목 차

그림 목차

표 목차

MCC

9851M08

김 숙 향, Sook-Hyang Kim,

A Statistical, Batch and Proxy-Initiated Web Prefetching Scheme for Efficient Internet Bandwidth Usage, **효율적인 인터넷 대역폭 사용을 위한 웹 프리페칭 기법**, **정보통신학과**, 2000, 60P,

Advisor: J. Won-Ki Hong, Text in Korean.

ABSTACT

As the number of World-Wide Web (Web) users grows, Web traffic continues to increase at an exponential rate. Currently, Web traffic is one of the major components of Internet traffic. Also, high bandwidth usage due to Web traffic is observed during peak periods while leaving bandwidth usage idle during off-peak periods. One of the solutions to reduce Web traffic and speed up Web access is through the use of Web caching. Unfortunately, Web caching has limitations for reducing network bandwidth usage during peak periods.

In this thesis, we focus our attention on the use of a prefetching algorithm for reducing bandwidth during peak periods by using off-peak period bandwidth. We propose a statistical, batch, proxy-side prefetching algorithm that improves cache hit rate while only requiring a small amount of storage.

We present simulation results based on Web proxy trace and show that this prefetching algorithm can reduce peak time bandwidth using off-peak bandwidth.

1. 서론

World Wide Web(이하 웹)은 저렴한 가격과 다양하고 흥미 있는 정보를 쉽고 간편하게 찾아볼 수 있다는 장점으로 웹 사용자는 하루가 다르게 증가되고 있다. 웹 사용자의 증가와 함께 웹을 통해 전달되는 데이터 즉, 웹 문서, 그림, 멀티미디어 데이터 등의 크기 또한 빠르게 증가되고 있다[20, 21]. 또한, 최근에는 인터넷 트래픽 (Internet traffic)의 많은 부분이 웹 트래픽 (Web traffic)으로 구성되고 있으며 이러한 웹의 급격한 보급과 각종 멀티미디어 데이터를 포함하는 웹 서비스의 팽창은 네트워크의 병목현상 (bottleneck)을 점점 더 가중시키고 있다[25, 26]. 웹 트래픽을 위한 네트워크 대역폭 (bandwidth)의 사용량을 살펴보면 사용자들의 요청이 많은 peak periods 동안에 대부분의 대역폭을 사용하고 있고, off-peak periods 동안에는 사용하지 않는 대역폭이 존재한다. 그림 1 은 포항공대의 기숙사 중에서 16 개의 subnet 에 대한 2 주 동안 (1999 년 10 월 15 일부터 1999 년 10 월 28 일까지)의 네트워크 트래픽의 하루 평균 대역폭 사용량을 시간대별로 나타낸 것이며, 그림 2 는 동일한 기간 동안에 요청되는 하루 평균 웹 객체의 횟수를 시간대별로 나타낸 것이다.

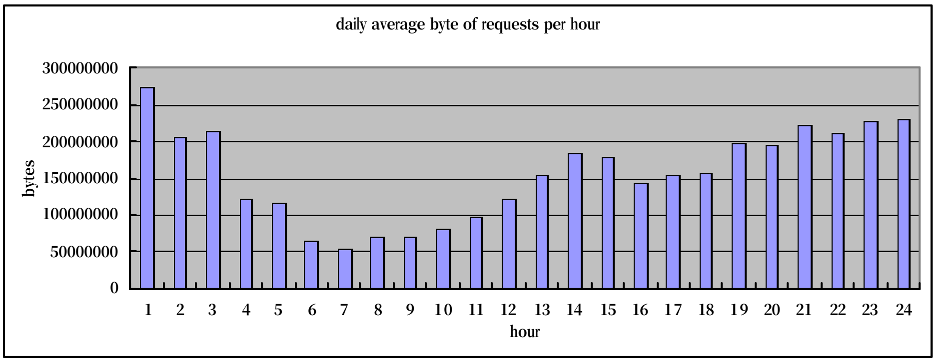

그림 1. 포항공대 기숙사의 시간대별 평균 웹 트래픽량

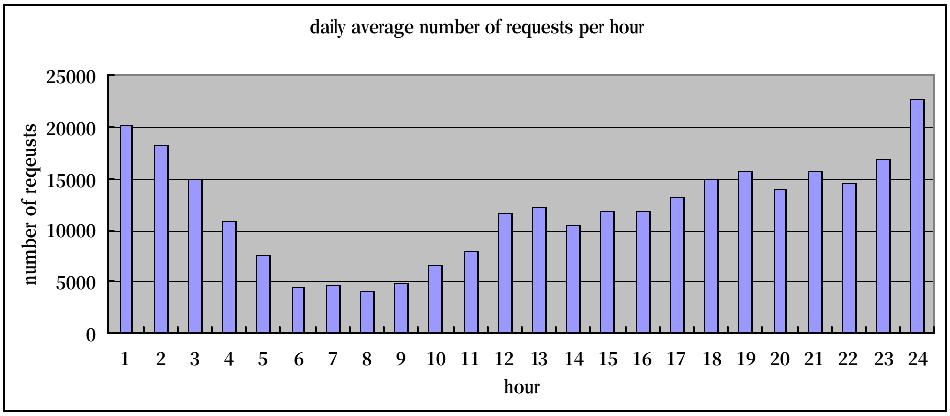

그림 2. 포항공대 기숙사의 시간대별 평균 웹 요청 횟수

하루동안 요청된 웹 트래픽량의 평균 (156624826 byte)을 기준으로 트래픽 발생량이 평균 이상인 시간대를 peak periods 로 정하고 평균 미만인 시간대를 off-peak periods 로 정하면 peak periods 는 14:00 부터 16:00 까지와 18:00 부터 새벽 04:00 까지, 12 시간이며 off-peak periods 는 새벽 04:00 부터 14:00 까지와 16:00 부터 18:00 까지 12 시간이 된다. 그림 1 과 그림 2 에도 나타나 있듯이 peak periods 동안에는 대부분의 네트워크 대역폭이 사용되며 off-peak periods 동안에는 사용되지 않는 대역폭이 발생되는 것을 알 수 있다. 따라서 peak periods 와 off-peak periods 간의 좀더 균형 있는 네트워크 대역폭의 사용이 필요하다.

지금까지 네트워크의 대역폭 소비량을 감소시키고, 검색 지연시간을 줄이기 위해 많은 연구가 이루어졌고 그 해결방안 중 하나가 웹 캐싱(Web caching)이다[10]. 웹 캐싱은 웹의 사용으로 인한 네트워크 트래픽을 줄이고, 말단 사용자들에게 빠른 응답 시간을 제공하기 위한 기술이다[11, 12]. 웹 캐싱은 사용자가 특정한 사이트를 반복적으로 접근하고, 여러 명의 사용자가 같은 사이트에 접근하는 경향으로부터 착안되어졌다. 자주 사용되고 많이 접근되는 정보인 경우 굳이 웹 서버에서 제공받을 필요 없이 최초의 방문을 통해 데이터를 가까운 저장소에 저장하

여 제공 받음으로써 빠른 응답 시간을 보장하고 네트워크 트래픽도 줄임으로써 대역폭 사용량도 줄일 수 있다[7, 27, 28]. 클라이언트들의 요구를 기반으로 하는 웹 캐싱은 peak peiods 와 off-peak periods 의 구별 없이 대역폭 사용량을 감소시킨다. 따라서, 웹 캐싱을 사용하더라도 off-peak periods 동안에 사용되지 않는 네트워크 대역폭을 효율적으로 사용하여 peak periods 동안의 대역폭 사용량을 감소시킬 수는 없다[4, 5, 11].

본 논문에서는 네트워크 대역폭을 균형 있게 사용하기 위해 캐싱 서버 (caching server)를 기반으로 하는 웹 프리페칭 (Web prefethcing) 기법을 제안한다. 본 논문에서 제안하는 웹 프리페칭 기법은 peak periods 동안 요청될 것이라고 예상되는 웹 객체를 off-peak periods 동안의 남는 대역폭을 이용하여 미리 캐싱 서버에 저장시켜 놓는 방법이다. 여기서 웹 객체는 웹 상에서 존재하는 문서, 이미지 또는 가능한 모든 타입의 데이터를 가리킨다. 사실, 이 방법은 일반적인 웹 캐싱과 비교해 볼 때 평균 대역폭 사용량을 줄이기보다는 오히려 총 대역폭 사용량을 증가시킬 수 있다. 그러나 이 방법은 사용되지 않는 off-peak periods 동안의 대역폭을 이용하여 peak periods 동안의 대역폭 사용량을 줄이는데 효율적일 것이다.

본 논문의 2 장에서는 본 논문과 관련하여 여러 논문에서 제시하고 있는 웹에서의 프리페칭 기법에 대해서 살펴보고 이들 논문들을 기반으로 하여 웹 프리페칭 기법에 대하여 살펴보며 3 장에서는 본 논문에서 제안하는 웹 프리페칭 기법에 대한 소개와 기존의 프리페칭 방법의 문제점을 소개하고 해결방안을 제시한다. 4 장에서는 본 논문에서 제시하는 프리페칭 시스템의 설계에 대해 설명하고 5 장에서는 시뮬레이션 방법을, 6 장에서는 시뮬레이션의 결과를 보여준다. 끝으로 7 장에서는 본 논문의 결론과 앞으로의 연구 발전 방향을 제시한다.

2. 관련 연구

웹 프리페칭이란 사용자들이 요구할 가능성이 높은 웹 문서를 사용자가 요청하기 전에 미리 캐싱하는 기법이다. 지금까지의 연구를 살펴보면 주로 네트워크 지연시간을 줄이기 위해 웹에서 프리페칭을 사용하였다. 이 장에서는 본 논문과 관련하여 여러 논문에서 제시하고 있는 웹에서의 프리페칭 기법에 대해서 살펴보고 이들 논문들을 기반으로 하여 웹 프리페칭 기법을 분류하도록 하겠다.

2.1 프리페칭에 관련된 연구

지금까지 사용자들에게 보다 빠른 응답 시간을 제공하고 캐싱 서버의 성능을 높이기 위하여 웹 프리페칭에 대한 많은 연구가 이루어졌다. 지금까지 연구된 웹 프리페칭 기법은 웹 서버, 클라이언트, 그리고 프락시에서 수행되고 있다.

먼저 클라이언트 쪽에서 수행되는 프리페칭 기법에 대한 논문들을 살펴보면 Padmanabhan 과 Mogul 은 네트워크 지연시간과 네트워크 트래픽을 감소시키기 위해 사용자가 현재의 웹 객체를 브라우징하고 있는 동안 곧 요청되리라고 생각되는 웹 객체를 예측하고 프리페칭하는 기법을 소개하고 있다[2]. 이 논문에서는 웹 서버가 각각의 클라이언트의 접근 패턴을 분석하여 프리페칭할 웹 객체를 예측 (prediction)하고 클라이언트에서는 웹 서버가 예측한 문서를 프리페칭하는 구조를 제안하고 있다. 앞으로 요청될 웹 객체를 예측하기 위한 예측 알고리즘 (prediction algorithm)으로는 Griffioen 과 Appletion[13]에서 제시하고 있는 알고리즘을 응용하여 사용하고 있는데, 이 알고리즘은 사용자가 어떤 웹 문서를 보고 있을 경우 다음에 요청할 웹 객체는 현재 보고 있는

웹 문서에 hyperlink 되어 있는 웹 객체일 확률이 매우 높다는 특성에서 착안되어 만들어졌다. 이것은 웹 서버에 저장되어 있는 웹 객체들 간의 관계를 dependency graph 로 나타낸다. 이 graph 는 웹 서버에 저장되어 있는 모든 웹 객체를 노드로 가지고 있고 만일 A 라는 웹 객체가 접근된 후에 B 라는 웹 객체가 요청되었다면 노드 A 에서 B 로 arc (edge)를 그린다. arc 의 값은 A 가 요청된 다음에 B 가 요청될 확률이다. 웹 서버에서는 웹 객체들간의 dependecy graph 를 생성하여 클라이언트에게 제공하고 클라이언트는 이러한 정보를 이용하여 다음에 요청될 확률이 높은 웹 객체를 프리페칭한다. 이 논문은 웹 서버 추적 (Web server traces)과 trace-driven 시뮬레이션을 이용하여 프리페칭의 유효성을 증명하였다. 프리페칭 성능 측면에서 보면 클라이언트의 네트워크 지연시간을 최고 45%까지 줄일 수 있으나 네트워크 트래픽은 2 배로 증가되는 단점이 있다.

Wang 과 Crowcroft 는 HotList Manager 라는 시스템을 구현하여 웹에서의 delay 를 줄이는 목적으로 프리페칭을 이용하고 있다[1]. 이 논문에서는 deterministic client-initiated prefetching 기법을 사용하고 있는데, 이것은 클라이언트에서 프리페칭할 웹 문서를 정적으로 설정하는 방법이다. Deterministic prefetching 은 적은 대역폭 오버헤드만을 발생시키는 가장 보수적인 프리페칭 기법이다.

두 번째로 웹 서버에서 수행되는 프리페칭 기법에 대한 논문들을 살펴보면 Bestavros 는 웹을 위한 Server-initiated prefetching 을 소개하고 있다[14]. 모든 웹 문서 쌍인 D_i, D_j 에 대하여 Bestavros 는 웹 문서 D_i 가 요청된 후에 time interval T_w 안에 D_j 가 요청될 확률 p[i,j]을 계산하였다. 이 확률을 바탕으로 웹 서버는 클라이언트에게 어떠한 웹 문서를 요청할 것인지 "advice" 한다. Bestavros 는 trace-driven 시뮬레이션을 통해 10%의 네트워크 트래픽을 더 증가 시키면서 23%의 cache miss rate 을 감소시키는 결과를 보여주었다. 이 논문에서는 웹 서버에 의해 관찰된

웹 문서의 참조 패턴이 프리페칭을 동작하기 위한 효율적인 정보로써 사용될 수 있다는 것을 보여주었다.

Kroeger 는 웹 서버와 프락시 사이에서의 프리페칭 성능의 한계를 조사하였고, 프락시에서 캐싱과 프리페칭을 통합하여 사용할 때 클라이언트의 지연시간을 60%정도 감소시킬 수 있다는 것을 보여주고 있다[15].

세 번째로 프락시 서버에서 수행되는 프리페칭 기법에 대한 논문들을 살펴보면 Chinen 과 Yamaguchi 는 프락시가 웹 서버로부터 프리페칭하는 Wcol 이라는 시스템을 제시하고 있다[3, 17]. 이 시스템은 HTML 문서를 파싱 (parsing)하고 그 문서에 링크된 다른 웹 문서와 이미지들을 프리페칭한다. 그러나 프락시는 클라이언트에게 프리-푸싱 (pre-pushing)하지는 않는다. Wcol 을 이용해 프리페칭하면 응답시간이 프리페칭을 하지 않았을 경우보다 2.46 배 빨라지고 캐시 적중률도 1.67 배가 증가하였다. 그러나 트래픽 발생량도 2.81 배로 증가하여 과다한 트래픽을 발생시키는 단점이 있다.

Jacobson 과 Cao 는 모뎀을 사용하고 있는 클라이언트의 네트워크 지연시간을 줄이기 위해 프락시가 클라이언트에게 프리-푸싱하는 기법을 제시하였다[18]. 프리-푸싱은 클라이언트가 다음에 요청할 것이라고 여겨지는 웹 객체를 예측하여 프락시가 클라이언트에게 푸시 (push)하는 방법이다. 이 논문은 모뎀을 사용하는 사용자들의 웹 접근 추적 (Web access traces)을 사용하여 예측 알고리즘을 개발하고 오버헤드 (overhead)를 알아 보기위해 정확도 (accuracy)를 측정하였고 예측 알고리즘을 이용한 프리-푸싱 기법 (pre-push scheme)의 지연시간 감소율을 측정하였다. 결과로써 프락시 서버를 기반으로 하는 웹 프리-푸싱을 사용하여 지연시간을 10%정도 감소시킬 수 있다. 이때 네트워크 트래픽은 18%가 증가되었고, 요청 (request)의 12%를 예측하였다.

그 외에도 캐시 적중률을 증가시키기 위한 목적으로 Makatos 와

Chronak 는 프리페칭을 위한 Top-10 Approach 를 제안하고 있다[16, 24]. 이들은 client-proxy-server framework 을 소개하고 있다. 각각의 웹 서버는 클라이언트들의 웹 접근 패턴을 분석하여 가장 많이 접근되는 웹 문서 Top-10 을 선정한다. 이러한 분석을 바탕으로 클라이언트는 웹 서버에게 요청하고 웹 서버는 가장 많이 접근되는 웹 문서들을 클라이언트에게 전달한다. 만일 클라이언트와 웹 서버 사이에 프락시 서버가 존재한다면 클라이언트는 프락시 서버에게 프리페칭할 웹 문서를 요청하고 프락시 서버는 다시 해당 웹 서버에게 요청한다. 이 논문에서는 다양한 웹 서버 추적 (Web server trace)을 사용하여 성능을 평가하였고, 단지 네트워크 트래픽을 10% 증가 시키면서 클라이언트들의 요청을 40% 이상 미리 예측할 수 있는 기법을 개발하였다. 그러나 이 논문에서 제안한 프리페칭 방법은 웹 서버들로부터의 협력을 요구하며 웹 서버, 프락시 서버, 클라이언트 모두에게 해당 소프트웨어를 추가하거나 변형시켜야 하기 때문에 모든 웹 서버, 프락시 서버와 클라이언트에게 적용하기에는 어려움이 있다는 단점이 있다.

2.2 웹 프리페칭 기법의 분류

프리페칭에 관련된 논문들을 기반으로 하여 웹 프리페칭의 기법은 크게 세가지 측면으로 분류될 수 있다. 첫 번째는 프리페칭을 수행하는 위치에 따른 분류이고, 두 번째로는 프리페칭될 웹 문서의 선정 방법에 따라 분류할 수 있으며, 세 번째로는 프리페칭하는 목적에 따라 분류할 수 있다.

2.2.1 프리페칭이 수행될 위치에 따른 분류

프리페칭은 웹 서버, 클라이언트, 캐싱 서버에 의해서 수행될 수 있다.

Server-initiated prefetching : 웹 서버에서 프리페칭이 수행되는 방법을 말하며, 이것은 클라이언트나 프락시로부터 웹 문서가 요청될 때 웹 서버는 그 웹 문서를 파싱하여 hyperlink 와 같이 다음에 따라올 가장 적당한 요청를 예측해서 해당하는 웹 페이지를 클라이언트나 프락시에게 푸싱 (pushing)한다[2, 14, 15]. 다음 웹 객체에 대한 요청을 하였을 때 그 웹 객체는 이미 전송되어 있거나 전송중일 것이다. 이 방법은 웹 서버에 접근하는 클라이언트들의 접근 패턴을 분석하여 프리페칭에 사용할 수 있다는 장점이 있으나 웹 객체를 푸싱하는 것은 클라이언트나 프락시로부터 허용되어 있어야 하고, 웹 서버 소프트웨어의 변경을 요구하기 때문에 모든 웹 서버로 확장하는데 어려움이 있다는 단점이 있다.

Client-initiated prefetching : 클라이언트에서 프리페칭을 수행하는 방법이다. 클라이언트는 특정한 웹 문서들을 위한 과거의 접근 패턴을 모니터할 수 있는 사용자 에이전트 (agent)를 바탕으로 프리페칭할 웹 문서를 초기화 할 수 있고 프리페칭할 수 있다[1, 2, 13]. 이 방법은 웹 서버나 프락시 서버에게 투명한 방법으로 개개의 클라이언트에 의해서 이루어 질 수 있다. Client-initiated prefetching 은 웹 서버와 클라이언트 간의 또는 프락시와 클라이언트 간의 상호 협력을 통해 수행된다. 이 기법은 단 하나의 클라이언트의 웹 접근 패턴 (Web access pattern)을 이용하기 때문에 인터넷의 전체적인 사용자들의 접근 패턴을 공유할 수 없다는 단점이 있다.

Proxy-initiated prefetching : 프락시 서버에서 사용자들의 웹 문서 접근 패턴을 바탕으로 프리페칭 하는 방법이다. 프락시 서버에서 클라이언트로부터 웹 객체에 대한 요청을 받은 후 바로 다음에 요청할 웹 객체를 예측하여 프리페칭을 수행하거나, 일괄적으로 프리페칭할 웹 문서들을 선정하여 일정한 시간대에 수행한다[3, 17, 18]. 프락시 서버는 연결된 많은 사용자들의 웹 접근 패턴을 관찰하여 가까운 미래에 요청될 웹 객체를 예측할 수 있다는 것과 클라이언트나 웹 서버를 변경시킬 필요가

없다는 장점이 있으나 근래에 요청된 웹 객체를 저장하고 각각의 클라이언트에게 제공하는 캐싱 서버로서의 기능이 있어야 한다는 것을 전제로 한다.

2.2.2 프리페칭될 웹 문서의 선정 방법에 따른 분류

프리페칭은 프리페칭될 웹 객체를 선정하는 방법에 따라 통계적 방법 (statistical prefetching)[5, 8, 9]과 결정적 방법 (deterministic prefetching)으로 분류할 수 있다[1].

Statistical prefetching : 통계적 방법은 클라이언트들에 대한 최근의 로그 (access log)를 바탕으로 일정한 기간동안의 웹 객체 접근의 상호 의존성을 계산하여 프리페칭될 웹 객체를 선정하는 방법이다. 이 방법은 통계에 의한 것이기 때문에 결정적 방법보다는 많은 네트워크 장비가 낭비될 수 있다.

Deterministic prefetching : 결정적 방법은 프리페칭될 웹 객체들이 사용자들에 의해 정적으로 구성될 수 있다. 예를 들어, 어떤 사용자가 뉴스 페이지를 접근할 때 미리 읽어들일 웹 객체들을 지정하면 그 뉴스 페이지가 접근될 때 지정된 웹 객체들이 프리페칭된다. 이 방법은 필요한 웹 객체들만을 지정하여 요청하기 때문에 낭비되는 네트워크 자원이 적다는 장점이 있으나 관리자나 사용자가 프리페칭될 웹 객체를 하나하나 지정해야 하기 때문에 전체 웹 객체를 대상으로 하기에는 한계가 있다는 단점이 있다.

2.2.3 목적에 따른 분류

프리페칭은 목적에 따라 응답 시간 (response time)을 줄이기 위한 프리

페칭과 균형 잡힌 대역폭 사용을 위한 프리페칭으로 구분해 볼 수 있다.

Prefetching for Response Time Reduction : 응답 시간을 줄이기 위한 프리페칭은 사용자가 특정한 웹 문서를 보고 있는 동안 다음에 읽혀질 가능성이 큰 순서대로 그 웹 문서와 연결된 문서들을 미리 받아서, 사용자가 그 문서를 요구했을 때 신속하게 제공하는 방법이다. 이 기법에서 사용되는 프리페칭 알고리즘은 예측 알고리즘[2, 5, 13, 16, 18]을 주고 사용하며 참조되지 않는 많은 문서들까지 프리페칭 하게 되므로 네트웍 자원의 낭비를 가져오고 네트워크 트래픽이 많은 peak periods 때에는 오히려 병목현상을 가중시키는 단점이 있다.

Prefetching for Balanced Bandwidth Usage : 균형 잡힌 대역폭 사용을 위한 프리페칭 기법은 peak periods 와 off-peak periods 간의 불균형적인 대역폭 사용을 프리페칭을 통해 해결하고자 하는 것이다. Batch prefetching 기법이 균형잡힌 balanced bandwidth usage 를 위해 많이 사용되는데 이 방법은 하루의 시간대 중에서 가장 네트웍 사용량이 적은 시간대에 사용자들의 요청이 많을 것으로 예상되는 웹 문서를 일괄적으로 미리 프리페칭하는 기법이다.

3. 제안 프리페칭 기법

위의 프리페칭 방법들 중에서 Server-initiated Prefetching 을 위해서 클라이언트들은 웹 문서가 정확하게 프리로드될 수 있도록 허가되어 있어야 하고 웹 서버 소프트웨어의 변경을 요구하기 때문에 모든 웹 서버로 확장하는데 어렵다는 단점이 있다. Client-initiated Prefetching 은 단 하나의 클라이언트의 웹 문서 접근 패턴을 이용하기 때문에 인터텟의 전체적인 사용자들의 접근 패턴을 공유할 수 없다는 단점이 있다.

웹 문서 선정 방법에 따른 분류에서 결정적 방법은 관리자나 사용자가 프리페칭될 웹 문서를 하나 하나 지정해야 하기 때문에 전체 웹 문서를 대상으로 하기에는 한계가 있다.

본 논문은 균형적으로 네트워크 대역폭을 사용하고 peak bandwidth usage 를 줄이기 위한 연구이다. 따라서, 본 논문에서는 위의 프리페칭 기법들 중에서 프락시 서버에서 통계적인 방법으로 프리페칭할 웹 문서들을 선택하여 일괄적으로 프리페칭하도록 한다. 그러나 본 논문에서 선택한 프리페칭 기법 각각에도 고려해야 될 사항들이 존재한다.

다음은 프리페칭시 고려사항들과 각각에 대한 해결방법을 소개한다. 또한 프리페칭될 웹 객체들을 선정할 기준이 되는 프리페칭에 필요한 인자 (parameter)에 대하여 알아보고 real-world traffic pattern 을 분석하기 위하여 포항공대 기숙사 지역 16 개의 서브넷 (subnet)에서 발생되는 웹 트래픽을 분석한다. 마지막으로 본 논문에서 제안된 프리페칭 알고리즘의 유효성을 증명하기 위한 프리페칭 성능 평가 척도 (performance metrics)에 대해서 알아보도록 하겠다.

3.1 고려사항

본 논문에서 선택한 방법에는 세 가지의 문제점이 존재한다.

- ?? 통계적 프리페칭을 사용할 경우 사용자들의 웹 문서 접근 패턴을 최대한 반영해야 되는데, 사용자들의 웹 문서 접근 패턴이 유동적으로 바뀐다는 점이다.
- ?? 프리페칭된 웹 문서를 사용자가 요청하지 않을 경우 자원의 낭비를 초래한다. 따라서 프리페칭된 웹 문서에 대해 높은 캐시 적중률이 필요하다.
- ?? 기존의 일괄 프리페칭 방법은 대용량의 디스크를 요구하는데 요구되는 디스크의 크기를 융통성 있게 조절할 수 있는 일괄 프리페칭 알고리즘이 필요하다.

3.2 해결방법

각 문제점에 대한 해결방법은 다음과 같다.

- ?? 사용자들의 웹 문서 접근 패턴을 프리페칭에 반영하기 위해서 최근 일정기간동안 캐싱 서버에게 요청되고 저장된 웹 문서를 대상으로 프리페칭한다 (예: 하루, 1주일). 사용자들의 접근 패턴은 인트라넷마다 성격이 다를 수 있으므로 패턴을 주기적인 관찰에 의해 기간을 정할 수 있도록 해야 한다.
- ?? 프리페칭된 웹 문서에 대한 적중률을 높이기 위해서 즉, 프리페칭의 정확도 (accuracy)를 높이기 위해 prefetchable objects 선택에 제한이 있어야 한다. 캐싱 서버는 클라이언트가 요청한 웹 객체가

자신의 캐시에 저장되어 있어도 그 웹 객체의 유효기간 (expiration time)이 지났다고 판명되면 캐시 미스 (cache miss)로 보고 해당 웹 서버에게 새로이 요청을 보낸다. 따라서 먼저 캐시에 저장되어 있고 유효기간이 지난 웹 객체를 대상으로 한다. 그런 후에 그 웹 객체를 접근한 사용자들의 요청 횟수를 나타내는 참조 횟수가 일정 횟수 이상인 웹 문서들만으로 프리페칭 대상에 제한을 둔다.

?? 위의 두 문제가 해결되면 기존의 방법보다 적은 디스크 용량을 요구하게 되며 파라미터들의 조절에 의해 각각의 인트라넷 환경에 맞는 프리페칭 방법을 적용할 수 있다.

3.3 프리페칭 파라미터 (Prefetching Parameters)

앞에서 설명된 해결방법을 반영하기 위한 프리페칭 파라미터를 설정하는데, 이것은 프리페칭할 웹 문서를 결정하는 기준이 된다. 프리페칭 파라미터는 네가지로써, 프리페칭 대상 선정 기간, 참조 횟수, 프리페칭될 웹 문서의 전체 바이트수와 프리페칭을 수행할 시간이다.

?? 첫째로 프리페칭 대상 선정 기간은 사용자들의 웹 접근 패턴을 충분히 반영하기 위한 것이다. 따라서 최근에 접근된 웹 객체를 대상으로 선택하는 것이 좋다. 프리페칭 대상 선정 기간은 현재로부터 과거 하루, 몇일 등으로 각 인트라넷의 특성에 맞게 결정한다.

?? 두 번째 프리페칭 파라미터는 참조 횟수이다. Traditional prefetching scheme 의 단점인 과다한 트래픽의 발생을 줄이고 프리페칭될 웹 문서에 대한 적중률을 높이기 위해서는 참조 횟수가 n 회 이상인

웹 객체를 대상으로 제한한다. 이것은 다음에 요청될 가능성이 높은 객체들만을 프리페칭함으로써 적은 수의 웹 객체를 프리페칭하여도 높은 프리페칭 성능을 나타내기 위함이다.

?? 세 번째 프리페칭 파라미터로는 프리페칭될 웹 문서의 전체 크기이다. 이것은 참조 횟수와 연관 지어서 결정할 수 있다. 예를 들어 프리페칭할 웹 문서의 전체 용량을 100M 로 제한한다면 참조 횟수가 높은 웹 객체들부터 선택되도록 한다.

?? 네 번째 프리페칭 파라미터는 프리페칭이 수행될 시간을 결정하는 것이다. 본 논문은 off-peak periods 의 남는 대역폭을 이용하여 peak periods 와 off-peak periods 간의 균형있는 대역폭 사용을 위한 프리페칭이다. 따라서 프리페칭이 적용될 시간은 웹 트래픽이 적은 off-peak periods 동안으로 설정하며 off-peak periods 를 정하는 기준에 대해서는 3.5 에서 자세하게 설명하도록 한다.

3.4 포항공대 기숙사의 웹 트래픽 추적 (Web Traffic Trace)

웹 접근 패턴을 분석하고 프리페칭 파라미터들의 값을 결정하기 위하여 real-world traffic pattern 을 조사하였는데 포항공대 기숙사 지역 16 개의 서브넷 (subnet)에서 발생되는 웹 트래픽을 대상으로 하였다. 포항공대의 웹 트래픽 추적 기간은 1999 년 10 월 15 일 (금)부터 10 월 28 (목)일까지 2 주 동안이다. 2 주 동안 웹 객체를 요청한 클라이언트들의 수는 447 대였다. 캐시 객체 적중률 (cache object hit rate)은 54.96%였으며 캐시 바이트 적중률 (cache byte hit rate)은 31.42%를 나타내고 이 기간동안 접근된 요청 (request)의 수는 4,077,308 개였고 total bytes 는 50.2G 였다. 또한 하루에 캐싱 서버로 들어오는 평균 요청의 개수는 291236 개 정도였고 하루 평균 bytes 는 3.6G 였다. 표 1 은 캐싱 서버의 성능을 정

리한 표이다.

표 1. Cache Summary Statistics

Number of Client Making Requests	447 대
Cache Object Hit Rate	54.96 %
Cache Byte Hit Rate	31.42 %
Total Number of Objects Requested	4,077,308 회
Total Bytes to Clients	50.2 Gbyte

표 2 는 캐시에 저장된 웹 객체에 대한 참조 횟수 (reference count)를 나타낸다. 참조 횟수는 클라이언트들이 웹 객체별로 요청한 횟수를 나타낸다. 캐시에 저장된 웹 객체들 중에서 단지 한번만 참조된 웹 객체들은 전체 웹 객체 중에서 70.75%를 나타내었고 2 회 이상 참조된 웹 객체들은 단지 29.25%에 불과하였다. 하루동안 클라이언트들에 의해 요청되는 웹 객체의 종류는 121,883 개 정도였다. 참조 횟수와 웹 객체수와의 관계를 그림으로 나타내면 그림 3 과 같다.

표 2. 참조 횟수에 따른 웹 객체 수

참조 횟수	하루 평균 웹 객체수	백분율 (%)
1 회	86,227	70.75
2 회	16,690	13.69
3 회	6,462	5.3
4 회	3,451	2.83
5 회 이상	9,053	7.43
Total	121,883	100

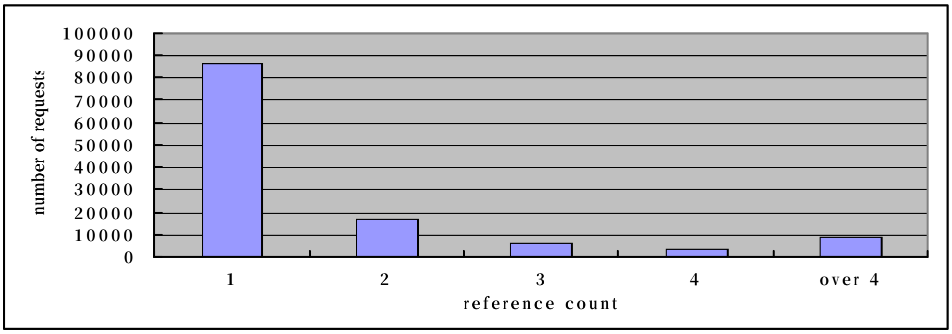

그림 3. 참조 횟수별 웹 객체수

표 3 은 참조 횟수에 따른 웹 객체의 평균 총량을 나타낸 것이다. 표 3 을 보면 웹 객체의 total size 는 참조 횟수가 1 회인 것이 68.12%를 차지하였고, 2 회 이상인 것이 전체의 31.88%를 차지하였다. 또한 하루동안 요청된 웹 객체들의 total bytes 는 2172.27 Mbyte 였다. 참조 횟수와 웹 객체의 총량과의 관계를 그림으로 나타내면 그림 4 와 같다. 웹 객체의 평균 총량도 웹 객체수와 비슷한 패턴을 보이고 있다는 것을 알 수 있다.

표 3. 참조 횟수에 따른 웹 객체의 평균 총량

참조 횟수	웹 객체의 하루 평균 총량 (Mbyte)	백분율 (%)
1 회	1479.75	68.12
2 회	310.79	14.31
3 회	118.20	5.44
4 회	74.01	3.41
5 회 이상	189.52	8.72
Total	2172.27	100

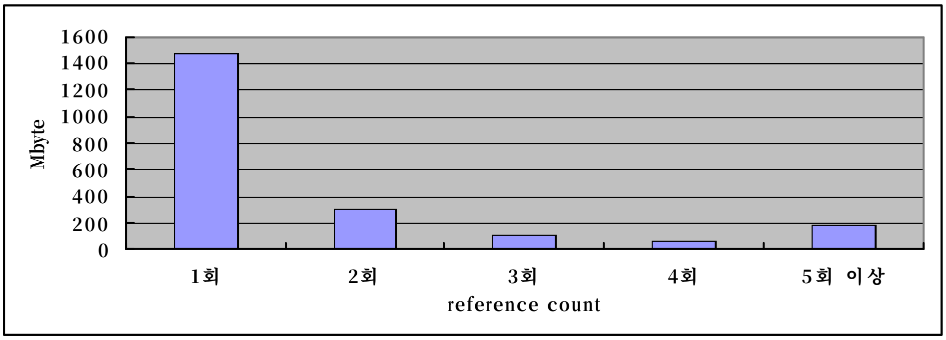

그림 4. 참조 횟수별 웹 객체의 평균 총량

그림 5 는 참조 횟수별 요청 수를 요일별로 나타낸 것이고 그림 6 은 참조 횟수별 웹 객체의 total bytes 를 요일별로 표현한 것이다.

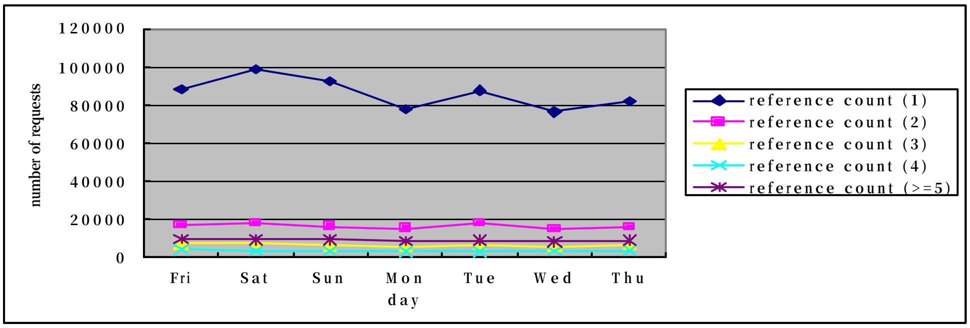

그림 5. 웹 객체 참조 횟수에 따른 요일별 요청수

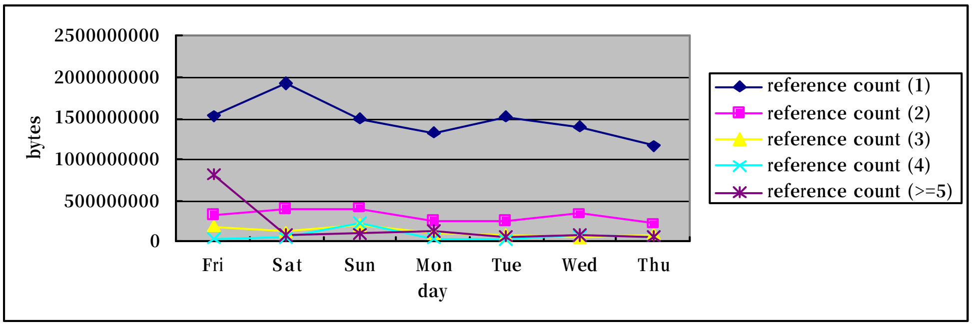

그림 6. 웹 객체 참조 횟수에 따른 요일별 요청의 총량

3.5 프리페칭 수행 시간 (Prefetching Time)

본 논문은 off-peak periods 의 남는 대역폭을 이용하여 peak periods 와 off-peak periods 간의 균형있는 대역폭 사용을 위한 프리페칭이다. 따라서 프리페칭이 적용될 시간은 웹 트래픽이 적은 off-peak periods 동안이다. 앞의 서론에서는 off-peak periods 를 전체 트래픽량의 평균을 기준으로 평균보다 미만일 경우를 off-peak periods 로 정하였었다. 그러나 프리페칭이 수행되면 off-peak periods 동안에 많은 웹 트래픽이 발생되므로 본 논문에서는 off-peak periods 를 전체 웹 트래픽량의 평균을 기준으로 하여 평균 80% 미만 (125299861 byte)인 시간대를 off-peak peirods 로 결정한다. 따라서 off-peak periods 는 새벽 04:00 부터 13:00 까지며, peak-periods 는 13:00 부터 04:00 까지다. off-peak periods 를 결정하는 기준은 각각의 인트라넷 구성 조직의 특성에 따라 변경이 가능하므로 본 논문에서는 자세한 언급은 하지 않겠다. 결과적으로 본 논문에서는 프리페칭이 수행될 시간은 off-peak periods (04:00 – 13:00)사이이며 prefetchable object 들을 일괄 프리페칭하도록 한다. 그림 7 은 시간대별 평균 웹 트래픽을 그림으로 나타낸 것이다. 중간의 점선은 웹 트래픽량의 평균 80%를 나타낸 것이다.

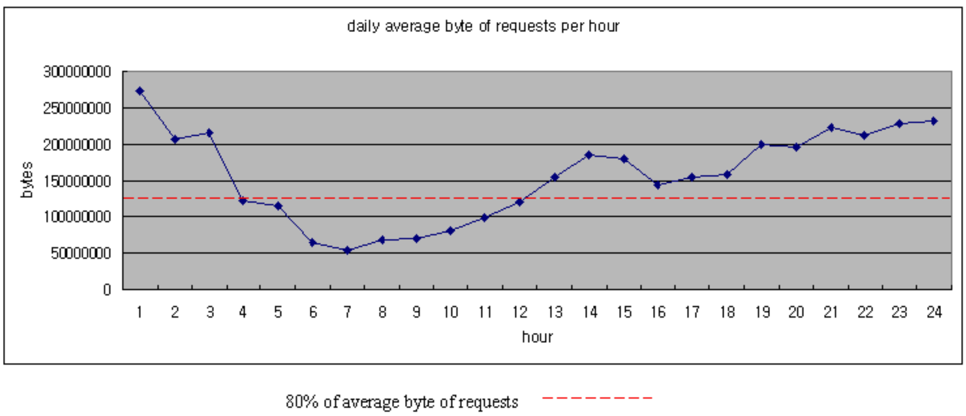

그림 7. 시간대별 평균 웹 트래픽량

3.6 성능 평가 척도 (Performance Metrics)

프리페칭 알고리즘의 유효성을 증명하기 위해서는 성능을 측정할 수 있는 기준이 필요한데, 프리페칭의 성능 측정 기준은 다음과 같이 네 가지로 볼 수 있다.

?? ***Request Saving*** : 프리페칭된 웹 객체들 중에서 다음날 실제로 사용자에 의해 요청되어 처음으로 hit 이 발생된 요청 횟수를 사용자의 전체 요청 수의 비율로 나타낸다. Request saving 이 높을수록 캐시 객체 적중률은 높아진다.

?? ***Bandwidth Saving*** : 프리페칭에 의해 절약된 대역폭을 사용자의 전체 웹 트래픽에 대한 대역폭 사용량의 비율로 나타낸다. Bandwidth saving 이 높아질수록 캐시 바이트 적중률은 높이지고 그만큼 peak periods 의 bandwidth usage 는 줄어들게 된다. 또한 프리페칭에 의해서 절약되는 대역폭은 실제로는 peak periods 동안에 사용될 대역폭을 줄여주기 때문에 프리페칭을 통해서 peak periods 동안에 절약되는 대역폭 감소율을 측정한다.

?? ***Accuracy*** : accuracy 는 본 논문에서 제안한 프리페칭 알고리즘의 정확성을 측정하기 위한 것이며 프리페칭 객체 적중률 (prefetched object hit rate)과 프리페칭 바이트 적중률 (prefetched byte hit rate)로 나타낸다. 프리페칭 객체 적중률은 프리페칭된 웹 객체들 중에서 실제로 다음날 요청된 요청 횟수의 비율을 나타내며 프리페칭 바이트 적중률은 프리페칭된 웹 객체의 total bytes 중에서 실제로 다음날 요청된 웹 객체들의 total bytes 의 비율을 나타낸 것이다. Accuracy 가 높아 질수록 프리페칭에 의해 낭비되는 대역폭은 적어진다.

?? ***Wasted Bandwidth*** : 프리페칭 되었지만 클라이언트에 의해서 요청

되지 않은 웹 객체의 전체 바이트수로 나타낸다. 본 논문은 off-peak periods 동안의 남는 대역폭을 사용하여 프리페칭하기 때문에 off-peak periods 동안에 증가되는 대역폭 증가율도 함께 측정한다.

4. 프리페칭 시스템 설계

앞에서 설명된 프리페칭 방법을 기반으로 캐싱 서버에 독립적으로 동작되는 프리페칭 시스템을 설계해 본다. 프리페칭 시스템의 구조는 그림 8과 같이 나타낼 수 있다.

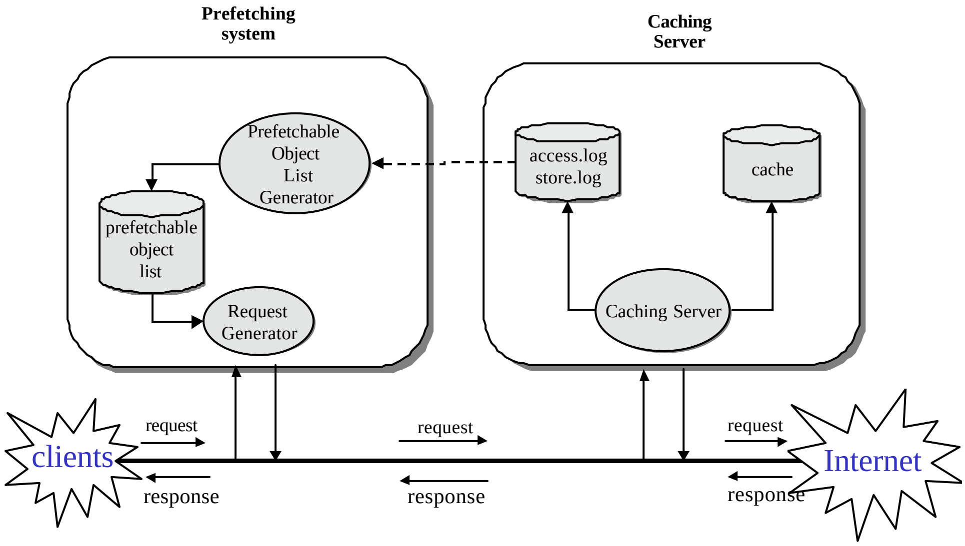

그림 8. 프리페칭 시스템 구조

프리페칭 시스템 설계에 들어가기 앞서 설계 시 필요한 몇 가지 가정들이 있다. 첫번째로 본 논문에서 제안하는 프리페칭 시스템은 캐싱 서버와의 연동이 필요한데, 캐싱 서버로는 Squid[7]를 사용하였다. Squid 를 캐싱 서버로 사용한 이유는 Squid 는 freeware 이기 때문에 학교, 회사등에서 많이 사용되고 있으며 상용 캐싱 서버 제품으로 Cobalt Network 의 CacheRaQ[29]와 Packetstorm Technologies 의 Webspeed[30]도 내부적으로 Squid 를 사용하고 있다. 따라서 Squid 를 기반으로 프리페칭 시스템을

설계하면 Squid 를 사용하고 있는 곳은 어디에서나 본 논문에서 제안하는 프리페칭 시스템을 사용할 수 있다. 두 번째로 캐싱 서버는 transparent 캐싱 서버로서 동작한다. Transparent 캐싱 서버는 클라이언트의 브라우저 (browser)와 웹 서버사이의 데이터 경로 (data path)상에 위치하여 웹 트래픽을 받아서 서비스를 제공하는 캐싱 서버의 한 종류이다[31]. 이것은 프락시 캐싱의 커다란 단점중의 하나인 브라우저에서 캐싱 서버를 configuration 하는 것을 제거한다는 장점이 있다. 클라이언트에서 default gateway 로 캐싱 서버를 지정함으로써 브라우저는 캐싱 서버를 지정할 필요가 없다.

전반적인 프리페칭 시스템의 모듈은 크게 Prefetchable Object List Generator 와 Request Generator 두 가지로 나눌 수 있다. 클라이언트들이 캐싱 서버 (Squid)에게 요청을 보내면 Squid 는 먼저 자신의 캐시에 요청받은 웹 객체가 존재하는지 확인하고, 있으면 그 웹 객체를 클라이언트에게 제공하고 존재하지 않거나 유효하지 않을 경우에는 해당 웹 서버에게 요청을 보낸다. 이때 Squid 에 요청되는 모든 request 에 대한 정보가 “access log”로, 캐시에 저장되는 웹 객체에 대한 정보는 “store log”로 기록된다. 이 시스템은 squid 에서 작성된 이러한 로그 (access.log, store.log)들을 이용하여 Prefetchable Object List Generator 에서 prefetchable object list 를 생성한다. Request Generator 는 하나의 클라이언트로써 동작하여 off-peak periods 동안에 prefetchable object list 에 있는 요청 (request)들을 웹 서버에게 보내어 Squid 의 캐시에 저장되어 있는 웹 객체들을 갱신시킨다. 이 프리페칭 시스템은 Squid 와는 독립적으로 add-on 으로서 동작한다.

4.1 Prefetchable Object List Generator

Prefetchable Object List Generator 는 프리페칭될 웹 객체를 선택하기 위

한 모듈이다. 이것은 캐시에 저장된 각각의 웹 객체에 대한 참조 횟수와 웹 객체의 크기를 알기 위해 Squid 에 의해서 생산되는 “access log”와 “store log”를 이용한다. “access log”는 Squid 에게 요청되는 requests 에 대한 정보가 나타나 있고 “store log”는 캐시에 저장된 웹 객체에 대한 정보가 표현되어 있다. 먼저 Prefetch Object List Generator 는 Squid 의 재생 알고리즘 (refresh algorithm)을 이용하여 유효기간이 지난 웹 객체 중에서 앞장에서 설명된 조건에 따라 프리페칭이 가능한 prefetchable object 를 선택한다. HTTP response message 의 헤더 부분에 유효기간이 나타나 있지 않는 웹 객체는 Squid 에서 기본적으로 설정해주는 값 (3 days)으로 정한다[7]. 수집한 정보는 그림 9 의 형식에 맞게 분석하여 prefetchable object list 를 생성한다.

Referenced Count	URL	Byte

그림 9. Prefetchable Object List 의 형식

Prefetchable Object List Generator 는 웹 객체들의 유효기간을 확인하기 위하여 Squid 의 재생 알고리즘을 응용하여 사용하는데, 재생 알고리즘에 필요한 매개 변수들은 “store log”에 나타나 있다. “store log”는 해당 웹 객체가 캐시에 저장될 때 HTTP response header 에 들어있는 각 필드들 중에서 필요한 항목만 추출하여 저장한다.

다음은 재생 알고리즘에서 사용하는 HTTP response header 와 HTTP request header 의 형식과 필드에 대하여 알아보고 Squid 에서 제공되는 재생 알고리즘과 각각의 웹 객체들의 유효성을 결정하는 방법에 대해서 설명한다.

4.1.1 HTTP/1.0 Response Header

HTTP 프로토콜[22, 23, 33]은 요구/응답 (Request/Response) 방식을 이용하여 동작하고 있다. 즉, 원하는 프로토콜 기능(예: GET, HEAD, POST)에 대해 서비스 요구를 하면 데이터 송수신을 위한 TCP 연결이 만들어지고, 서버가 응답을 보내어 데이터 전송을 끝내면 자동적으로 연결이 끊어지게 되는 것이다. 즉, 클라이언트는 서버와의 사이에 TCP 를 기반으로 한 HTTP 연결을 만들고 method, URI, protocol version, 클라이언트 정보, 사용자 데이터 등과 같이 규정되어 있는 요구 형식에 따라 서버에게 요구 메시지를 보낸다. 서버는 프로토콜 버전, 성공 또는 오류코드 번호, 서버 정보, 데이터 정보, 사용자 데이터 등을 포함하는 응답 메시지를 보낸다.

캐싱 서버에서 사용되는 HTTP response header 의 필드들을 설명하기 전에 먼저 HTTP/1.0 response header 의 형식에 대해서 살펴보면 그림 10 과 같다.

```
Full-Response     = Status-Line
                    *(General-Header | Response-Header | Entity-Header)
                    CRLF [Entity-Body]
Status-Line       = HTTP-Version | Status-Code | Reason-Phrase CRLF
General-Header    = Date | Progma
Response-Header = Location | Server | WWW-Authenticate
Entity-Header     = Allow | Content-Encoding | Content-Length | Content-type |
                      Expires | Last-Modified | extension-header
extension-header  = HTTP-header
```

그림 10. HTTP/1.0 Response Header

Full-Response 메시지에 들어가는 첫 번째 줄의 내용이 Status-Line 이며 다음과 같은 구성 형식으로 되어 있다. HTTP 버전이 제일 먼저 나오고 이어서 숫자로 된 상태 코드 (Status-Code)가 표시되고 이어서 관련된 추가적인 내용 설명 (Reason-Phrase)이 덧붙여진다. 메시지 정보의 끝을 나타내는 CRLF 는 마지막 이외의 장소에는 허용되지 않는다. 다만 사용자 데이터와 같은 Entity 부분은 CRLF 에 뒤이어 나타날 수 있다. 따라서 실제 사용자 데이터를 보내기 위해서는 메시지 정보 다음에 공백의 빈 줄을 반드시 집어넣어야 인식할 수 있다. Status-Code 는 서버가 요구 메시지를 수신하여 처리한 결과를 알려주는 세 자리의 정수로 표현되는 처리 결과 번호이다. 첫 번째 자리 숫자는 응답의 분류 기호이며 나머지 두 자리 숫자는 일련 번호이다. 현재 첫 번째 자리 숫자에 의해 다섯 가지로 분류되어 쓰이고 있다.

?? 1xx : Informational - 향후의 사용을 위해 예약. 아직 사용되지 않음.

?? 2xx : Success - 성공적으로 수신되고 해독되고 처리된 경우.

?? 3xx : Redirection - 완전한 처리를 위해 추가적인 동작이 필요로 하는 경우.

?? 4xx : Client Error - 요구 메시지에 글자상의 문제가 있는 경우거나 처리할 수 없을 때.

?? 5xx : Server Error - 서버가 요구 메시지를 처리하는 가운데 문제가 발생한 경우.

응답 메시지 (response message)의 항목들에 보면, General-Header, Request-Header, Response-Header, Entity-Header 등과 같이 네 가지 헤더 형식이 있다. 여기에서 공통적으로 사용하는 표현 형식은 RFC 822 의 3.1 절에서 정의하고 있는 것이다[32].

General Header 는 Full-Request 메시지와 Full-Response 메시지에 공통적으로 포함되어 있는 헤더 형식이다. 이것은 전송되고 있는 메시지에 관한 사항을 알리는 것이지, 전송되고 있는 사용자 데이터에 관한 사항을 알리는 것이 아니다. General Header 는 Date 와 Progma 로 구성되어 있다. Date 는 메시지가 만들어지는 날짜와 시간을 나타낼 때 쓰이며 Progma 는 요구/응답의 연쇄 동작에 따라 어느 수신측에 적용되는 변수로서 구현에 관련된 것들을 포함하는 데 쓰인다.

Response Header 필드는 서버가 응답 메시지를 보낼 때 Status-Line 에다 실을 수 없는 추가적인 정보를 전달하고자 할 때 쓰인다. 서버는 이것을 이용하여 서버 자신에 대한 정보나 Request-URI 에서 명시한 자원에 대한 접근 및 이용 방법에 대한 정보를 전달할 수 있다.

Entity-Header 필드는 Entity-Body 에 대한 인코딩 방식, 최종 수정 일자, 유효 기간, 문서 길이 등과 같은 외형적 정보(metainformation)를 나타낼 때 쓰이거나, 전달할 entity body 데이터가 없을 때는 요구 메시지에서 지정한 해당 자원에 대한 정보를 알려줄 때 쓰인다.

Squid 의 재생 알고리즘에서 사용되는 HTTP response header 의 필드에 대해서 좀더 살펴보면 다음과 같다.

?? Date : General Header 의 Date 는 메시지가 만들어지는 날짜와 시간을 나타낼 때 쓰이며, RFC 822[32]에서 정의하고 있는 orig-date 와 같은 의미이다. 이의 예는 다음과 같다.

Date : Wed, 15 Oct 1999 08:12:31 GMT

?? Expires : Entity Header 의 Expires 필드는 전달하는 데이터를 의미없는 대상으로 간주하는 시기를 표시한다. 다시 말해, 식품의 유효기간 표시와 같이 일정 기간 유효한 대상에 대해 그 시각을 지나서는 유효하지 않다고 지정할 때 사용하는 것이다. 만약 캐시되

어 있는 데이터에 대해 이렇게 표시되어 있을 때라면 지정한 시각이 지나고 난 다음에는 캐시되어 있는 데이터를 지워도 되는 것이다. Expires 의 예는 다음과 같다.

Expires : Mon, 01 Nov 1999 16:00:00 GMT

?? Last-Modified : Entity Header 필드에 들어가는 Last-Modified 필드는 송신측에서 이 문서의 마지막 작업 일자와 시간을 알려주는 용도로 쓰인다. 이의 정확한 의미는 수신측이 이 문서를 어떻게 처리해야 하는가를 알려주는 것이다. 만약 수신측에서 수신하는 문서의 Last-Modified 날짜가 수신측에 똑같이 저장되어 있는 복사본 문서의 날짜 이후 것이라면 수신측은 앞서 저장된 문서를 무효로서 인식하여 삭제하든지 대치하든지 구현상의 문제로서 적절히 처리한다. 이의 예는 아래와 같이 보일 수 있다.

Last-Modified : Tue, 14 Oct 1999 12:45:26 GMT

4.1.2 HTTP/1.1 Request Header

HTTP/1.0 프로토콜의 특징은 단순성에 있다. 그래서 연결을 만들고 동작하고 연결을 해제하는 단순한 과정으로 구성되어 있으며, 하나의 URL 은 하나의 TCP 연결이 되도록 만들어져 있다. 그러나 이것은 연결의 설립/동작/해제의 단순 동작이 계속 반복됨으로 해서 네트워크의 congestion 에 대한 정보를 확보할 수가 없었다. 연결이 계속 지속되어 있다면 정보 축적과 분석을 통해 트래픽의 혼잡을 인식할 수 있는 것이다. 또한 잦은 연결 설립과 동시에 여러 개의 연결을 설립하는 동작을 통해 대역폭이 낮은 링크에서 congestion 문제의 가능성을 높이고 사용자에게는 불만족스런 성능을 제공해주게 된다. 캐시 모델의 미흡함도 문제이다. 캐싱에 관해 설계한 모델이 초기적인 형태로 구성되었기 때

문에 동작상의 오버헤드도 만들고 캐시된 데이터 관리에도 문제를 갖고 있었던 것이다. 이러한 문제들을 해결하고, 추가적인 고려 사항을 반영한 HTTP/1.1 프로토콜을 설계되었다[23]. 여기서는 HTTP/1.1 의 전체 규격에 대한 상세 소개는 하지 않고, Squid 의 재생 알고리즘에서 사용되는 Cache-Control Request Header 의 max-age 에 대해서만 설명하기로 한다.

HTTP/1.0 규격에서도 HTTP 프로토콜 동작 상의 데이터 캐싱에 대하여 다루고 있다. HTTP/1.1 에서는 이 기능이 보다 잘 동작하게 하기 위한 많은 요소들이 추가되었는데, HTTP/1.1 의 General Header 에 새로이 Cache-Control 필드도 그 중 하나이다. Cache-Control 필드는 캐싱 메커니즘을 잘 수행하기 위한 특정한 지시문 (directive)으로써 사용된다. 이들 캐시 지시문 (cache directive)들은 반드시 프락시나 게이트웨이를 통해서 전달되어야 한다. Cache-Control 의 형식은 그림 11 과 같다. 그림 11 은 cache-response-diective 에 대한 내용은 삭제된 Cache-Control 형식이다.

```
Cache-Control   = "Cache-Control" ":" 1#cache-directive

cache-directive = cache-request-directive        | cache-response-directive

cache-request-directive =
                        "no-cache"
                        | "no-store”
                        | "max-age" "=" delta-seconds
                        | "max-stale" [ "=" delta-seconds ]
                        | "min-fresh" "=" delta-seconds
                        | "no-transform"
                        | "only-if-cached"
                        | cache-extension

cache-extension = token [ "=" ( token | quoted-string ) ]
```

그림 11. Cache-Control General Header Field

위의 그림에서 cache-request-directive 의 “max-age”가 Squid 의 재생 알고리즘에서 사용되는데 이것은 클라이언트가 개별적으로 정해주는 웹 객체의 최대 객체 나이 (maximum object age)이다. 만일 웹 서버에 의해서 제공되는 응답 메시지의 나이가 “max-age”보다 작으면 이 응답 메시지는 유효한 것이고, 만일 “max-age”보다 크면 이 응답 메시지는 유효하지 않으므로 클라이언트쪽에서 받아들이지 않게 된다. “max-age”는 Squid 의 재생 알고리즘에서 CLIENT_MAX_AGE 로 표현된다.

4.1.3 Squid 의 재생 알고리즘

본 논문에서 제안된 프리페칭 알고리즘은 캐싱 서버에 저장된 웹 객체들 중에서 프리페칭이 가능한 웹 객체를 선택하는데, 먼저 유효기간이 지난 웹 객체를 선택한다. 왜냐하면 캐싱 서버는 클라이언트가 요청한 웹 객체가 자신의 캐시에 저장되어 있어도 그 웹 객체의 유효기간 (expiration time)이 지났다고 판명되면 캐시 미스 (cache miss)로 보고 해당 웹 서버에게 새로이 요청을 보내기 때문이다. 이러한 웹 객체의 유효성을 판명하기 위해서 Squid 에서는 재생 알고리즘을 제공하고 있다. 그림 13 은 Squid 에서 캐시에 저장된 객체를 언제 재생할 것인가를 결정하는 재생 알고리즘을 순서도로 표현한 것이다.

조건식 중에서 최초로 일치하는 엔트리를 사용하며 만일 일치하는 엔트리가 없다면 default value 를 사용한다. OBJ_DATE 는 웹 서버에 의해서 응답 메시지가 만들어진 시간을 나타낸다. OBJ_LASTMOD 는 그 객체가 마지막으로 갱신된 시간을 나타내며 Expires 는 웹 서버에서 정해주는 웹 객체에 대한 유효기간을 나타낸다. 각각의 OBJ_DATE, OBJ_LASTMOD, Expires 는 HTTP Respons Header 로부터 가져올 수 있다 [22]. CLIENT_MAX_AGE 는 클라이언트가 개별적으로 정해주는 웹 객체의 최대 객체 나이이며 HTTP/1.1 Cache-Control Request Header 로부터 알

수 있다[23]. 또한 Squid 는 refresh_pattern 규칙을 통해 CONF_MIN, CONF_PERCENT, CONF_MAX 의 값을 squid.conf 에서 기본적으로 정해 줄 수 있다. 이들을 HTTP Response Header 에 Expires 에 해당하는 값이 없을 경우 default 로 Expires 값을 결정하기 위해 사용된다[7]. NOW 는 현재 시간을 나타내며, OBJ_AGE 는 해당 웹 객체가 웹 서버에 의해서 만들어진 후의 경과시간을 나타낸다. LM_AGE 는 해당 웹 객체가 만들어졌을 때 최근에 갱신된 이후에 얼마나 오래되었는가를 나타낸다. LM_FACTOR 는 OBJ_AGE 를 LM_AGE 로 나눈 값이다.

그림 12 는 재생 알고리즘에서 사용되는 파라미터들을 정리한 것이다. OBJ_DATE, Expires, OBJ_LASTMOD 의 단위는 UNIX time 에서 millisecond 를 포함한 시간 (예: Fri Oct 15 04:00:27 1999 GMT = 939927627.725)이고 CLIENT_MAX_AGE, OBJ_AGE, LM_AGE, LM_FACTOR 의 단위는 second 이다. 또한 CONF_MAX 와 CONF_MIN 은 minute 을 단위로 사용하며 CONF_PERCENT 의 단위는 %이다.

OBJ_DATE	: 메시지가 만들어지는 날짜와 시간을 표시
Expires	: 전달하는 데이터를 의미없는 대상으로 간주하는 시기를 표시
OBJ_LASTMOD	: 송신측에서 이 문서의 마지막 작업 일자와 시간을 표시
CLIENT_MAX_AGE	: 클라이언트가 개별적으로 정해주는 최대 객체 나이를 표시
OBJ_AGE = NOW - OBJ_DATE (sec)	
LM_AGE = OBJ_DATE - OBJ_LASTMOD (sec)	
LM_FACTOR = OBJ_AGE / LM_AGE (sec)	
CONF_MAX	: 4320 (min , 3 days)
CONF_MIN	: 0 (min)
CONF_PERCENT	: 0.2 (20%)

그림 12. 재생 알고리즘에서 사용되는 파라미터

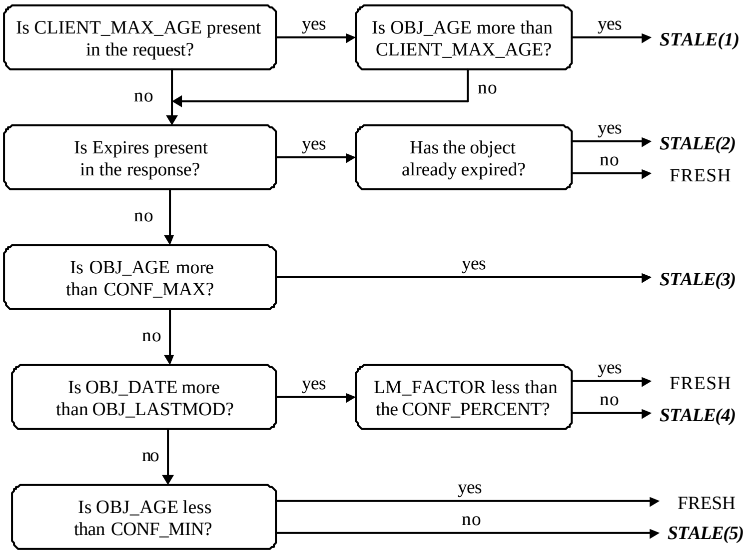

그림 13. 재생 알고리즘의 순서도

그림 14 는 재생 알고리즘에서 웹 객체가 'STALE'한 5 가지의 경우를 예로 설명한 것이다. 편의상 Expires, OBJ_DATE, OBJ_LASTMOD, NOW 는 HTTP 프로토콜에서 나타내는 단위를 사용하였다. 첫번째의 경우는 클라이언트가 개별적으로 웹 객체의 최대 나이를 선언하여 요청 메시지로부터 CLIENT_MAX_AGE 의 값을 알 수 있었고 이 값이 100 sec 였고 OBJ_AGE 는 120 sec 일 경우 OBJ_AGE 가 CLIENT_MAX_AGE 보다 크므로 이 웹 객체는 'STALE' (STALE(1))하다. 두 번째의 경우는 첫번째의 엔트리를 만족하지 못하면서 응답 메시지에 Expires 필드가 세팅되어 있고 이 값이 NOW 보다 작을 경우, 즉 유효기간이 지났을 경우이다 (STALE(2)). 세 번째는 위의 두 엔트리를 만족하지 못하면서 웹 객체의 OBJ_AGE 가 CONF_MAX 보다 클 경우이다 (STALE(3)). 네 번째 경우는 위의 3 가지 경우를 만족하지 못하며 OBJ_DATE 가 OBJ_LASTMOD 보다 크고 LM_FACTOR (30%)가 CONF_PERCENT (20%)보다 작지 않을 경우이다 (STALE(4)). 다섯번째의 경우는 위의 네 가지 모든 경우를 만

족하지 못하며 OBJ_AGE (5760 min)가 CONF_MIN (0 min)보다 클 경우 해당 웹 객체를 'STALE'로 정의한다 (STALE(5)).

STALE(1)의 예 CLIENT_MAX_AGE : 100 (sec) OBJ_AGE : 120 (sec)
STALE(2)의 예 EXPIRES : Fri, 01 Oct 1999 16:00:00 GMT NOW : Sun, 03 Oct 1999 12:01:20 GMT
STALE(3)의 예 OBJ_AGE : 5760 min (4 days) CONF_MAX :4320 min (3 days)
STALE(4)의 예 OBJ_LASTMOD : Wed, 29 Sep 1999 16:00:00 GMT OBJ_DATE : Fri, 01 Oct 1999 16:00:00 GMT LM_FACTOR : 30 % CONF_PERCENT : 20 %
STALE(5)의 예 OBJ_AGE : 5760 min (4 days) CONF_MIN : 0 min

그림 14. 재생 알고리즘에서 웹 객체가 STALE 한 경우

4.1.4 Freshness 결정

캐시에 저장되어 있는 웹 객체는 그것의 유효기간이 지나면 캐시에서 제거된다. 만일 웹 객체가 재생 알고리즘에 의해 'FRESH'라고 판명이 나면 아직 유효한 것이고, 만일 'STALE'이라고 판명되면 유효성이 지나서 사용할 수 없는 것이다. 따라서 'STALE'한 웹 객체는 프리페칭 대상이 될 수 있다. 웹 객체들의 유효기간이 유효성를 결정하기 위하여 본 논문에서는 squid 의 재생 알고리즘을 응용하여 사용하였다. OBJ_DATE, OBJ_LASTMOD, Expires 는 squid 가 제공하는 store.log 를 통해서 얻을 수 있고, CONF_MAX, CONF_MIN, CONF_PERCENT 또한 squid 가 제공하는 squid.conf 를 통해서 알 수 있다. CLIENT_MAX_AGE 는 로그를 통해 얻을 수 없고 각각의 클라이언트마다 정해주는 값이 다르므로 일반화 시킬 수 없다. 따라서 이 값은 사용하지 않기로 한다. 그림 15 은 Squid 의 "access log"의 형식을, 그림 16 은 Squid 의 "store log"의 형식을 나타낸다.

Time	Elapsed	Remotehost	Code/Status	Byte	Method	URL

그림 15. Access Log Format

각각의 필드의 의미는 다음과 같다.

?? Time 은 해당 로그가 기록된 시간이며 단위는 UNIX time 에 milliseconds 를 포함시킨 것이다.

?? Elapsed 는 클라이언트와 연결 (connection)이 지속된 시간이다.

?? Remotehost 는 웹 객체를 요청한 클라이언트의 IP 주소를 나타낸다.

?? Code/Status 에서 Code 는 요청에 대한 캐시 결과를 나타낸다 (예: TCP_HIT, TCP_MISS, etc). Status 는 HTTP status code 를 나타낸다.

?? Byte 는 클라이언트에게 전달된 응답 메시지의 총 바이트수를 나타낸다.

?? Method 는 HTTP request 에 나타나 있는 Method (예: GET, HEAD, POST)를 표현한다.

?? URL 은 요청된 URL 이다.

Time	Action	Status	OBJ_DATE	OBJ_LASTMOD	Expires	Type	Len	Method	URL

그림 16. Store Log Format

"store log"의 각각의 필드를 설명하면 아래와 같다. Time, Status, Method, URL 은 "access log"의 필드와 같은 의미이고 OBJ_DATE, OBJ_LASTMOD, Expires 는 4.1.3 절에서 설명되어 있다.

?? Action 은 RELASE, SWPIN, SWPOUT 중에 하나의 값을 가지며 RELASE 는 해당 웹 객체가 캐시에서 삭제되었다는 것을 의미하고 SWAPOUT 은 캐시에 저장되어 있는 상태라는 것을 의미하며 SWAPIN 은 캐시에 저장되어 있을 뿐만 아니라 캐싱 서버의 메모리에도 저장되어 있다는 것을 의미한다.

?? Type 은 응답 메시지의 종류를 나타내는데, 예를 들어 text/html, image/gif 와 같이 표현된다.

?? Len 은 expect-len 과 real-len 으로 구성되어 있는데, expect-len 은 응답 메시지의 HTTP Content-Length Response Header 의 값을 나타내며, real-len 은 캐싱 서버에게 실제로 읽혀진 바이트수를 나타낸다.

웹 객체의 유효성을 검사할 때 고려되어야 될 사항은 NOW, 즉 현재를 어느 시점으로 볼 것인가를 결정하는 방법이다. 다시 말해 웹 문서가 fresh 한지 stale 한지를 결정할 때 현재의 입력 데이터로 사용될 "access log" 중에서 stale 한 것과 다음 프리페칭이 실행되기 전까지 stale 한 것을 함께 고려해야 한다. 따라서 Squid 의 재생 알고리즘을 따르되 NOW 는 프리페칭 주기를 더한 값을 사용한다.

$$NOW ? Current\,PrefetchngTime ? PrefetchingFrequency$$

$Current\,PrefetchingTime$: **현재의 프리페칭 적용 시간**
$PrefetchingFrequency$: **프리페칭 주기**

본 논문에서는 하루 중에서 off-peak periods **의 남는 대역폭을 사용하기 때문에 프리페칭 주기** (prefetching frequency)**를 하루로 결정하였다. 그림** 17 **은 프리페칭 주기를 설명한 것인데 가장 최근에 프리페칭을 수행한 시간은** 10 **월** 15 **일** 04 **시이며 현재의 프리페칭 시간은** 16 **일** 04 **시, 다음에 프리페칭할 시간은** 17 **일** 04 **시이다. 현재** (16 **일** 04:00) **프리페칭을 하기 위하여** 10 **월** 15 **일** 04:00 **부터** 10 **월** 16 **일** 04:00 **까지의** "access log"**와** "store log"**를 입력 데이터로 사용한다. 따라서 프리페칭 주기는 하루이다.**

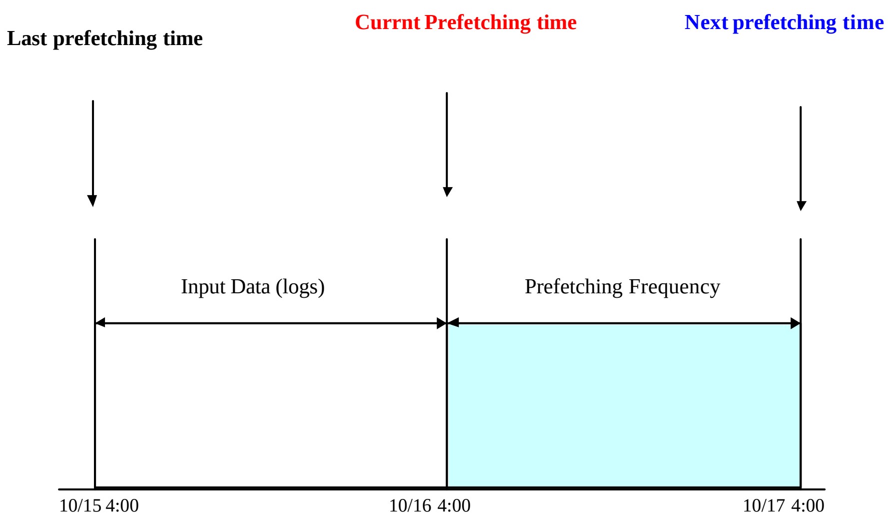

그림 17. 프리페칭 주기

4.2 Request Generator

Request generator 는 Prefetchable Obejct List 에 있는 웹 객체를 off-peak periods 시에 캐싱 서버에 저장되도록 요청하는 모듈이며 하나의 클라이언트로써 동작한다. Prefetchable Object List Generator 에 의해 생성된 Prefetchable Object List 에 따라 프리페칭할 request 을 생성하여 off-peak periods 동안에 해당되는 웹 서버에게 request 를 보낸다. 본 논문에서는 HTTP 를 지원하는 command-line Web client 로써 동작하는 "wget"[19] 을 사용하였다. 또한 crontab 을 이용하여 자동으로 off-peak periods 동안에 프리페칭할 수 있도록 하였다.

Request Generator 를 이용하여 Off-peak periods 동안 프리페칭을 수행할 경우 고려해야 할 사항으로 프리페칭으로 인해 병목현상이 발생되는 것을 방지해야 한다. 따라서 프리페칭을 수행하는 동안에 발생되는 대역폭량이 여유분의 off-peak 대역폭량을 넘지 않도록 해야 한다. 이를 위해서 WAN 트래픽을 처리하는 라우터와 같은 네트워크 장치에서 제공하는 SNMP agent 를 이용한다. SNMP agent 를 주기적으로 폴링(polling) 함으로써 현재 사용되는 대역폭량을 알 수 있는데, 이 값을 이용하여 프리페칭에 사용되는 대역폭량을 조절한다.

5. 시뮬레이션

본 논문에서 제시한 프리페칭 알고리즘의 성능을 평가하기 위하여 시뮬레이션 모델을 설계하였다. 시뮬레이션을 하는 목적은 첫째로 본 논문에서 제시한 프리페칭 기법을 사용하였을 경우와 사용하지 않았을 경우의 캐싱 서버의 성능을 비교하기 위함이고 둘째로는 프리페칭 파라미터의 조건에 따라 프리페칭의 성능을 비교함으로써 본 논문에서 제시한 프리페칭 알고리즘의 유효성을 입증하기 위함이다.

다음은 시뮬레이션에을 수행할 때 사용되는 입력 파라미터 (input parameter)와 프리페칭할 웹 객체의 선택 기준에 대하여 설명하고 마지막으로 시뮬레이션을 위한 모델과 시뮬레이션 과정에 대하여 설명한다.

5.1 시뮬레이션 입력 파라미터

시뮬레이션은 trace-driven 이며 현재 포항공대에서 운영되고 있는 캐싱 서버인 CacheRaQ[29]에서 제공하는 로그를 이용하였다. 이 캐싱 서버는 포항공대 기숙사중에서 16 개의 서브넷 (subnet)에서 발생되는 웹 트래픽을 처리하고 있으며 내부적으로는 Squid 를 사용하고 있다. 시뮬레이션을 수행할 때 사용되는 입력 파라미터 (input parameter)는 아래와 같다.

?? Logs : 시뮬레이션에서 입력 데이터로 사용된 로그는 1999 년 10 월 15 일부터 10 월 28 일 2 주간의 "access log"와 "store log"이다.

?? Off-peak periods : 3.5 절에서 설명된 대로 off-peak periods 는 04:00 부터 13:00 까지이다. 따라서 프리페칭의 수행이 시작되는 시간은 04:00 이다.

?? 프리페칭할 웹 객체의 선택 기준 : 3.3 절의 프리페칭 파라미터를 기반으로 프리페칭할 웹 객체를 선택한다. 자세한 설명은 5.2 절에서 하겠다.

?? 재생 알고리즘 파라미터 : 캐싱 서버에 저장되어 있는 웹 객체 중에서 유효기간이 지났는가를 확인하기 위해 Squid 의 재생 알고리즘을 사용하는데, 이때 사용되는 파라미터는 4.1.3 절에 정의한 것을 사용한다. 그러나 NOW 는 4.1.4 절에서 설명된 대로 현재의 프리페칭 시간에다 프리페칭 주기를 더한값이다.

5.2 프리페칭이 가능한 웹 객체의 선택 기준

본 논문에서 제안하는 프리패칭 알고리즘의 성능을 평가하고 유효성을 입증하기 위해서 시뮬레이션을 수행한다. 프리페칭될 웹 객체를 선정하기 위해서는 프리페칭 파라미터와 웹 트래픽을 추적 (Web traffic trace)한 결과를 고려해야 하는데, 위에서 설명된 프리페칭 파라미터 중에서 프리페칭 대상 선정 기간과 참조 횟수, 그리고 프리페칭될 웹 객체 (Prefetchable Web Objects)의 총량을 기준으로 하고 웹 트래픽 추적을 분석한 결과를 기반으로 하여 프리페칭이 가능한 웹 객체를 결정하면 다음과 같다.

첫번째로 프리페칭 대상 선정 기간으로는 최근 하루동안의 웹 트래픽을 대상으로 하였다. 웹 트래픽 추적의 결과로부터 우리는 웹 트래픽 접근 패턴이 하루를 주기로 변한다는 것을 알 수 있었다. 따라서 우리는 최근의 과거 하루 동안의 로그를 이용하도록 하였다.

두 번째로 웹 객체의 유효기간 (expiration time)이 지난 객체들 중에서 참조 횟수는 1 회 이상, 2 회 이상, 3 회 이상, 4 회 이상, 5 회 이상인 웹

객체들을 대상으로 선택한다. 여기서 참조 횟수를 하나로 지정하지 않는 이유는 참조 횟수에 따른 웹 객체들의 총량과 실제로 프리페칭을 통해서 얻게 될 이득과의 관계를 시뮬레이션을 통해서 분석한 후 결정하기 위함이다. 전체를 대상으로 프리페칭을 하기보다는 참조 횟수가 높은 웹 객체들을 프리페칭 했을 때 프리페칭의 성능을 더 높아질 것이라고 예상되나 자세한 비교 분석은 뒤에서 하기로 한다.

세 번째로 프리페칭될 웹 문서의 전체 크기는 off-peak periods 동안의 남는 대역폭 용량보다 크지 않게 한다.

5.3 시뮬레이션 모델

시뮬레이션 모델은 그림 18과 같다. Prefetchable Object List Generator 는 off-peak periods 동안에 프리페칭할 웹 객체를 선정하여 prefetchable object list 를 생성하고 performance analyzer 는 prefetchable object list 를 이용하여 캐싱 서버가 프리페칭을 수행했을 경우와 프리페칭을 하지 않았을 경우의 각각의 성능 평가 척도를 분석한다.

시뮬레이션을 통해서 프리페칭을 사용했을 경우와 사용하지 않았을 경우의 성능을 비교 분석하기 위한 성능 평가 척도로는 request saving 과 bandwidth saving 이며 프리페칭의 유효성을 측정하기 위한 성능 평가 척도로는 accracy 와 wasted bandwidth 이다.

Prefetchable Object List Generator 는 위에서 설명된 프리페칭 파라미터와 프리페칭할 웹 객체의 선택 기준을 바탕으로 하루를 주기로 “access log” 와 “store log”를 이용하여 prefetchable object list 를 생성한다.

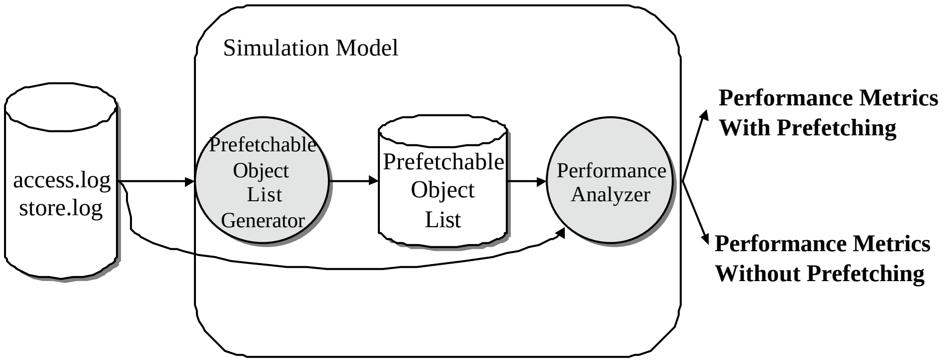

그림 18. 시뮬레이션 모델

Performance Analyzer는 prefetchable object list와 다음날의 "access log"를 입력 데이터로 받아 prefetchable object list에 들어있는 웹 객체가 다음 날의 "access log"에도 기록되어 있으면 성능 평가 척도 중에서 request saving (캐시 객체 적중률), bandwidth saving (캐시 바이트 적중률)과 accuracy (프리페칭 객체 적중률, 프리페칭 바이트 적중률)를 증가 시킨다. 만일 다음 날의 "access log"에 들어 있지 않다면 wasted bandwidth를 증가 시킨다. 또한 프리페칭을 사용한 캐싱 서버와 프리페칭을 사용하지 않는 캐싱 서버와의 request saving과 bandwidth saving을 비교하고 참조 횟수별로 프리페칭의 성능을 accuracy와 wasted bandwidth 별로 비교하여 본 논문에서 제시한 프리페칭 알고리즘의 유효성을 조사한다.

6. 결과

이번 장에서는 앞에서 설명된 시뮬레이션 환경과 방법에 기초하여 3장에서 설명된 성능 평가 척도에 적용하여 웹 프리페칭의 성능을 비교 분석하고 유효성을 측정한 결과이다. 먼저 프리페칭의 효율성을 측정하기 위해 accuracy 와 wasted bandwidth 의 시뮬레이션 결과를 알아보며, performance parameter 중에서 request saving 과 bandwidth saving 을 통해서 프리페칭을 사용했을 경우와 사용하지 않았을 경우의 성능을 비교 분석하였다.

6.1 Accuracy

Accuracy 는 프리페칭 객체 적중률과 프리페칭 바이트 적중률로 구분하여 비교하였다. 표 4 는 10 월 15 일부터 10 월 28 일까지의 로그 데이터를 이용하여 위에서 설명된 프리페칭 시스템을 이용하여 daily prefetchable object list 를 생성한 후 실제로 프리페칭이 적용되는 다음날부터인 10 월 16 일부터 10 월 29 일까지의 프리페칭 객체 적중률을 나타낸 것이다. 표에도 나타나 있듯이 참조 횟수가 1 회인 경우가 15.22%로 가장 낮은 프리페칭 객체 적중률을 보였으며 5 회 이상인 경우는 87.14%로 프리페칭된 대부분의 웹 객체가 다음날 요청된 것으로 나타났다. 또한 참조 횟수가 1 회인 prefetchable object 는 전체의 50.64%를 나타내고 있지만 다음날 요청된 웹 객체는 전체중에 19.55%로 매우 낮아졌다. 참조 횟수가 높아짐에 따라 다음날 요청될 확률도 linear 하게 지속적으로 증가하는 것을 알 수 있었다.

표 4. 참조 횟수별 프리페칭 객체 적중률

참조 횟수	프리페칭된 평균 웹 객체수 (%)	프리페칭된 웹 객체 중에서 다음날 적중된 평균 웹 객체수 (%)	프리페칭 객체 적중률
1 회	16,274 (50.64 %)	2,476 (19.55 %)	15.22 %
2 회	5,525 (17.2 %)	2,286 (18.05 %)	41.38 %
3 회	3,008 (9.36 %)	1,804 (14.24 %)	59.96 %
4 회	1,966 (6.12 %)	1,430 (11.29 %)	72.76 %
5 회 이상	5,358 (16.68 %)	4,469 (36.87 %)	87.14 %
Total	32,131 (100.00 %)	12,665 (100.00 %)	39.42 %

그림 19 는 참조 횟수별 평균 프리페칭된 웹 객체수와 프리페칭된 웹 객체 중에서 다음날 요청된 웹 객체수를 참조 횟수별로 나타낸 그림이다. 참조 횟수가 높아 질수록 대부분의 프리페칭된 웹 객체가 다음날 요청되어 높은 prefetched object hit 을 보여주는 것을 알 수 있다. 그림 20 은 참조 횟수별 프리페칭 객체 적중률을 그래프로 나타낸 것이다.

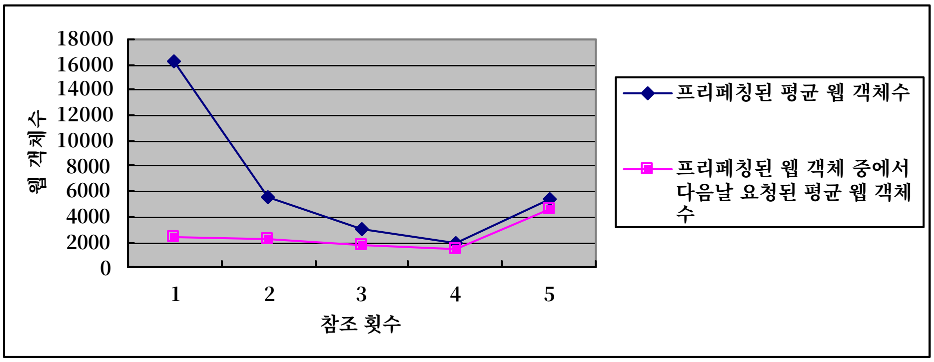

그림 19. 참조 횟수별 평균 프리페칭된 웹 객체수와 실제로 다음날 요청된 평균 웹 객체수

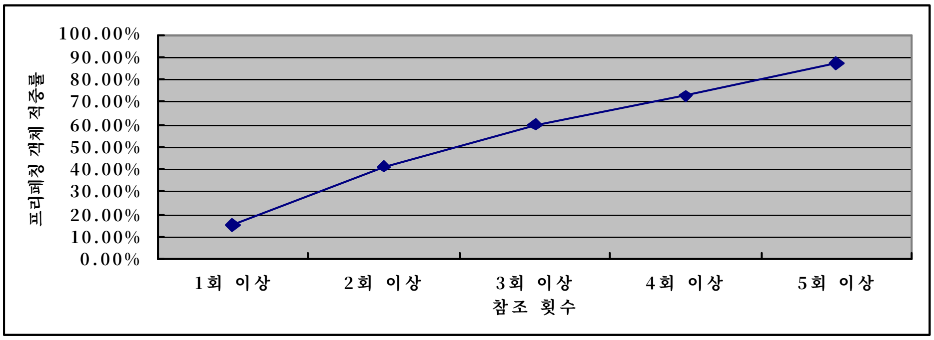

그림 20. 참조 횟수별 프리페칭 객체 적중률

표 5 는 위와 같은 방법으로 프리페칭 바이트 적중률을 나타낸 도표이다. 프리페칭 객체 적중률과 마찬가지로 참조 횟수가 1 회인 경우가 8.68%로 가장 낮았고 역시 5 회 이상인 경우가 72.64%로 가장 높게 나타났다. 참조 횟수별로 프리페칭 객체 적중률보다 프리페칭 바이트 적중률의 증가폭의 차이가 크진 않았지만 프리페칭의 accuracy 는 참조 횟수의 증가에 따라 높아진다는 것을 알 수 있었다.

표 5. 참조 횟수별 프리페칭 바이트 적중률

참조 횟수	프리페칭된 평균 웹 객체의 총량 (%)	프리페칭된 웹 객체 중에서 다음날 적중된 평균 웹 객체의 총량 (%)	프리페칭 바이트 적중률
1 회	545,399,428 (63.43%)	47,336,276 (25.02%)	8.68 %
2 회	154,356,860 (17.95%)	56,170,230 (29.7%)	36.39 %
3 회	64,124,072 (7.46%)	25,570,556 (13.52%)	39.89 %
4 회	33,972,272 (3.95%)	15,026,692 (7.94%)	44.23 %
5 회 이상	62,024,975 (7.21%)	45,057,416 (23.82%)	72.64 %
Total	859,877,607 (100%)	189,161,170 (100%)	22 %

그림 21 은 참조 횟수별 평균 프리페칭된 웹 객체의 총량과 프리페칭

된 웹 객체 중에서 다음날 요청된 웹 객체의 총량을 참조 횟수별로 나타낸 그림이다. 참조 횟수가 높아 질수록 대부분의 프리페칭된 웹 객체의 총량 중에서 다음날 요청된 웹 객체의 총량도 웹 객체수와 마찬가지로 증가되는 것을 알 수 있었다. 그림 22 는 참조 횟수별 프리페칭 바이트 적중률을 그림으로 나타낸 것이다.

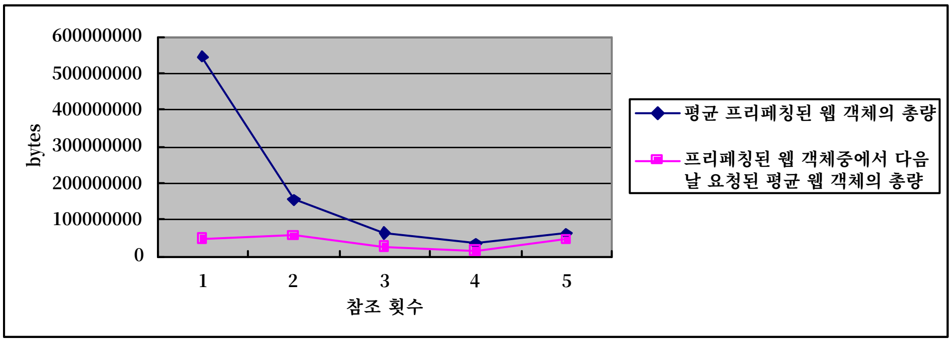

그림 21. 참조 횟수별 평균 프리페칭된 웹 객체의 총량과 실제로 다음날 요청된 평균 웹 객체의 총량

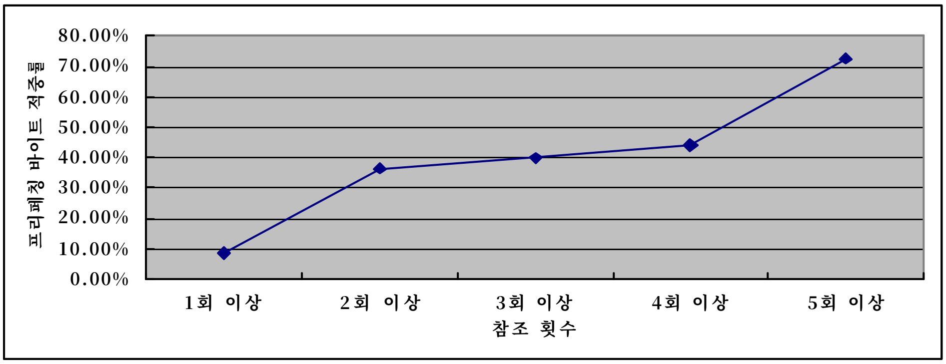

그림 22. 참조 횟수별 프리페칭 바이트 적중률

표 6은 참조 횟수를 1회 이상, 2회 이상, 3회 이상, 4회 이상 5회 이상으로 프리페칭을 수행했을 경우의 정확도를 설명하고 있다. 참조 횟

수가 2 회 이상인 웹 객체만을 프리페칭하였을 경우 프리페칭 객체 적중률은 64.26%였으며 참조 횟수가 1 회 이상인 웹 객체도 함께 프리페칭했을 경우 39.42%로 약 25% 정도 떨어졌다. 프리페칭 바이트 적중률을 살펴보면 참조 횟수가 2 회 이상인 웹 객체만을 프리페칭했을 경우 46.1%였으며 1 회 이상으로 확대하여 프리페칭했을 경우는 22%로 나타났다. 종합하여 볼 때 참조 횟수를 높여서 프리페칭을 할수록 정확도가 높아진다는 것을 알 수 있었다. 그림 23 은 참조 횟수별 정확도를 그래프로 표현한 것이다.

표 6. 참조 횟수별 정확도

참조 횟수	프리페칭 객체 적중률	프리페칭 바이트 적중률
1 회 이상	39.4 %	22 %
2 회 이상	64.3 %	45.1%
3 회 이상	76.5 %	53.5 %
4 회 이상	83.3 %	62.6 %
5 회 이상	87.1 %	72.6 %

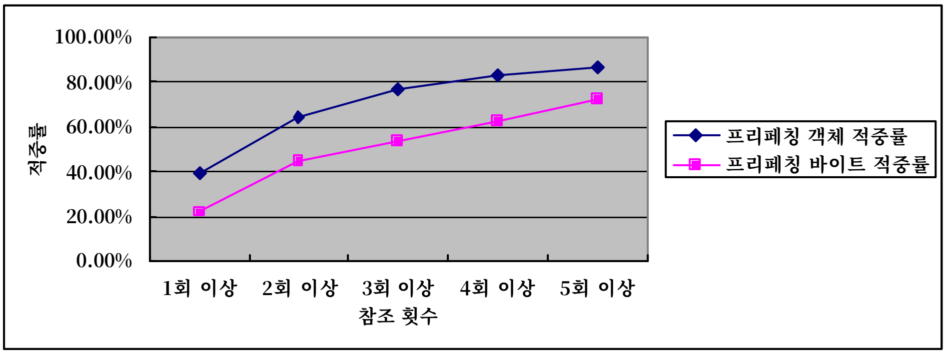

그림 23. 참조 횟수별 프리페칭 객체 적중률과 프리페칭 바이트 적중률

6.2 Wasted Bandwidth

Wasted bandwidth 를 참조 횟수 별로 나타내면 표 7과 같다. 하루 평균 낭비되는 대역폭은 참조 횟수가 1 회 이상인 웹 객체를 대상으로 프리페칭했을 경우 670.72 Mbyte 로 전체 웹 트래픽 (3775.043 Mbyte)의 17.8%를 더 증가시키는 결과를 가져오며 참조 횟수가 2 회 이상인 웹 객체 만을 대상으로 프리페칭했을 경우에는 172.65 Mbyte 로 단지 전체 웹 트래픽의 4.6%만을 증가시켰다. 이것은 참조 횟수가 2 회 이상으로 프리페칭했을 때보다 1 회 이상으로 했을 경우가 약 4 배정도 더 많은 대역폭이 낭비된다는 것을 보여준다.

표 7. 참조 횟수별 낭비되는 대역폭

참조 횟수	하루 평균 낭비되는 대역폭	백분율
1 회 이상	670.72 M	17.8 %
2 회 이상	172.65 M	4.6 %
3 회 이상	74.47 M	2.0 %
4 회 이상	35.91 M	1.0 %
5 회 이상	16.97 M	0.5 %

본 논문에서는 off-peak periods 동안에 프리페칭을 한다. 따라서 off-peak periods 의 대역폭 사용량은 증가된다. 하루 평균 off-peak 대역폭 사용량은 796.356 Mbyte 이다. 따라서 참조 횟수를 1 회 이상인 웹 객체를 대상으로 프리페칭하였을 경우 off-peak 대역폭 사용량의 51.9%만큼 증가되었고, 프리페칭시 낭비되는 대역폭은 off-peak periods 동안의 대역폭 사용량을 2 회 이상으로 수행했을 경우 28.3%, 3 회 이상으로 수행했을 경우 16.7%, 4 회 이상일 경우는 10.8%, 마지막으로 5 회 이상으로 수행했을 경우는 7.2%로 가장 낮았다. 표 8 은 참조 횟수별 off-peak periods 동안에 증가되는 대역폭 사용량 증가율을 나타낸 것이며 그림 24 는 그래

프로 표현한 것이다.

표 8. 참조 횟수별 Off-peak Periods 동안의 대역폭 사용량 증가율

참조 횟수	Off-peak periods 동안의 대역폭 사용량 증가율
1 회 이상	51.9 %
2 회 이상	28.3 %
3 회 이상	16.7 %
4 회 이상	10.8 %
5 회 이상	7.2 %

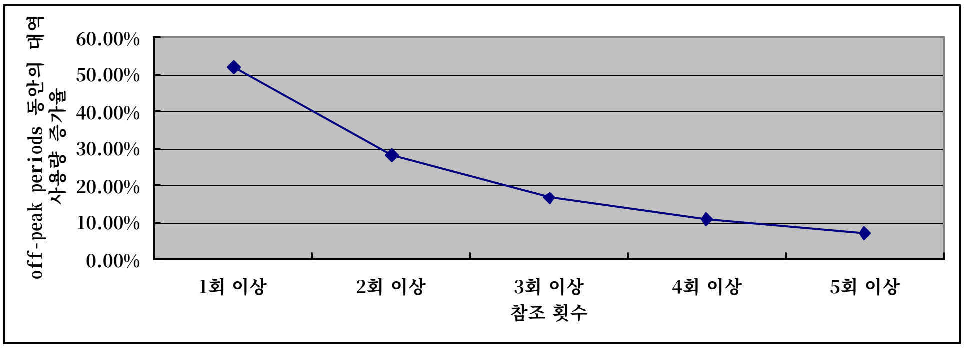

그림 24. 참조 횟수별 하루 평균 Off-peak Periods 동안의 대역폭 사용량 증가율

6.3 Request Saving

표 9 는 프리페칭을 사용하지 않은 캐싱 서버의 캐시 객체 적중률과 참조 횟수별 프리페칭을 사용한 캐싱 서버의 캐시 객체 적중률을 나타낸 것이다. 프리페칭을 수행하지 않았을 경우 캐시 객체 적중률은 55% 였고, 참조 횟수가 1 회 이상인 웹 객체를 대상으로 프리페칭을 수행했

을 경우 증가된 캐시 객체 적중률은 4.3%로 전체 캐시 객체 적중률은 59.3%를 나타내었다. 참조 횟수가 2회 이상인 경우의 증가율은 3.53 %, 3회 이상인 경우의 증가율은 2.72%, 4회 이상인 경우의 증가율은 2.1%, 참조 횟수가 5회 이상인 경우의 증가율은 1.61%였다. 참조 횟수를 1회 이상으로 했을 경우에는 참조 횟수를 2회 이상으로 프리페칭했을 때보다 0.77% 증가하였다. 이것은 프리페칭한 웹 객체 수에 비해 매우 적은 증가율을 나타낸 것이다.

표 9. 프리페칭을 수행했을 경우와 수행하지 않았을 경우의 캐시 객체 적중률

캐시 객체 적중률 (캐싱)	55 %
캐시 객체 적중률 (캐싱+프리페칭) (참조 횟수 1회 이상)	59.3 %
캐시 객체 적중률 (캐싱+프리페칭) (참조 횟수 2회 이상)	58.5 %
캐시 객체 적중률 (캐싱+프리페칭) (참조 횟수 3회 이상)	57.7 %
캐시 객체 적중률 (캐싱+프리페칭) (참조 횟수 4회 이상)	57.1 %
캐시 객체 적중률 (캐싱+프리페칭) (참조 횟수 5회 이상)	56.7 %

그림 25는 프리페칭을 수행하지 않았을 경우의 캐시 객체 적중률과 객체 참조 횟수별로 프리페칭을 수행했을 경우의 캐시 객체 적중률을 그래프로 나타낸 것이다.

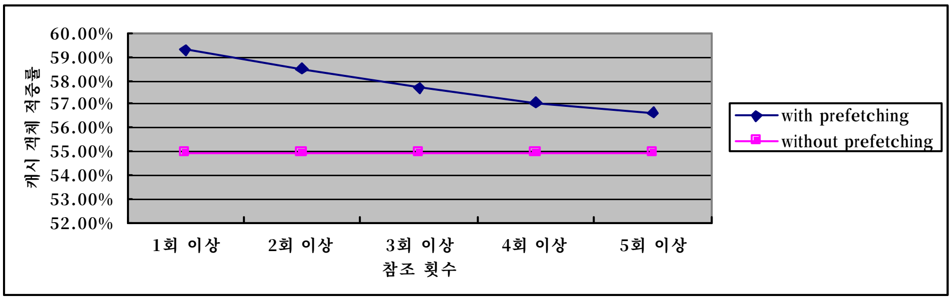

그림 25. 참조 횟수별로 프리페칭을 수행했을 경우와 프리페칭을 수행하지 않았을 경우의 캐시 객체 적중률

6.4 Bandwidth Saving

표 10 은 프리페칭을 사용하지 않은 캐싱 서버의 캐시 바이트 적중률과 참조 횟수별 프리페칭을 사용한 캐싱 서버의 캐시 바이트 적중률을 나타낸 것이다. 프리페칭을 수행하지 않았을 경우 캐시 바이트 적중률은 31.4%였고, 참조 횟수가 1 회 이상인 웹 객체를 대상으로 프리페칭을 수행했을 경우 증가된 캐시 객체 적중률은 5.01%, 참조 횟수가 2 회 이상인 경우의 증가율은 3.78 %, 3 회 이상인 경우의 증가율은 2.27%, 4 회 이상인 경우의 증가율은 1.59%, 참조 횟수가 5 회 이상인 경우의 증가율은 1.19%였다. 참조 횟수를 1 회 이상으로 프리페칭을 수행했을 경우에는 참조 횟수를 2 회 이상으로 수행했을 경우보다 1.23% 증가하였다.

표 10. 프리페칭을 수행했을 경우와 수행하지 않았을 경우의 캐시 바이트 적중률

캐시 바이트 적중률 (캐싱)	31.4 %
캐시 바이트 적중률 (캐싱+프리페칭) (참조 횟수 1회 이상)	36.4 %
캐시 바이트 적중률 (캐싱+프리페칭) (참조 횟수 2회 이상)	35.2 %
캐시 바이트 적중률 (캐싱+프리페칭) (참조 횟수 3회 이상)	33.7 %
캐시 바이트 적중률 (캐싱+프리페칭) (참조 횟수 4회 이상)	33.0 %
캐시 바이트 적중률 (캐싱+프리페칭) (참조 횟수 5회 이상)	32.6 %

그림 26 은 객체 참조 횟수별로 프리페칭을 수행했을 경우의 캐시 객체 적중률과 프리페칭을 수행하지 않았을 경우의 캐시 객체 적중률을 그래프로 나타낸 것이다. Request saving 과 비슷한 패턴을 보이고 있다.

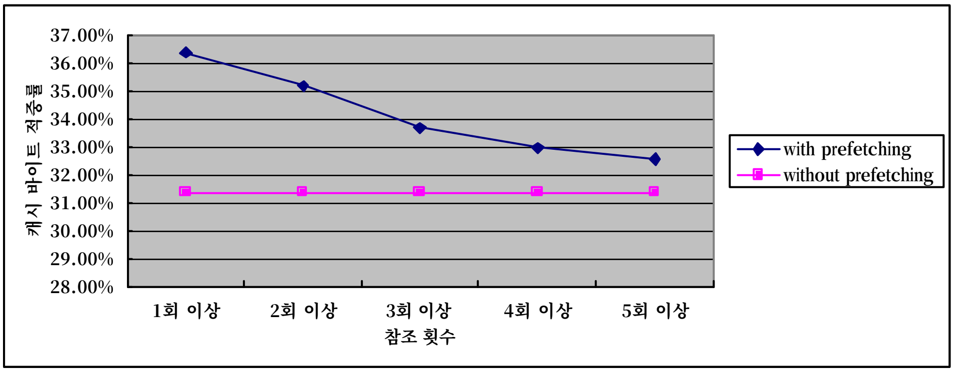

그림 26. 참조 횟수별로 프리페칭을 수행했을 경우와 프리페칭을 수행하지 않았을 경우의 캐시 바이트 적중률

본 논문에서 제시한 프리페칭 알고리즘을 사용하면 프리페칭에 의해서 절약되는 대역폭은 실제로는 peak periods 동안에 사용될 대역폭을 줄여준다. 표 11 은 프리페칭을 통해서 peak periods 동안에 절약되는 대역폭 감소율을 나타낸 것이며 그림 27 은 그래프로 표현한 것이다.

표 11. 참조 횟수별 Peak Periods 동안의 대역폭 감소율

참조 횟수	Peak periods 동안의 대역폭 사용량 감소율
1 회 이상	6.4 %
2 회 이상	4.8 %
3 회 이상	2.9 %
4 회 이상	2.0 %
5 회 이상	1.5 %

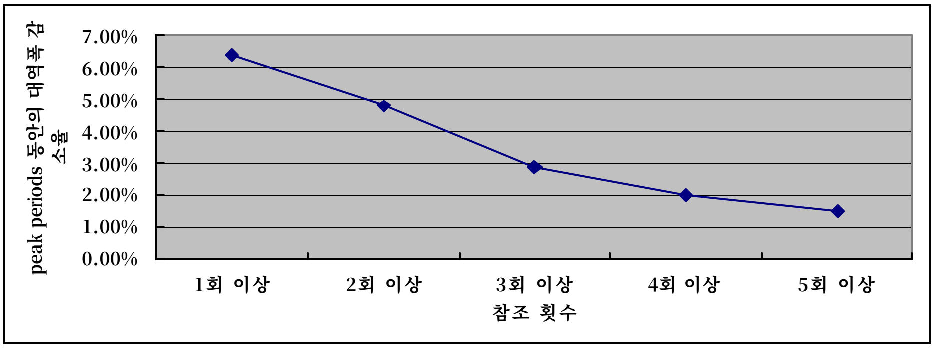

그림 27. 참조 횟수별 하루 평균 Peak Periods 동안의 대역폭 사용량 감소율

6.5 Summary

본 논문에서 제시한 방법으로 프리페칭했을 경우의 성능 평가 척도를 참조 횟수별로 정리하면 표 12와 같다.

표 12. Performance Mateics

참조 횟수	Request saving	Bandwidth saving	Wasted bandwidth	Accuracy	
				Prefetched object hit rate	Prefetched byte hit rate
1 회이상	4.30 %	5.01 %	17.8 %	39.4 %	22 %
2 회이상	3.53 %	3.78 %	4.6 %	64.3 %	45.1%
3 회이상	2.72 %	2.27 %	2.0 %	76.5 %	53.5 %
4 회이상	2.10 %	1.59 %	1.0 %	83.3 %	62.6 %
5 회이상	1.62 %	1.19 %	0.5 %	87.1 %	72.6 %

본 논문에서 제안한 프리페칭 알고리즘으로 프리페칭을 수행했을 경우 off-peak periods 동안의 대역폭은 증가되고 peak periods 동안의 대역폭

은 감소된다. 참조 횟수별로 off-peak periods 동안에 증가되는 하루 평균 대역폭 증가율과 peak periods 동안 감소되는 하루 평균 대역폭 감소율을 도표로 나타내면 표 13과 같다.

표 13. 참조 횟수별 Peak Periods 동안의 대역폭 감소율과 Off-peak Periods 동안의 대역폭 증가율

참조 횟수	Peak periods 대역폭 감소율	off-peak periods 대역폭 증가율
1회 이상	6.4 %	51.9 %
2회 이상	4.8 %	28.3 %
3회 이상	2.9 %	16.7 %
4회 이상	2.0 %	10.8 %
5회 이상	1.5 %	7.2 %

지금까지 참조 횟수 별로 프리페칭의 성능 평가 척도에 따라 살펴 보았다. 프리페칭의 효율성을 높이기 위해서는 프리페칭에 의해서 절약되는 대역폭량은 많을수록 좋으며 프리페칭에 의해서 낭비되는 대역폭량은 적을수록 좋다. 따라서 프리페칭의 효율성은 아래와 같은 수식으로 나타낼 수 있다.

$$E_p ? B_s / B_w$$

E_p :프리페칭 효율성
B_s :프리페칭에 의해서 절약되는 대역폭량(*Mbyte*)
B_w :프리페칭에 의해서 낭비되는 대역폭량(*Mbyte*)

본 논문에서는 참조 횟수별로 웹 객체를 구분하여 1회 이상, 2회 이상, 3회 이상, 4회 이상, 5회 이상으로 프리페칭을 시뮬레이션 하였다. 이들 다섯 가지 경우 중에서 가장 프리페칭의 효율성이 높은 것을 선택하여 프리페칭을 수행해야 하는데, 참조 횟수가 높을수록 프리페칭의

효율성은 높아질 것이다. 그러나 여기서 고려해야 될 사항은 프리페칭을 통해서 증가되는 peak periods 동안의 절약되는 대역폭량이다. 참조 횟수가 높아지면 프리페칭의 효율성도 높아지나 프리페칭하는 웹 객체 수가 적어지므로 peak periods 동안의 절약되는 대역폭량은 줄어들게 된다. 본 논문에서는 peak periods 동안의 절약되는 대역폭량이 peak periods 동안 사용된 대역폭량의 4% 이상으로 보장하고자 한다. 참조 횟수별 프리페칭의 효율성과 peak periods 동안의 대역폭 감소율을 표로 나타내면 표 14와 같다.

표 14. 참조 횟수별 프리페칭 효율성과 Peak Periods 대역폭 감소율

참조 횟수	B_s (Mbyte)	B_w (Mbyte)	E_p (Mbyte)	Peak periods 대역폭 감소율
1회 이상	189.16	670.72	0.28	6.4 %
2회 이상	141.82	172.65	0.82	4.8 %
3회 이상	85.65	74.47	1.15	2.9 %
4회 이상	60.08	35.91	1.67	2.0 %
5회 이상	45.06	16.97	2.66	1.5 %

표 14를 보면 peak periods 대역폭 감소율이 4%이상인 경우는 참조회수를 1회 이상, 2회 이상으로 프리페칭을 수행했을 경우다. 또한 참조 횟수를 2회 이상으로 프리페칭을 수행했을 경우의 프리페칭 효율성은 0.82였지만 참조 횟수를 1회 이상으로 수행했을 경우의 프리페칭 효율성은 0.28로 참조 횟수 2회 이상으로 프리페칭을 수행했을 경우가 효율성이 더 높게 나왔다. 따라서 최소한 peak periods 대역폭 감소율을 4% 이상으로 보장하고 프리페칭의 효율성이 높이기 위해서 참조 횟수를 2회 이상으로 프리페칭을 수행하도록 결정한다. 그러나 이것은 포항공대 기숙사의 환경에 따른 결정이며 환경이 바뀌면 결정 사항도 변경될 수 있다.

참조 횟수가 2 회 이상일 경우 프리페칭에 의해 감소되는 웹 객체의 하루 평균 바이트 수는 141,824,895 bytes 이다. peak periods 는 54,000 seconds (19 hours)이므로 프리페칭에 의해 peak periods 동안에 감소되는 대역폭 감소량은 다음과 같다.

peak periods 동안의 대역폭 감소량 = (141824895*8)/54000 = 20.52 Kbps

따라서 참조 횟수를 2 회 이상으로 프리페칭할 웹 객체의 대상을 제한하여 프리페칭했을 경우 peak periods 동안에 감소된 평균 대역폭 감소량은 20.52 Kbps 였다.

7. 결론 및 향후 과제

본 논문은 peak periods 동안의 대역폭 사용량을 감소시키기 위한 목적으로 웹 트래픽을 대상으로 하는 프리페칭 알고리즘을 제안하고, 시뮬레이션을 통해 그 유효성을 보였다. 시뮬레이션의 결과는 본 논문에서 제안한 프리페칭 알고리즘으로 off-peak periods 동안의 사용되지 않는 대역폭을 사용하여 미리 예상되는 웹 객체를 읽어옴으로써 peak periods 동안의 대역폭 사용량을 줄일 수 있다는 것을 보여주었다.

본 논문에서 제안한 프리페칭 알고리즘을 통해 request saving 은 캐싱 서버만을 사용했을 경우보다 하루 평균 3.53%가 증가되었고 bandwidth saving 은 3.78%가 증가되었다. 또한 참조 횟수를 2회 이상인 웹 객체만을 대상으로 프리페칭을 했기 때문에 프리페칭 객체 적중률과 프리페칭 바이트 적중률은 각각 64.26%, 45.1%로 매우 높았다. 이것은 적은 양의 웹 객체를 프리페칭하여도 프리페칭된 웹 객체에 대한 적중률을 높임으로써 프리페칭의 효율성을 높일 수 있었고 참조 횟수의 조절을 통해 프리페칭에 요구되는 디스크 용량을 조절할 수 있다는 장점이 되었다. 프리페칭에 의해 감소된 peak bandwidth usage 는 20.52 Kbps 였다. 따라서 캐싱 서버에 본 논문에서 제시한 프리페칭 기법을 사용함으로써 낭비되는 대역폭이 거의 없이 peak periods 동안의 대역폭 사용량을 줄일 수 있었다.

prefetching scheme 은 squid 라는 캐싱 서버에 독립적으로 add-on feature 로써 추가되었다. 따라서 웹 서버나 클라이언트의 브라우저의 변경이 필요 없이 웹 서버나 클라이언트에 투명하게 프리페칭을 수행할 수 있다.

시뮬레이션 결과를 통해 본 논문에서 제안한 프리페칭 알고리즘의 유효성을 확인할 수 있었다. 따라서 이와 같은 프리페칭 시스템을 캐싱

서버에 추가하면 새로운 네트워크 회선을 증가시킬 필요없이 효율적으로 사용될 수 있다.

본 논문에서 수행된 Web traffic trace 나 시뮬레이션에서 사용된 웹의 환경은 일반적인 환경이 아닌 학교 기숙사라는 특수한 환경이었다. 따라서 off-peak periods (e.g., 20:00~08:00)가 더 큰 회사와 같은 일반 산업 환경에서는 프리페칭의 성능이 비슷하거나 더 좋아질 것이라고 기대된다. 본 논문에서는 시뮬레이션만을 통해서 프리페칭 알고리즘의 유효성을 측정하였는데, 실제로 캐싱 서버와 함께 운영함으로써 유효성을 입증하는 것이 필요하다.

현재의 프리페칭 시스템은 로그를 한꺼번에 읽어와서 프리페칭이 가능한 웹 객체를 선택하는데, real-time 으로 prefetchable object list 를 생성할 수 있도록 확장시킬 필요성이 있다.

참고 문헌

[1] Z. Wang and J. Crowcroft, "Prefetching in World Wide Web," IEEE Globecom 96, http://www.cs.ucl.ac.uk/staff/zwang/papers/prefetch.ps.z.

[2] V. Padmanabhan and J. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," Computer Communication Review, 26(3):22-36, July 1996.

[3] Ken-ichi Chinen and Suguru Yanaguchi, "An Interactive Prefetching Proxy Server for Improvement of WWW Latency," INET'97, 1997, http://www.isco.org/INET97/procceding/A1/A1_3.HTM.

[4] Arthur Goldberg, Ilya Pevzner and Robert Buff, "Caching Characteristic of Internet and Intranet Web proxy Traces," In Computer Measurement Group Conference (CMG'98), Anaheim, CA, December 1998, http://www.cs.nyu.edu/artg.

[5] David Barnes and Neil G. Smith, "An Analysis of World-Wide Web Proxy Cache Performance and its Application to the Modelling and Simulation of Network Traffic," In Proceedings of the Fourth International Conference on Telecommunication Systems Modeling and Analysis, March 1996, http://www.cs.ukc.ac.uk/people/staff/djb/pubs.html.

[6] Themistoklis Palpanas and Alberto Mendelzon, " Web Prefetching Using Partial Match Prediction," In Web Caching Workshop WCW'99, 1999, http://www.ircache.net/Cache/Workshop99/program.html.

[7] Squid Internet Object Cache, available from http://squid.nlanr.net/Squid/.

[8] Gihan V.Dias, Graham Cope and Ravi Wijayaratne, "A Smart Internet Caching System, " INET'96 Conference, 1996, http://www.isoc.org/isoc/whatis/conferences/inet/96/proceedings/a4/a4_3.htm.

[9] Katsuo Doi, "WWW Access by Proactively Controlled Caching Proxy, " Sharp Technical Journal, No. 66, December 1996.

[10] Brad Duska, David Marwood, and Michael J.Feeley, "The Measured Access Characteristics of World-Wide Web Client Proxy Caches," In Usenix Symposium on Internet Technologies and Systems (USITS), Monterey, CA, USA, December 8-11 1997, Usenix, http://www.cs.ubc.ca/spider/marwood/Projects/SPA/Report/Report.html.

[11] Marc Abrams, C.R.Standridge, G.Abdulla, S.Williams, and E.A.Fox, "Caching Proxies: Limitations and Potentials," In Proceedings of the Fourth International WWW Conference, 1995, http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html.

[12] Anawat Chankhunthod et al, "A Hierarchical Internet Object Cache," Technical Conference, Usenix 1996, http://excalibur.usc.edu/cache-html/cache.html.

[13] James Griffioen and Randy Appleton. "Reducing File System Latency using a Predictive Approach," Proceddings of the 1994 Summer USENIX Technical Conference, Boston, Massachusetts, USA, 1994, http://usenix.org/publications/library/proceedings/bos94/griffioen.html.

[14] Azer Bestavros, "Speculative Data Dissemination and Service to Reduce Server Load," Network Traffic and Service Tome in Distributed Information System, In International Conference on Data Engineering, pages 180-189, New Orleans, LO, February 1996.

[15] Tomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. "Exploring the bounds of web latency reduction from caching and prefetching," In Proceedings of USENIX Symposium on Internet Technology and Systems, December 1997, http://www.usenix.org/publications/library/proceedings/usits97/kroeger.html.

[16] Evangelos P. Margatos and Catherine E. Chronaki, "A top-10 Approach to Prefetching on the Web," Technical report, In Proceedings of INET' 98 (The Internet Summit), Geneva, Switzerland, July 1998, http://www.ics.forth.gr/proj/arch-vlsi/OS/www.html.

[17] Wcol Group, "WWW Collector – the prefetching proxy server for WWW," 1997, http://shika.aist-nara.ac.jp/products/wcol/wcol.html.

[18] Li Fan, Quinn Jacobson, Pei Cao, and Wei Lin, "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance", In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99), Atlanta, GA, May 1999, http://www.cs.wisc.edu/~cao/.

[19] Wget, available from http://subzero.campus.luth.se/FreeDocs/wget-1.4.2/wget_toc.html.

[20] Tim Bray, "Measuring the Web," In Proceedings of the Fifth International

World Wide Web Conference, pages 993-1005, Paris, France, May 1996.

[21] Allison Woodruff, Paul M. Aoki, Eric Brewer, Paul Gauthier, and Lawrence A. Rowe, "An Investigation of Documents from the WWW", In Proceedings of the Fifth International WWW Conference, pages 963-979, Paris, France, May 1996.

[22] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0," RFC 1945, May, 1996.

[23] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1," RFC 2616, June 1999.

[24] Evangelos Markatos , Catherine E. Chronaki, "A Top-10 Approach to Prefetching on the Web," Technical Report No. 173, ICS-FORTH, Heraklion, Crete, Greece ,August 1996.

[25] Ghaleb Abdulla, Edward A. Fox, Marc Abrams, and Stephen Williams, "WWW Proxy Traffic Characterization with Application to Caching," Technical Report TR-97-03, Computer Science Department, Virginia Tech, March 1997, http://www.cs.vt.edu/~chitra/work.html.

[26] James E. Pitkow, "Summary of WWW characterizations," In Proceedings of the Seventh International World Wide Web Conference, Brisbane, Australia, April 1998, http://www7.scu.edu.au/programme/fullpapers/1877/com1877.htm.

[27] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li, "A Study of Integrated Prefetching and Caching Strategies," In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1995, http://www.cs.wisc.edu/~cao/publications.html.

[28] Eric A. Brewer, Paul Gauthier, and Dennis McEvoy, "The long-term viability of large-scale caching," In Proceedings of the Third International WWW Caching Workshop, Manchester, England, June 1998, http://wwwcache.ja.net/events/workshop.

[29] CacheRaQ of Cobalt Network, available from http://www.coblatnet.com.

[30] Webspeed of Packetstorm Technologies, available from http://www.packetstorm.on.ca/products/webspeed/featuresindetail.html.

[31] Bert Williams, "Transparent web caching solutions," In Proceedings of the Third International WWW Caching Workshop, Manchester, England, June

1998. http://wwwcache.ja.net/events/workshop/33/cachepaper.html.

[32] David H. Crocker, “Standard For The Format Of Arpa Internet Text Message,” RFC 822, August 13, 1982.

[33] **김용운**, “Protocals for the World-Wide Web”, available form http://pec.etri.re.kr/~qkim/HTTP/C11.html.