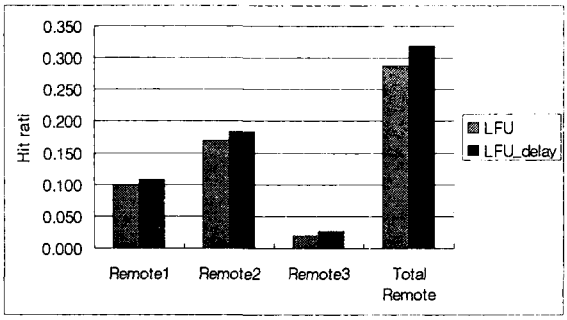


<그림 9> 지연 저장 방식을 추가한 경우의 적중률

원격 캐쉬 적중은 캐쉬 적중까지 요구들이 통과하는 링크의 수에 따라 세 가지로 구분될 수 있다. 먼저, 사용자 컴퓨터와 직접 연결된 최하위 레벨의 프록시 서버에서 캐쉬 적중이 실패하고, 그 부모 프록시 서버에서 적중이 일어난, 즉 캐쉬 적중까지 한 홉(hop)이 걸린 경우를 Remote1이라 부르기로 한다. Remote2는 캐쉬 적중까지 두 홉이 걸린 경우를 말하며, 마지막으로 캐쉬 적중까지 세 홉이 걸린 경우를 Remote3이라 하자. 서비스 응답 시간은 지역 캐쉬 적중일 경우가 가장 짧으며, 다음으로 Remote1, Remote2, Remote3 순이 된다. 가장 서비스 응답 시간이 가장 느린 경우는 모든 프록시 서버에서 캐쉬 적중이 실패하여 원래의 웹 서버(original web server)로부터 문서를 읽어오는 경우이다.

<그림 10>은 이와 같이 원격 캐쉬 적중을 홉 수에 따라 세분하였을 때의 적중률을 분석한 결과를 보여주고 있다. 그림의 결과를 보면, Remote1은 0.10에서 0.11로 향상되었고, Remote2는 0.17에서 0.18로 각각 향상되었다. 그리고 Remote3도 지연 저장 방식을 추가한 결과 0.02에서 0.03으로 향상되어, 전체 원격 캐쉬 적중률이 0.29에서 0.32로 약 10.34 % 향상되었다.

연구 결과들을 종합해보면, LFU 교체 방식에 지연 저장 방식을 추가한 경우에 지역 캐쉬 적중률은 지연 저장 방식을 추가하지 않았을 때와 거의 차이가 없었지만, 원격 캐쉬 적중률은 10 % 내지 29 % 정도가 상승하였다. 결과적으로 전체 캐쉬 적중률이 향상됨으로써 인터넷 서비스 시간이 단축될 수 있는 것을 알 수 있었다.

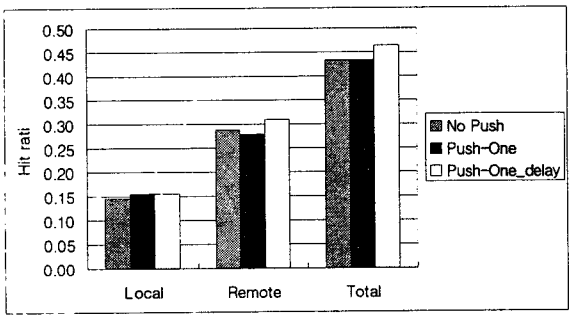


<그림 10> 홉 수에 따른 원격 캐쉬 적중률

4.3.2 푸쉬 방식에 지연 저장 방식을 추가한 경우의 캐쉬 적중률

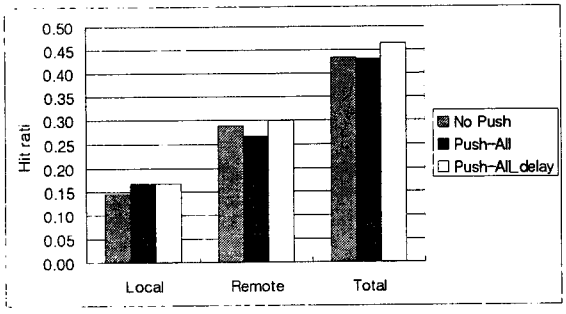
여기서는 4.2절에서 설명한 푸쉬 알고리즘들에 지연 저장 방식을 추가하여 시뮬레이션한 결과를 분석하였다. <그림 11>과 <그림 12>는 4.2절에서 설명한 푸쉬 방식들 중에서 push-1과 push-all에 지연 저장 방식을 적용하였을 경우의 성능 향상을 보여주고 있다. 시뮬레이션에서 각 프록시 서버의 캐쉬 크기는 32 Mbytes이고, 캐쉬 교체 방식은 LFU 방식을 기반으로 하였다.

Push-1에 지연 저장 방식을 추가한 경우에는 <그림 11>에서 보는 바와 같이 지역 캐쉬 적중률은 0.16으로 변함이 없었다. 그러나 원격 캐쉬 적중률은 0.28에서 0.31로 10.71 % 향상되었다. 따라서 전체 캐쉬 적중률도 0.43에서 0.46으로 약 6.98 % 향상되었다.



<그림 11> 지연 저장 방식을 추가한 경우의 적중률 (push-1)

지역 캐쉬 적중률이 가장 높은 push-all에 지연 저장 방식을 추가한 경우에는 <그림 12>에서와 같이 지역 캐쉬 적중률은 0.17로 변함이 없었다. 그러나 원격 캐쉬 적중률은 위의 경우와 마찬가지로 0.27에서 0.30으로 약 11.11 % 향상되었다. 전체 캐쉬 적중률도 0.43에서 0.46으로 약 6.98 % 향상되었다.



<그림 12> 지연 저장 방식을 추가한 경우의 적중률 (push-all)

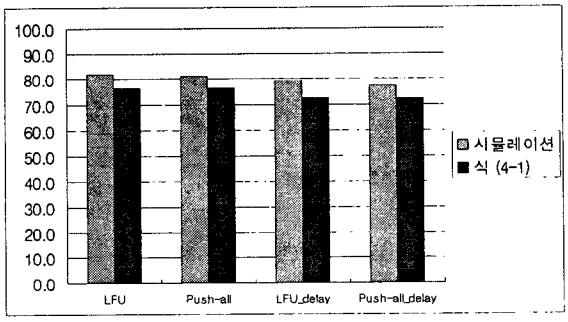
4.3.3 지연 저장 방식 추가에 따른 응답 시간의 단축

이 절에서는 지연 저장 방식의 추가에 따른 응답 시간을 분석하였다. 각 프록시 캐쉬의 크기는 32 MBytes이며, 교체 방식은 LFU 방식을 기반으로 하였다. 캐쉬 적중률에 따른 평균 서비스 응답시간 T_{acc}는 다음의 식으로 구할 수 있다.

$$T_{acc} = H_L T_C + H_R(3.5T_N + 2.5T_C) + (1 - H_L - H_R)(4T_N + 3T_C + T_M) \quad (1)$$

식 (1)에서 H_L은 지역 캐쉬 적중률, H_R은 원격 캐쉬 적중률이다. 그리고 T_C는 캐쉬 액세스 시간, T_N은 통신망에서 소비되는 시간이며, T_M은 모든 프록시 캐쉬에서 캐쉬 적중이 실패하여 원래의 웹서버에서 서비스 받는데 걸리는 시간이다. 캐쉬 적중 실패시는 통신망 링크를 네 번 통과하게 되고, 캐쉬는 세 번을 액세스하게 된다.

이 시뮬레이션에서는 지연 저장 방식과 푸쉬 방식을 적용하지 않은 일반적인 LFU 교체 방식을 이용하였을 경우와 지연 저장 방식을 추가한 경우, push-all 방식만 적용한 방식, 그리고 push-all 방식과 지연 저장 방식을 모두 적용한 경우에 대하여 응답 시간을 분석하였다. 여기서 T_C는 10ms로 가정하였다. 통신망에서 지연 시간 T_N은 웹 문서의 평균 크기를 고려하여 0.053ms로, T_M은 100ms로 가정하였다. 시뮬레이션 결과에 따른 서비스 응답 시간과 식 (1)을 이용하여 구한 응답 시간을 비교해보면 <그림 13>과 같다.



<그림 13> 평균 서비스 응답 시간

그림에서 보는 바와 같이, LFU 방식과 push-all 방식에 지연 저장 방식을 추가함에 따라 평균 서비스 응답 시간이 4 ms 정도 단축된다는 것을 알 수 있다. 식 (1)에 의한 평균 서비

스 응답 시간과 시뮬레이션에서 얻어진 평균 응답 시간에 다소의 차이가 있는 것은 시뮬레이션에서는 서비스 요구와 응답의 큐잉 지연 시간과 서비스되는 웹 문서의 크기에 따른 통신망 지연이 추가되었기 때문이다. 반면에 수식에 의한 계산에서는 그러한 점들은 고려되지 않았다.

5. 결 론

최근 인터넷 사용자 수가 급속히 증가함에 따라 웹 서버에 걸리는 부하와 통신망의 트래픽이 크게 증가하고 있다. 초고속 정보통신망을 비롯한 정보 인프라의 구축이 확대되고 고성능 개인용 컴퓨터가 보급이 확산됨에 따라 멀티미디어와 같은 다양한 서비스에 대한 사용자의 요구가 증대되고, 인터넷 저변 인구가 크게 늘어나고 있다. 이와 같은 사용자의 요구를 수용하고 인터넷 서비스를 개선하기 위하여 vBNS, APAN과 같은 초고속 정보통신망의 개발과 다양한 프로토콜에 대한 연구들이 활발히 진행되고 있다.

본 연구에서는 상호협력적인 계층적 분산 프록시 캐쉬 구조에서 웹 서버의 부하와 통신망의 트래픽을 줄여 인터넷 서비스 응답 시간을 단축시키기 위해 캐쉬 적중률을 높이는 방법을 연구하였다. 이를 위하여 3레벨 4진 트리 구조의 시뮬레이션 모델을 구성하고, 실제 웹 트레이스를 이용하여 이산 사건 시뮬레이션을 수행함으로써 다양한 환경에서의 캐쉬 적중률을 분석하였다. 그리고 본 논문에서 제안하는 지연 저장 방식을 적용하여 캐쉬 적중률 향상을 시도하였다.

시뮬레이션 결과에 따르면, 계층적 분산 프록시 캐쉬 구조에서 지연 저장 방식을 적용하였을 경우에 지역 캐쉬 적중률은 거의 변함이 없었지만, 원격 캐쉬 적중률이 향상되어 전체 캐쉬 적중률이 높아졌다. 캐쉬 교체 알고리즘을 중에서 LFU 방식을 적용한 경우에 가장 높은 캐쉬 적중률을 보였다. 그리고 LFU 방식에 지연 저장 방식을 추가 적용하였을 때 원격 캐쉬 적중률이 10~29 %가 향상되어 전체 캐쉬 적중률이 10 % 정도가 높아졌다. 지역 캐쉬 적중률을 높여 서비스 응답 시간을 단축시키는 푸 쉬 알고리즘에 지연 저장 방식을 추가한 경우에는 지역 캐쉬 적중률뿐만 아니라 원격 캐쉬 적중률도 향상되어 전체 캐쉬 적중률이 약 7 % 높아졌다. 이 정도의 적중률 향상은 인터넷상에 전송되는 웹 액세스 요구 수가 매우 많고 원래 서버들의 위치가 여러 국가들에 산재해 있는 점을 감안할 때 액세스 시간 및 트래픽 감소에 상당히 큰 효과를 얻을 수 있다고 볼 수 있다. 따라서 본 연구에서 제안한 지연 저장 방식을 사용하면 서비스 응답 시간을 상당히 단축하는 효과를 얻을 수 있다는 것을 확인하였다.

참고 문헌

[1] A. Chankhunthod and M. F. Schwartz, "A hierarchical Internet object cache," In Proceedings of USENIX 1996 Annual Technical Conference, January 1996.

[2] D. Wessels et al. "Squid Internet object cache," <http://squid.nlnar.net/>

[3] S. Gadde, M. Rabinovich, and J. Chase, "Reduce, Reuse, Recycle:An approach to buildiing large Internet ·caches," In Proceedings of 6th Workshop on Hot Topics in Operating Systems(HOTOS-VI), May 1997.

[4] S. Gadde, J. Chase, and M. Rabinovich, "Directory structures for scalable Internet caches," Technical Report CS-1997-18, Duke University Department of Computer Science, November 1997.

[5] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, "Design considerations for distributed caching on the Internet," Technical Report UTCS TR98-04, Department of Computer Science, University of Texas at Austin, October 1998.

[6] E. P. Markatos, "Main Memory Caching of Web Documents," Computer Networks and ISDN Systems, 1996.

[7] I. Tatrainov, V. Sooviev, and A. Rousskov, "Static caching in Web servers," In Proceedings of The Sixth International Conference on Computer Communication and Networks, 1997.

[8] J. Yang, W. Wang, R. Muntz, and J. Wang, "Dynamic Web caching," Department of Computer Science, University of California, LA, November 1998.

[9] J. Wang, "A Survey of Web Caching Schemes for the Internet", Department of Computer Science, Cornell University, 1999.

[10] A. Alan, B. Pritsker, J. J. O'Reilly, and D. K. LaVal, Simulation with Visual SLAM and AweSim

[11] FORE SYSTEM, "ForeRunner ATM Switch Architecture," April 1996.

[12] UC Berkeley Home IP Web Traces, <http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html>

[13] A. Alan, B. Pritsker, Introduction to Simulation and SLAM II, 3rd Edition, Systems Publishing Corporation, 1986.

[14] "Zipf's Law", http://www.elisdsu.edu/courses/fall96/cs660/notes/splayPerf/splayperf.html

[15] M. F. Arlitt and C. L. Willamson, "Internet Web Servers: Workload Characterization and Performance Implications," IEEE/ACM Transactions on Networking, Vol. 5, No. 5, pp. 631-644, October 1997

[16] G. Abdula, E. A. Fox, M. Abrams, and S. Williams, "WWW Proxy Traffic Characterization with Application to Caching," TR-97-03, Virginia Polytechnic Institution and State University, 1997.

[17] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, "Removal Policies in Network Caches for World -Wide Web Documents," Proceedings of Sigcomm'96, 1996.

[18] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching Proxies: Limitations and Potentials," Proceedings of the 4th International WWW Conference, Boston, MA, December 1995.

저자소개

이효일

1997년 연세대학교 문리대학 전산학과 졸업 (이학사)
2000년 연세대학교 대학원 전산학과 졸업 (이학석사)
2000년~현재 삼성SDS 정보기술연구소 연구원
관심분야 : 컴퓨터구조, 시뮬레이션, 3-D 그래픽

김종현

1976년 연세대학교 공과대학 전기공학과 졸업 (공학사)
1981년 연세대학교 대학원 전기공학과 졸업 (공학석사)
1988년 Arizona State University 컴퓨터공학과 졸업 (Ph.D)
1976~82년 국방과학연구소 연구원
1988~90년 한국전자통신연구소 실장
1990년~현재 연세대학교 전산학과 교수
관심분야 : 컴퓨터구조, 시뮬레이션, 인터넷 성능분석