효율적인 인터넷 대역폭 사용을 위한 웹 프리페칭 기법

(Web Prefetching Scheme for Efficient Internet Bandwidth Usage)

김 숙 향 † 홍 원 기 ††
(Sook-Hyang Kim) (Won-Ki Hong)

요 약 World Wide Web(이하 웹)은 저렴한 가격과 다양하고 흥미 있는 정보를 쉽고 간편하게 찾아볼 수 있다는 장점으로 웹의 사용자는 하루가 다르게 증가되고 있으며 웹의 사용자의 증가와 함께 웹을 통해 전달되는 데이타 즉, 웹 문서, 그림, 멀티미디어 데이타 등의 크기 또한 빠르게 증가되고 있다. 웹 트래픽을 위한 네트워크 대역폭의 사용량을 살펴보면 사용자들의 요청이 많은 peak periods에는 대부분의 대역폭을 사용하고 있고, off-peak periods에는 사용하지 않는 대역폭이 존재한다. 지금까지 네트워크의 대역폭 소비량을 감소시키고, 검색 지연시간을 줄이기 위해 많은 연구가 이루어졌고 그 해결방안 중 하나가 웹 캐싱이다. 그러나, 웹 캐싱을 사용하더라도 peak periods 동안에는 네트워크 대역폭의 사용량을 감소시키기에는 한계가 있으며 off-peak periods에 여유 있는 네트워크 대역폭을 효율적으로 사용할 수 없다. 본 논문에서는 네트워크 대역폭을 균형 있게 사용하기 위해 캐싱 서버(SQUID)를 기반으로 하는 웹 프리페칭(Web prefetching) 기법을 제안한다. 24시간 동안의 웹 사용 상황을 분석하여, 가장 많이 사용되면서 다음 24시간 내에 유효기간을 초과하는 웹 객체를 프리페칭하는 방법을 사용한다. 제안된 웹 프리페칭 기법은 peak periods 동안 요청되리라 예상되는 웹 객체를 off-peak periods 동안 남는 대역폭을 이용하여 미리 캐싱 서버에 가져다 놓는 방법으로써 많은 디스크 용량을 요구하지 않으면서도 캐시 객체 히트율을 높일 수 있다. 또한 사용자들의 웹 접근 패턴을 기반으로 하기 때문에 프리페칭된 웹 객체에 대하여 높은 객체 히트율을 보인다. 본 논문에서 소개한 기법의 유효성 시뮬레이션을 통해서 증명하였다. 시뮬레이션 결과는 본 논문에서 제안된 프리페칭 기법이 효율적으로 peak bandwidth를 줄일 수 있다는 것을 나타낸다.

Abstract 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 paper, 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 scheme that improves cache hit rate while only requiring a small amount of storage. Web objects that were accessed many times in previous 24 hours but would be expired in the next 24 hours, are selected and prefetched in our scheme. We present simulation results based on Web proxy trace and show that this prefetching algorithm can reduce peak time bandwidth using off-peak bandwidth.

† 정 회 원 : 포항공과대학교 정보통신대학원 정보통신학과
shk@postech.ac.kr
†† 정 회 원 : 포항공과대학교 컴퓨터공학과 교수
jwkhong@postech.ac.kr
논문접수 : 2000년 1월 31일
심사완료 : 2000년 7월 5일

1. 서 론

World Wide Web (이하 웹)은 다양하고 흥미 있는 정보를 쉽고 간편하게 찾아볼 수 있다는 장점으로 하루가 다르게 사용자가 증가하고 있다. 웹 사용자의 증가와 함께 웹을 통해 전달되는 데이타 즉, 웹 문서, 그림, 멀

티미디어 데이타 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 데이타를 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 periods동안에 대부분의 대역폭을 사용하고 있고, off-peak periods동안에는 사용하지 않는 대역폭이 존재한다. 그림 1은 포항공대의 기숙사 중에서 16개의 서브넷에 대한 2주 동안 (1999년 10월 15일부터 1999년 10월 28일까지)의 네트워크 트래픽의 하루 평균 대역폭 사용량을 시간대별로 나타낸 것이다.

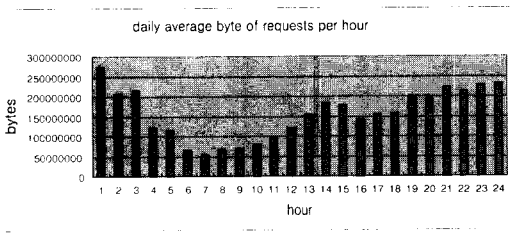

그림 1 포항공대 기숙사의 시간대별 평균 웹 트래픽량

그림 1에 나타나 있듯이 peak periods동안에는 대부분의 네트워크 대역폭이 사용되며 off-peak periods동안에는 사용되지 않는 대역폭이 발생되는 것을 알 수 있다. 따라서 peak periods와 off-peak periods간의 좀 더 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 [7, 11, 12, 27, 28]. 클라이언트들의 요구를 기반으로 하는 웹 캐싱은 peak peiods와 off-peak periods의 구별 없이 대역폭 사용량을 감소시킨다. 따라서, 웹 캐싱을 사용하더라도 off-peak periods동안에 사용되지 않는 네트워크 대역폭을 효율적으로 사용하여 peak periods동안의 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 periods동안 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 데이타를 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 periods동안의 대역폭 사용량을 줄이는데 효율적일 것이다.

본 논문의 2장에서는 본 논문과 관련하여 웹에서의 프리페칭 기법에 대하여 연구한 여러 논문들을 기반으로 하여 웹 프리페칭 기법에 대하여 살펴보며 3장에서는 본 논문에서 제안하는 웹 프리페칭 기법에 대한 소개와 기존의 프리페칭 방법의 문제점을 소개하고 해결방안을 제시한다. 4장에서는 본 논문에서 제시하는 프리페칭 시스템의 설계에 대해 설명하고 5장에서는 시뮬레이션 방법을, 6장에서는 시뮬레이션의 결과를 보여준다. 끝으로 7장에서는 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 프리페칭을 사용하였다. 프리페칭에 관련된 논문들을 기반으로 하여 웹 프리페칭의 기법은 크게 세가지 측면으로 분류될 수 있다. 첫번째는 프리페칭을 수행하는 위치에 따른 분류이고, 두번째는 프리페칭될 웹 문서의 선정 방법에 따라 분류할 수 있으며, 세번째로는 프리페칭하는 목적에 따라 분류할 수 있다.

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 하이퍼링크와 같이 다음에 따라올 가장 적당한 요청을 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 서

버에 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 prefetching은 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 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 네트웍 자원이 적다는 장점이 있으나 관리자나 사용자가 프리페칭될 웹 객체를 하나하나 지정해야 하기 때문에 전체 웹 객체를 대상으로 하기에는 한계가 있다는 단점이 있다.

2.3 목적에 따른 분류

프리페칭은 목적에 따라 응답 시간 (response time)을 줄이기 위한 프리페칭과 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 periods와 off-peak periods간의 불균형적인 대역폭 사용을 프리페칭을 통해 해결하고자 하는 것이다. Batch prefetching 기법이 균형잡힌 balanced bandwidth usage를 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 알고리즘

위의 프리페칭 방법들 중에서 server-initiated prefetching을 위해서 클라이언트는 웹 문서가 정확하게 프리로드될 수 있도록 허가되어 있어야 하고 웹 서버 소프트웨어의 변경을 요구하기 때문에 모든 웹 서버로 확장하는데 어렵다는 단점이 있다. Client-initiated Prefetching은 단 하나의 클라이언트의 웹 문서 접근 패턴을 이용하기 때문에 인터넷의 전체적인 사용자들의 접근 패턴을 공유할 수 없다는 단점이 있다. 웹 문서 선정 방법에 따른 분류에서 결정적 방법은 관리자나 사용자가 프리페칭될 웹 문서를 하나하나 지정해야 하기 때

문에 전체 웹 문서를 대상으로 하기에는 한계가 있다. 본 논문은 균형적으로 네트워크 대역폭을 사용하고 peak bandwidth usage를 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 실제 사용되는 트래픽 패턴을 분석하기 위하여 포항공대 기숙사 지역 16개의 서브넷에서 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 파라미터의 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 프리페칭이 수행될 시간이다.

• 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.

• 참조 횟수에 대해서는 traditional prefetching scheme의 단점인 과다한 트래픽의 발생을 줄이고 프리페칭될 웹 문서에 대한 적중률을 높이기 위해서는 참조 횟수가 n회 이상인 웹 객체를 대상으로 제한한다. 이것은 다음에 요청될 가능성이 높은 객체들만을 프리페칭함으로써 적은 수의 웹 객체를 프리페칭하여도 높은 프리페칭 성능을 나타내기 위함이다.

• 프리페칭될 웹 문서의 전체 크기는 참조 횟수와 연관 지어서 결정할 수 있다. 예를 들어 프리페칭할 웹 문서의 전체 용량을 100M로 제한한다면 참조 횟수가 높은 웹 객체들부터 선택되도록 한다.

• 프리페칭이 수행될 시간도 결정되어야 한다. 본 논문은 off-peak periods의 남는 대역폭을 이용하여 peak periods와 off-peak periods간의 균형있는 대역폭 사용을 위한 프리페칭이다. 따라서 프리페칭이 적용될 시간은 웹 트래픽이 적은 off-peak periods 동안으로 설정한다. 본 논문에서는 프리페칭이 수행되면 off peak periods동안에 많은 웹 트래픽이 발생되므로 off-peak periods를 전체 웹 트래픽량의 평균을 기준으로 하여 평균 80% 미만 (125299861 byte)인 시간대를 off-peak periods로 결정한다. 따라서 off-peak periods는 새벽 04:00부터 13:00까지며, peak-periods는 13:00부터 04:00까지다. off-peak periods를 결정하는 기준은 각각의 인트라넷 구성 조직의 특성에 따라 변경이 가능하다.

3.4 포항공대 기숙사의 웹 트래픽 추적(Web Traffic

Trace)

웹 접근 패턴을 분석하고 프리페칭 파라미터들의 값을 결정하기 위하여 실제 트래픽 패턴을 조사하였는데 포항공대 기숙사 지역 16개의 서브넷에서 발생되는 웹 트래픽을 대상으로 하였다. 포항공대의 웹 트래픽 추적 기간은 1999년 10월 15일(금)부터 10월 28(목)일까지 2주 동안이다. 2주 동안 웹 객체를 요청한 클라이언트들의 수는 447대였다. 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 4,077,308개였고 total bytes는 50.2G 였다. 또한 하루에 캐싱 서버로 들어오는 평균 요청의 개수는 291236개 정도였고 하루 평균 bytes는 3.6G였다.

표 1 참조 횟수에 따른 웹 객체 수

참조 횟수	하루 평균 웹 객체수(%)	웹 객체의 하루 평균 총량(Mbyte) (%)
1 회	86,227 (70.75 %)	1479.75 (68.12 %)
2 회	16,690 (13.69 %)	310.79 (14.31 %)
3 회	6,462 (5.3 %)	118.20 (5.44 %)
4 회	3,451 (2.83 %)	74.01 (3.41 %)
5회 이상	9,053 (7.43 %)	189.52 (8.72 %)
Total	121,883 (100 %)	2172.27 (100 %)

표 1은 참조 횟수 (reference count)에 따른 캐시에 저장된 웹 객체의 객체수와 웹 객체의 평균 총량을 나타낸 것이다. 참조 횟수는 클라이언트들이 웹 객체별로 요청한 횟수를 나타낸다. 캐시에 저장된 웹 객체들 중에서 단지 한번만 참조된 웹 객체들은 전체 웹 객체 (121,883개) 중에서 70.75%를 나타내었고 2회 이상 참조된 웹 객체들은 단지 29.25%에 불과하였다. 웹 객체의 total size는 참조 횟수가 1회인 것이 68.12%를 차지하였고, 2회 이상인 것이 전체의 31.88%를 차지하였다. 또한 하루동안 요청된 웹 객체들의 total bytes는 2172.27 Mbyte였다. 웹 객체의 평균 총량도 웹 객체수와 비슷한 패턴을 보이고 있다는 것을 알 수 있다.

3.5 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 hit이 발생된 요청 횟수를 사용자의 전체 요청 수의 비율로 나타낸다. Request saving이 높을수록 캐시 객체 적중률은 높아진다.

Bandwidth Saving : 프리페칭에 의해 절약된 대역폭을 사용자의 전체 웹 트래픽에 대한 대역폭 사용량의 비율로 나타낸다. Bandwidth saving이 높아질수록 캐시 바이트 적중률은 높이지고 그만큼 peak periods의 bandwidth usage는 줄어들게 된다. 또한 프리페칭에 의해서 절약되는 대역폭은 실제로는 peak periods동안에 사용될 대역폭을 줄여주기 때문에 프리페칭을 통해서 peak periods동안에 절약되는 대역폭 감소율을 측정한다.

Accuracy : accuracy는 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 bytes의 비율을 나타낸 것이다. Accuracy가 높아질수록 프리페칭에 의해 낭비되는 대역폭은 적어진다.

Wasted Bandwidth : 프리페칭 되었지만 클라이언트에 의해서 요청되지 않은 웹 객체의 전체 바이트수로 나타낸다. 본 논문은 off-peak periods동안의 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 2와 같이 나타낼 수 있다.

프리페칭 시스템 설계에 들어가기 앞서 설계 시 필요

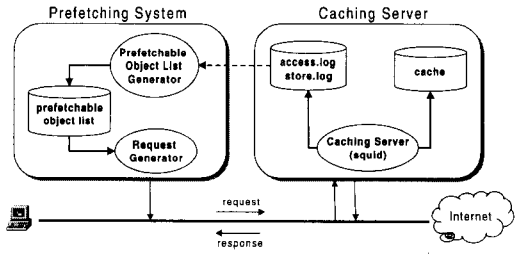

그림 2 프리페칭 시스템 구조

한 몇 가지 가정들이 있다. 첫번째로 본 논문에서 제안하는 프리페칭 시스템은 캐싱 서버와의 연동이 필요한데, 캐싱 서버로는 Squid [7]를 사용하였다. Squid를 캐싱 서버로 사용한 이유는 Squid는 freeware이기 때문에 학교, 회사등에서 많이 사용되고 있으며 상용 캐싱 서버 제품으로 Cobalt Network의 CacheRaQ [29]와 Packetstorm Technologies의 Webspeed [30]도 내부적으로 Squid를 사용하고 있다. 따라서 Squid를 기반으로 프리페칭 시스템을 설계하면 Squid를 사용하고 있는 곳은 어디에서나 본 논문에서 제안하는 프리페칭 시스템을 사용할 수 있다. 두번째로 캐싱 서버는 transparent 캐싱 서버로서 동작한다.

전반적인 프리페칭 시스템의 모듈은 크게 Prefetchable Object List Generator와 Request Generator 두 가지로 나눌 수 있다. 클라이언트들이 캐싱 서버(Squid)에게 요청을 보내면 Squid는 먼저 자신의 캐시에 요청받은 웹 객체가 존재하는지 확인하고, 있으면 그 웹 객체를 클라이언트에게 제공하고 존재하지 않거나 유효하지 않을 경우에는 해당 웹 서버에게 요청을 보낸다. 이 시스템은 squid에서 작성된 이러한 로그(access.log, store.log)를 이용하여 Prefetchable Object List Generator에서 prefetchable object list를 생성한다. Request Generator는 하나의 클라이언트로써 동작하여 off-peak periods동안에 prefetchable object list에 있는 요청 (request)를 웹 서버에게 보내어 Squid의 캐시에 저장되어 있는 웹 객체들을 갱신 시킨다. 이 프리페칭 시스템은 Squid와는 독립적으로 add-on으로서 동작한다.

4.1 Prefetchable Object List Generator

Prefetchable Object List Generator는 프리페칭될 웹 객체를 선택하기 위한 모듈이다. 이것은 캐시에 저장된 각각의 웹 객체에 대한 참조 횟수와 웹 객체의 크기를 알기 위해 Squid에 의해서 생산되는 access log와 store log를 이용한다. access log는 Squid에게 요청되는 requests에 대한 정보가 나타나 있고 store log는 캐시에 저장된 웹 객체에 대한 정보가 표현되어 있다. 먼저 Prefetch Object List Generator는 Squid의 재생 알고리즘 (refresh algorithm)을 이용하여 유효기간이 지난 웹 객체 중에서 앞장에서 설명된 조건에 따라 프리페칭이 가능한 prefetchable object를 선택한다. HTTP response message의 헤더 부분에 유효기간이 나타나 있지 않는 웹 객체는 Squid에서 기본적으로 설정해주는 값 (3일)으로 정한다 [7]. 수집한 정보는 그림 3의 형식에 맞게 분석하여 prefetchable object list를 생성한다.

Reference Count	URL	Byte

그림 3 Prefetchable Object List의 형식

Prefetchable Object List Generator는 웹 객체들의 유효기간을 확인하기 위하여 Squid의 재생 알고리즘을 응용하여 사용하는데, 재생 알고리즘에 필요한 매개 변수들은 store log에 나타나 있다. store log는 해당 웹 객체가 캐시에 저장될 때 HTTP response header에 들어있는 각 필드들 중에서 필요한 항목만 추출하여 저장한다.

다음은 재생 알고리즘에서 사용하는 HTTP response header의 형식과 필드에 대하여 알아보고 Squid에서 제공되는 재생 알고리즘과 각각의 웹 객체들의 유효성을 결정하는 방법에 대해서 설명한다.

4.1.1 HTTP/1.0 Response Header

HTTP 프로토콜 [22, 23]은 요구/응답 (Request/Response) 방식을 이용하여 동작하고 있다. 즉, 원하는 프로토콜 기능(예: GET, HEAD, POST)에 대해 서비스 요구를 하면 데이타 송수신을 위한 TCP 연결이 만들어지고, 서버가 응답을 보내어 데이타 전송을 끝내면 자동적으로 연결이 끊어지게 되는 것이다. 클라이언트는 서버와의 사이에 TCP를 기반으로 한 HTTP 연결을 만들고 method, URI, protocol version, 클라이언트 정보, 사용자 데이타 등과 같이 규정되어 있는 요구 형식에 따라 서버에게 요구 메시지를 보낸다. 서버는 프로토콜 버전, 성공 또는 오류코드 번호, 서버 정보, 데이타 정보, 사용자 데이타 등을 포함하는 응답 메시지를 보낸다. 캐싱 서버에서 사용되는 HTTP response header의 필드들을 설명하기 전에 먼저 HTTP/1.0 response header의 형식에 대해서 살펴보면 그림 4와 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 4 HTTP/1.0 Response Header

Full-Response 메시지에 들어가는 첫 번째 줄의 내용이 Status-Line이며 구성 형식은 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 CRLF는 마지막 이외의 장소에는 허용되지 않는다. 다만 사용자 데이타와 같은 Entity 부분은 CRLF에 뒤이어 나타날 수 있다.

응답 메시지(response message)의 형식을 보면, General-Header, Request-Header, Response-Header, Entity-Header 등과 같이 네 가지 헤더 형식이 있다. 여기에서 공통적으로 사용하는 표현 형식은 RFC 822의 3.1절에서 정의하고 있는 것이다 [31].

General Header는 Full-Request 메시지와 Full-Response 메시지에 공통적으로 포함되어 있는 헤더 형식이다. 이것은 전송되고 있는 메시지에 관한 사항을 알리는 것이다. General Header는 Date와 Progma로 구성되어 있다. Date는 메시지가 만들어지는 날짜와 시간을 나타낼 때 쓰이며 Progma는 요구/응답의 연쇄 동작에 따라 어느 수신측에 적용되는 변수로서 구현에 관련된 것들을 포함하는 데 쓰인다.

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

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

Squid의 재생 알고리즘에서 사용되는 HTTP response header의 필드에 대해서 좀더 살펴보면 다음과 같다.

• Date : General Header의 Date는 메시지가 만들어지는 날짜와 시간을 나타낼 때 쓰이며, RFC 822 [32]에서 정의하고 있는 orig-date와 같은 의미이다. 이의 예는 다음과 같다.

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

• Expires : Entity Header의 Expires 필드는 전달하는 데이타를 의미 없는 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 캐시되어 있는 데이타에 대해 이렇게 표시되어 있을 때라면 지정한 시각이 지나고 난 다음에는 캐시되어 있는 데이타를 지워도 되는 것이다. Expires의 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 Squid의 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 Squid에서는 재생 알고리즘을 제공하고 있다. 그림 5는 Squid에서 캐시에 저장된 객체를 언제 재생할 것인가를 결정하는 재생 알고리즘을 순서도로 표현한 것이다.

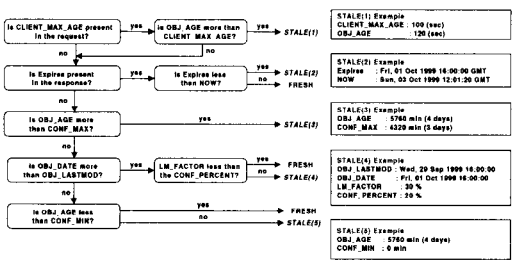

그림 5 재생 알고리즘의 순서도

조건식 중에서 최초로 일치하는 엔트리를 사용하며 만일 일치하는 엔트리가 없다면 default value를 사용한다. OBJ_DATE는 웹 서버에 의해서 응답 메시지가 만들어진 시간을 나타낸다. OBJ_LASTMOD는 그 객체가 마지막으로 갱신된 시간을 나타내며 Expires는 웹 서버에서 정해주는 웹 객체에 대한 유효기간을 나타낸다. 각각의 OBJ_DATE, OBJ_LASTMOD, Expires는

HTTP Respons Header로부터 가져올 수 있다 [22]. CLIENT_MAX_AGE는 클라이언트가 개별적으로 정해주는 웹 객체의 최대 객체 나이이며 HTTP/1.1 Cache-Control Request Header로부터 알 수 있다[23]. 또한 Squid는 refresh_pattern 규칙을 통해 CONF_MIN, CONF_PERCENT, CONF_MAX의 값을 squid.conf에서 기본적으로 정해줄 수 있다. 이들을 HTTP Response Header에 Expires에 해당하는 값이 없을 경우 default로 Expires 값을 결정하기 위해 사용된다[7]. NOW는 현재 시간을 나타내며, OBJ_AGE는 해당 웹 객체가 웹 서버에 의해서 만들어진 후의 경과시간을 나타낸다. LM_AGE는 해당 웹 객체가 만들어졌을 때 최근에 갱신된 이후에 얼마나 오래되었는가를 나타낸다. LM_FACTOR는 OBJ_AGE를 LM_AGE로 나눈 값이다. 그림 6은 재생 알고리즘에서 사용되는 파라미터들을 정리한 것이다.

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 6 재생 알고리즘에서 사용되는 파라미터

4.1.3 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 squid의 재생 알고리즘을 응용하여 사용하였다. OBJ_DATE, OBJ_LASTMOD, Expires는 squid가 제공하는 store.log를 통해서 얻을 수 있고, CONF_MAX, CONF_MIN, CONF_PERCENT 또한 squid가 제공하는 squid.conf를 통해서 알 수 있다. CLIENT_MAX_AGE는 로그를 통해 얻을 수 없고 각각의 클라이언트마다 정해주는 값이 다르므로 일반화 시킬 수 없다. 따라서 이 값은 사용하지 않기로 한다. 그림 7은 Squid의 access log의 형식을, 그림 8은 Squid의 store log의 형식을 나타낸다.

Time	Elapsed	Remotehost	Code/Status	Byte	Method	URL

그림 7 Access Log Format

Time	Action	Status	OBJ_DATE	OBJ_LASTMOD	Expires	Type	Len	Method	URL

그림 8 Store Log Format

웹 객체의 유효성을 검사할 때 고려되어야 될 사항은 NOW, 즉 현재를 어느 시점으로 볼 것인가를 결정하는 방법이다. 다시 말해 웹 문서가 fresh한지 stale한지를 결정할 때 현재의 입력 데이타로 사용될 access log 중에서 stale한 것과 다음 프리페칭이 실행되기 전까지 stale한 것을 함께 고려해야 한다. 따라서 Squid의 재생 알고리즘을 따르되 NOW는 프리페칭 주기를 더한 값을 사용한다. 다음과 같이 정의할 수 있다.

NOW = Current Prefetching Time + Prefetching Frequency

Current Prefetching Time : 현재의 프리페칭 적용 시간

Prefetching Frequency : 프리페칭 주기

본 논문에서는 하루 중에서 off-peak periods의 남는 대역폭을 사용하기 때문에 프리페칭 주기 (prefetching frequency)를 하루로 결정하였다. 그림 9는 프리페칭 주기를 설명한 것인데 가장 최근에 프리페칭을 수행한 시간은 10월 15일 04시이며 현재의 프리페칭 시간은 16일 04시, 다음에 프리페칭할 시간은 17일 04시이다. 현재 (16일 04:00) 프리페칭을 하기 위하여 10월 15일 04:00부터 10월 16일 04:00까지의 access log와 store log를 입력 데이타로 사용한다. 따라서 프리페칭 주기는 하루이다.

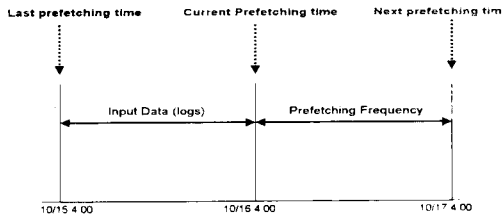

그림 9 프리페칭 주기

4.2 Request Generator

Request generator는 Prefetchable Obejct List에 있는 웹 객체를 off-peak periods시에 캐싱 서버에 저장

되도록 요청하는 모듈이며 하나의 클라이언트로써 동작한다. Prefetchable Object List Generator에 의해 생성된 Prefetchable Object List에 따라 프리페칭할 request을 생성하여 off-peak periods동안에 해당되는 웹 서버에게 request를 보낸다. 본 논문에서는 HTTP를 지원하는 command-line Web client로써 동작하는 wget [19]을 사용하였다. 또한 crontab을 이용하여 자동으로 off-peak periods동안에 프리페칭할 수 있도록 하였다.

Request Generator를 이용하여 Off-peak periods동안 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 agent를 이용한다. SNMP agent를 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. 본 장에서는 시뮬레이션을 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-driven이며 현재 포항공대에서 운영되고 있는 캐싱 서버인 CacheRaQ [29]에서 제공하는 로그를 이용하였다. 이 캐싱 서버는 포항공대 기숙사중에서 16개의 서브넷에서 발생되는 웹 트래픽을 처리하고 있으며 내부적으로는 Squid를 사용하고 있다. 시뮬레이션을 수행할 때 사용되는 입력 파라미터 (input parameter)는 아래와 같다.

• Logs : 시뮬레이션에서 입력 데이타로 사용된 로그는 1999년 10월 15일부터 10월 28일 2주간의 access log와 store log이다.

• 프리페칭 파라미터 : 3.3절의 프리페칭 파라미터를 기반으로 프리페칭할 웹 객체를 선택한다.

- 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.

- 웹 객체의 유효기간 (expiration time)이 지난 객체들 중에서 참조 횟수는 1회 이상, 2회 이상, 3회 이상, 4회 이상, 5회 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 프리페칭했을 때 프리페칭의 성능을 더 높아질 것이라고 예상되나 자세한 비교 분석은 뒤에서 하기로 한다.

- 프리페칭될 웹 문서의 전체 크기는 off-peak periods동안의 남는 대역폭 용량보다 크지 않게 한다.

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

• 재생 알고리즘 파라미터 : 캐싱 서버에 저장되어 있는 웹 객체 중에서 유효기간이 지났는가를 확인하기 위해 Squid의 재생 알고리즘을 사용하는데, 이때 사용되는 파라미터는 4.1.2절에 정의한 것을 사용한다. 그러나 NOW는 4.1.3절에서 설명된 대로 현재의 프리페칭 시간에다 프리페칭 주기를 더한 값이다.

5.2 시뮬레이션 모델

시뮬레이션 모델은 그림 10과 같다. Prefetchable Object List Generator는 off-peak periods동안에 프리페칭할 웹 객체를 선정하여 prefetchable object list를 생성하고 performance analyzer는 prefetchable object list를 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 saving과 bandwidth saving이며 프리페칭의 유효성을 측정하기 위한 성능 평가 척도로는 accracy와 wasted bandwidth이다.

Prefetchable Object List Generator는 위에서 설명된 프리페칭 파라미터와 프리페칭할 웹 객체의 선택

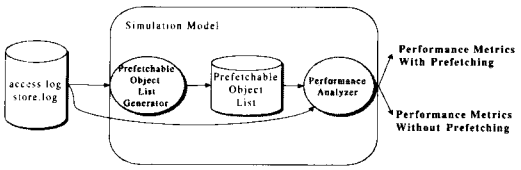

그림 10 시뮬레이션 모델

기준을 바탕으로 하루를 주기로 access log와 store log를 이용하여 prefetchable object list를 생성한다.

Performance Analyzer는 prefetchable object list와 다음날의 access log를 입력 데이타로 받아 prefetchable object list에 들어있는 웹 객체가 다음 날의 access log에도 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 log에 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. 결 과

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 accuracy와 wasted bandwidth의 시뮬레이션 결과를 알아보며, performance parameter중에서 request saving과 bandwidth saving 을 통해서 프리페칭을 사용했을 경우와 사용하지 않았을 경우의 성능을 비교 분석하였다.

6.1 Accuracy

Accuracy는 프리페칭 객체 적중률과 프리페칭 바이트 적중률로 구분하여 비교하였다. 표 2는 참조 횟수를 1회 이상, 2회 이상, 3회 이상, 4회 이상 5회 이상으로 프리페칭을 수행했을 경우의 정확도를 설명하고 있다. 참조 횟수가 2회 이상인 웹 객체만을 프리페칭하였을 경우 프리페칭 객체 적중률은 64.26%였으며 참조 횟수가 1회 이상인 웹 객체도 함께 프리페칭했을 경우 39.42%로 약 25% 정도 떨어졌다. 프리페칭 바이트 적중률을 살펴보면 참조 횟수가 2회 이상인 웹 객체만을 프리페칭했을 경우 46.1%였으며 1회 이상으로 확대하여 프리페칭했을 경우는 22%로 나타났다. 참조 횟수가 높아 질수록 대부분의 프리페칭된 웹 객체가 다음날 요청되어 높은 prefetched object hit을 보여주는 것을 알 수 있었으며 참조 횟수가 높아짐에 따라 다음날 요청될 확률도 linear하게 지속적으로 증가하는 것을 알 수 있었다. 종합하여 볼 때 참조 횟수를 높여서 프리페칭을 할수록 정확도가 높아진다는 것을 알 수 있었다.

표 2 참조 횟수별 정확도

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

6.2 Wasted Bandwidth

Wasted bandwidth와 off-peak periods 동안에 증가되는 대역폭 사용량 증가율을 참조 횟수 별로 나타내면 표 3과 같다. 하루 평균 낭비되는 대역폭은 참조 횟수가 1회 이상인 웹 객체를 대상으로 프리페칭했을 경우 670.72 Mbyte로 전체 웹 트래픽 (3775.043 Mbyte)의 17.8%를 더 증가시키는 결과를 가져오며 참조 횟수가 2회 이상인 웹 객체 만을 대상으로 프리페칭했을 경우에는 172.65 Mbyte로 단지 전체 웹 트래픽의 4.6%만을 증가시켰다. 이것은 참조 횟수가 2회 이상으로 프리페칭했을 때보다 1회 이상으로 했을 경우가 약 4배정도 더 많은 대역폭이 낭비된다는 것을 보여준다. 본 논문에서는 off-peak periods 동안에 프리페칭을 하므로 off-peak periods의 대역폭 사용량은 증가된다. 하루 평균 off-peak 대역폭 사용량은 796.356 Mbyte이다. 따라서 참조 횟수를 1회 이상인 웹 객체를 대상으로 프리페칭하였을 경우 off-peak 대역폭 사용량의 51.9%만큼 증가되었고, 프리페칭시 낭비되는 대역폭은 off-peak periods 동안의 대역폭 사용량을 2회 이상으로 수행했

표 3 참조 횟수별 낭비되는 대역폭

참조 횟수	하루 평균 낭비되는 대역폭 (%)	Off-peak periods 동안의 대역폭 사용량 증가율
1회 이상	670.72 M (17.8 %)	51.9 %
2회 이상	172.65 M (4.6 %)	28.3 %
3회 이상	74.47 M (2.0 %)	16.7 %
4회 이상	35.91 M (1.0 %)	10.8 %
5회 이상	16.97 M (0.5 %)	7.2 %

을 경우 28.3%, 3회 이상으로 수행했을 경우 16.7%, 4회 이상일 경우는 10.8%, 마지막으로 5회 이상으로 수행했을 경우는 7.2%로 가장 낮았다.

6.3 Request Saving

프리페칭을 수행하지 않았을 경우 캐시 객체 적중률은 55%였고, 참조 횟수가 1회 이상인 웹 객체를 대상으로 프리페칭을 수행했을 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. 그림 11은 프리페칭을 수행하지 않았을 경우의 캐시 객체 적중률과 객체 참조 횟수별로 프리페칭을 수행했을 경우의 캐시 객체 적중률을 그래프로 나타낸 것이다.

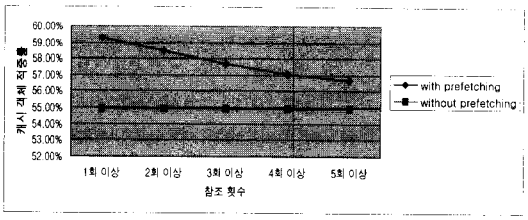

그림 11 참조 횟수별로 프리페칭을 수행했을 경우와 프리페칭을 수행하지 않았을 경우의 캐시 객체 적중률

6.4 Bandwidth Saving

프리페칭을 수행하지 않았을 경우 캐시 바이트 적중률은 31.4%였고, 참조 횟수가 1회 이상인 웹 객체를 대상으로 프리페칭을 수행했을 경우 증가된 캐시 객체 적중률은 5.01%, 참조 횟수가 2회 이상인 경우의 증가율은 3.78 %, 3회 이상인 경우의 증가율은 2.27%, 4회 이상인 경우의 증가율은 1.59%, 참조 횟수가 5회 이상인 경우의 증가율은 1.19%였다. 참조 횟수를 1회 이상으로 프리페칭을 수행했을 경우에는 참조 횟수를 2회 이상으로 수행했을 경우보다 1.23% 증가하였다. 그림 12는 객체 참조 횟수별로 프리페칭을 수행했을 경우의 캐시 객체 적중률과 프리페칭을 수행하지 않았을 경우의 캐시 객체 적중률을 그래프로 나타낸 것이다. Request saving과 비슷한 패턴을 보이고 있다.

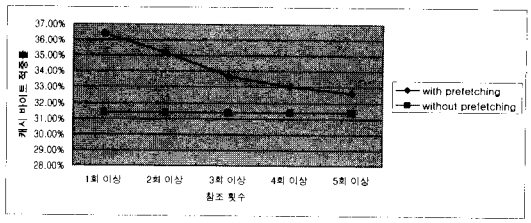

그림 12 참조 횟수별로 프리페칭을 수행했을 경우와 프리페칭을 수행하지 않았을 경우의 캐시 바이트 적중률

본 논문에서 제시한 프리페칭 알고리즘을 사용하면 프리페칭에 의해서 절약되는 대역폭은 실제로는 peak periods동안에 사용될 대역폭을 줄여준다. 그림 13은 프리페칭을 통해서 peak periods동안에 절약되는 대역폭 감소율을 그래프로 표현한 것이다.

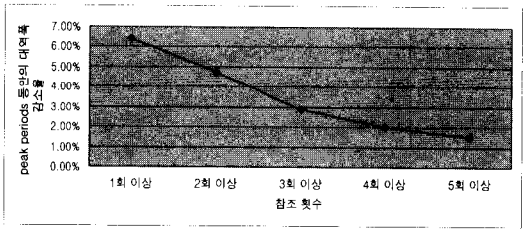

그림 13 참조 횟수별 하루 평균 Peak Periods 동안의 대역폭 사용량 감소율

6.5 Summary

본 논문에서 제시한 방법으로 프리페칭했을 경우의 성능 평가 척도를 참조 횟수별로 정리하면 표 4와 같다.

표 4 Performance Metrics

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

본 논문에서 제안한 프리페칭 알고리즘으로 프리페칭을 수행했을 경우 off-peak periods동안의 대역폭은 증가되고 peak periods동안의 대역폭은 감소된다. 참조 횟수별로 off-peak periods 동안에 증가되는 하루 평균 대역폭 증가율과 peak periods 동안 감소되는 하루 평균 대역폭 감소율을 도표로 나타내면 표 5와 같다.

표 5 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 절약되는 Bandwidth Saving과 낭비되는 대역폭량이다. 참조 횟수가 높아지면 프리페칭의 효율성도 높아지나 프리페칭하는 웹 객체수가 적어지므로 절약되는 대역폭량은 줄어들게 되며 프리페칭에 의해서 낭비되는 대역폭 또한 증가하게 된다. 따라서 프리페칭 효율성, Bandwidth Saving과 Wasted Bandwidth을 고려하여 참조 횟수를 선택한다. 참조 횟수별 프리페칭의 효율성, Bandwidth Saving과 Wasted Bandwidth를 표로 나타내면 표 6과 같다.

표 6 참조 횟수별 프리페칭 효율성과 Peak Periods 대역폭 감소율

참조 횟수	Bs (Mbyte)	Bw (Mbyte)	Ep	Bandwidth Saving	Wasted Bandwidth
1회 이상	189.16	670.72	0.28	5.01 %	17.8 %
2회 이상	141.82	172.65	0.82	3.78 %	4.6 %
3회 이상	85.65	74.47	1.15	2.27 %	2.0 %
4회 이상	60.08	35.91	1.67	1.59 %	1.0 %
5회 이상	45.06	16.97	2.66	1.19 %	0.5 %

표 6을 보면 참조 횟수를 높일수록 프리페칭 효율성은 높아지고 낭비되는 대역폭량도 적어지나 프리페칭에 의해서 절약되는 대역폭량도 적어진다. 따라서 본 논문에서는 전체적으로 만족시킬 수 있도록 참조 횟수를 2회 이상으로 하여 프리페칭 하도록 결정하였다. 그러나 이것은 포항공대 기숙사의 환경에 따른 결정이며 환경이 바뀌면 결정 사항도 변경될 수 있다.

7. 결론 및 향후 연구

본 논문은 peak periods동안의 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 periods동안의 사용되지 않는 대역폭을 사용하여 미리 예상되는 웹 객체를 읽어옴으로써 peak periods동안의 대역폭 사용량을 줄일 수 있다는 것을 보여주었다.

본 논문에서 제안한 프리페칭 알고리즘을 통해 request saving은 캐싱 서버만을 사용했을 경우보다 하루 평균 3.53%가 증가되었고 bandwidth saving은 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. 따라서 캐싱 서버에 본 논문에서 제시한 프리페칭 기법을 사용함으로써 낭비되는 대역폭이 거의 없이 peak periods 동안의 대역폭 사용량을 줄일 수 있었다. 또한, prefetching scheme은 squid라는 캐싱 서버에 독립적으로 add-on feature로써 추가되므로 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 trace나 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. 또한 현재의 프리페칭 시스템은 로그를 한꺼번에 읽어와서 프리페칭이 가능한 웹 객체를 선택하는데, real-time으로 prefetchable object list를 생성할 수 있도록 확장시킬 필요성이 있다.

참 고 문 헌

[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/work-shop.
[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] David H. Crocker, Standard For The Format Of Arpa Internet Text Message, RFC 822, August 13, 1982.

김 숙 향
1998년 동국대학교 컴퓨터공학과 학사 졸업, 2000년 포항공과대학교 정보통신대학원 석사 졸업. 2000년 ~ 현재 LG 정보통신 근무. 관심분야는 웹 캐싱, 인터넷 트래픽 관리

홍 원 기
1983년 Univ. of Western Ontario 전산학 학사 졸업. 1985년 Univ. of Western Ontario 전산학 석사 졸업. 1991년 Univ. of Waterloo 전산학 박사 졸업. 1991년 ~ 1992년 Univ. of Waterloo Post-Doc Fellow. 1992년 ~ 1995년 Univ. of Western Ontario 연구교수. 1985 ~ 현재 포항공대 컴퓨터공학과 부교수. 관심분야는 네트워크 및 시스템 관리, CORBA, 분산멀티미디어 시스템.