웹 환경에서의 캐시 프록시 기술

Caching Proxy Technique for the World Wide Web

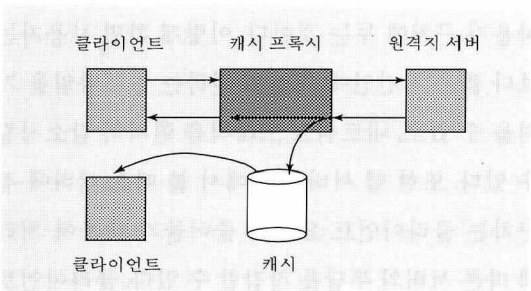
임연호(Y. H. Im) 정보망연구실 선임연구원
박찬범(C. B. Park) 정보망연구실 책임연구원, 실장

웹 이용자가 폭발적으로 증가함에 따라 여러가지 기술적으로 대처해야 할 문제점들이 대두되고 있다. 웹 서버에 접근하는 클라이언트 요구가 급증하면 이에 따라 많은 비용을 들여 네트워크 대역폭과 서버의 처리능력을 확장하지 않으면 안된다. 이러한 문제들을 적은 비용으로 해결할 수 있는 가장 좋은 방법은 캐시를 이용하는 것이다. 캐시는 서버 및 네트워크 부하를 경감시키고, 사용자가 웹 서버로부터 문서를 가져오는데 걸리는 시간을 현격히 감소시킨다. 최근 네트워크상에 프록시 서버를 설치하고 이 곳에 캐시를 두는 캐시 프록시에 관한 연구가 다각도로 이루어지고 있다. 본 고에서는 캐시의 일치성 유지, 삭제 정책 및 다중 캐시 구성 등 캐시 프록시에 관한 주요 기술적 이슈를 살펴보고 앞으로 해결해야 할 과제에 대해서 고찰하였다.

I. 개 요

World Wide Web(이하 웹) 이용자가 폭발적으로 증가하고 있으나 이에 따른 네트워크 설비는 이를 따라가지 못하고 있다. 특히 인기있는 웹 사이트는 하루 수만건의 서버 접속 횟수를 기록하고 있고, 방대한 양의 문서파일을 네트워크상의 여러 경로를 통해 전달하고 있다. 이러한 현상은 여러 측면에서 비용을 유발시킨다. 우선 네트워크 관리자 입장에서 볼 때 네트워크 활용도 증가 추세에 맞춰 대역폭을 지속적으로 확장하지 않으면 안된다. 또한 웹 서버를 관리하는 입장에서 보면 서버의 부하가 가중되기 때문에 더욱 빠른 서버를 도입하거나 기억장치를 증설해야만 한다. 반면 사용자는 자신이 원하는 정보를 서버로부터 얻는데 소요되는 시간이 증가함에 따라 대기시간이 길어지고 이에 따른 생산성 저하를 감수해야 한다. 이러한 문제점들을 해결하기 위해 단순히 네트워크 설비를 확장하거나 서버의 처리 능력을 배가하는 것은 비용이 많이드는 방법이다. 적은 비용으로 이를 해결하는 방법은 캐시를 이용하는 것이다. 캐시의 기본 개념은 사용자가 자주 사용하는 문서들을 서버로부터 미리 가져와 사용자 근처에 두는 것이다. 이렇게 하면 사용자는 보다 짧은 시간안에 자신이 원하는 문서파일을 가져올 수 있고, 네트워크 트래픽을 현저히 감소시킬 수 있다. 또한 웹 서버 입장에서 볼 때도 서버에 접근하는 클라이언트 요구가 줄어들기 때문에 처리에 따른 서버의 부담을 경감할 수 있다. 클라이언트와 가까운 곳에 캐시를 두는 방법에는 두가지가 있

다. 첫째, 클라이언트인 웹 브라우저에 독립적인 캐시를 두는 것으로 이를 클라이언트 캐시라고 부른다. 즉 웹 브라우저는 자신이 과거에 이미 요구한 바가 있는 문서를 자신의 캐시에 저장함으로써 해당 문서를 또다시 요구할 때 즉시 가져올 수 있도록 한다. 클라이언트 캐시는 존속형(persistent)과 비존속형(non-persistent)이 있는데 존속형은 브라우저가 종료된 이후에도 일정 기간 동안 계속해서 캐시 내용을 유지하는 것을 말하며, 비존속형은 웹 브라우저가 종료되는 순간 캐시 내용을 삭제한다. 두번째 방식은 (그림 1)에서와 같이 네트워크상에 프록시 서버를 설치하고, 이곳에 캐시를 두는 것으로 이를 캐시 프록시라고 한다. 캐시 프록시는 동일한 클라이언트가 같은 문서를 두번 이상 요구하거나 서로 다른 클라이언트가 같은 문서를 요구할 때 이전에 요구한 결과로 캐시에 저장된 문서를 내보낸다. 캐시 프록시는 앞서 언급한 클라이언트 캐시에 대해 상위 레벨의 캐시로 생각할 수 있다. 즉 클라이언트는 자신이 요구한 문서파일을 클라이언트 캐시에서 찾지 못한 경우 해당문서를 캐시 프록시에 요구한다. 캐시 프록시는 서로 다른 다수의 클라이언트의 요구에 의해 원격지 웹 서버로부터 가져온 각각의 문서들을 캐시에 저장한다. 따라서 임의의 클라이언트가 문서를 요구할 때 요구한 문서가 이전에 자신 또는 다른 클라이언트 요구에 의해 원격지 웹 서버로부터 불려와 캐시 프록시에 이미 저장되었다면 이 문서를 되돌려 줄 것이다.



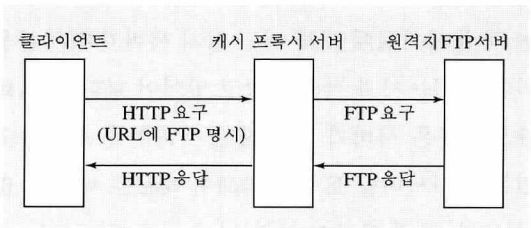
(그림 1) 캐시 프록시 서버 구성

캐시 프록시는 여러 개를 묶어서 다중 캐시로 구성할 수 있다[4, 8]. 따라서 임의의 캐시 프록시는 자신의 캐시에서 클라이언트가 요구한 문서를 찾지 못할 경우 상호 연관된 또 다른 캐시에게 이를 요구할 수 있다. 캐시에 관한 최근의 연구는 주로 캐시 프록시를 대상으로 하고 있다. 캐시 프록시를 이용하면 네트워크 대역폭과 서버의 부하를 현저히 감소시킬 수 있다. 그러나 아직도 캐시와 관련된 여러 가지 해결해야 할 기술적인 문제가 남아 있고, 성능 개선을 위한 여러가지 시도가 진행되고 있다. 본 고에서는 캐시 프록시와 관련된 지금까지의 연구동향을 살펴보고 향후 과제에 대해 고찰하였다.

II. 캐시 프록시 동작

웹 환경에서 클라이언트는 HyperText Transport Protocol(HTTP) 트랜잭션에 의해 HTTP 서버에게 요구 메시지를 보낸다. 그러나 캐시 프록시 서버가 설치된 경우 클라이언트는 HTTP 요구 메시지를 캐시 프록시에게 보낸다. 이때 캐시 프록시는 클라이언트가 요구한 문서파일이 자신의 캐시내에 저장되어 있지 않을 경우 Uniform Resource Locator(URL)에 명시된 원격지 서버로 연결한다. 즉 캐시 프록시 서버는 클라이언트와 원격지 서버의 중간에 위치하면서 원격지 서버에 대해 자신이 마치 클라이언트인 것처럼 동작한다. 이때 캐시 프록시는 클라이언트가 보낸 HTTP 요구 메시지의 헤더 정보를 전혀 수정하지 않고 그대로 원격지 서버로 보낸다. 따라서 클라

이언트 입장에서 볼 때 캐시 프록시를 통해 원격지 서버에 연결하는데 따른 기능 추가나 손실이 발생하지 않는다. 캐시 프록시는 웹이 지원하는 모든 프로토콜, 즉 HTTP외에 File Transfer Protocol(FTP), gopher, Wide Area Information System(WAIS) 및 Network News Transport Protocol(NNTP) 등에 의해 연결이 가능하다. 그러나 클라이언트는 HTTP가 아닌 이들 다른 프로토콜을 이용하여 원격지 서버에 접근할 때에도 캐시 프록시 서버에는 오직 HTTP 트랜잭션으로 요구 메시지를 보낸다[3]. 이와 같이 클라이언트로부터 HTTP 요구를 받은 캐시 프록시는 URL에 명시된 프로토콜에 의해 원격지 서버를 연결하여 클라이언트가 요구한 파일을 가져온 후 이를 다시 HTTP 응답으로 클라이언트에게 보내준다. (그림 2)는 이와 같은 방식으로 클라이언트가 원격지 FTP 서버로부터 파일을 가져오는 것을 보여준다.



(그림 2) 캐시 프록시 서버를 통한 FTP 서버 연결

캐시의 기본개념은 임의의 클라이언트가 캐시 프록시를 통해 가져온 문서를 캐시 프록시에 저장함으로써 나중에 다른 클라이언트가 똑같은 문서를 가져오기 위해 원격지 서버를 연결하는 일이 없도록 하는 것이다. 그러나 언뜻 볼 때 이같은 과정이 단순해 보이지만 캐시와 관련된 여러가지 기술적 문제점들을 면밀히 고려해야 한다. 우선 캐시 프록시 서버에 저장되는 문서를 얼마나 오랫 동안 서버에 보존해야할 것인가? 또한, 새로운 문서를 저장하기 위해 기존의 문서중 어떤 문서를 삭제하는 것이 클라이언트 요구에 대한 히트율(hit rate)을 높이고, 결과적으로 서버의 성능을 향상시킬 수 있을까? 한편 웹 서버에 저장된 대부분의 문서들은 수시로 변경되며, 문서 작성자 조차도 언제쯤 이들 문서를 변경할 것인지를 미리 정할 수 없다. 따라서 원격지 서버에서 원본문서가 갱신되는 순간, 캐시 프록시에 저장된 이 문서에 대한 복사문서와 원본문서가 서로 일치하지 않는 문제가 발생한다.

III. 캐시의 일치성

오늘날 인터넷에서 사용되고 있는 대부분의 웹 캐시는 완벽한 일치성(consistency)을 보장하고 있지 않다. 만약 캐시에 복사된 웹 문서가 원격지 서버에서 최근에 갱신된 원본 문서와 일치하지 않는다면 캐시의 가치는 그만큼 떨어질 것이 분명하다. 이에 따라 원본 데이터의 변화를 즉시 반영하여 캐시 데이터를 갱신함으로써 캐시의 일치성을 유지하기 위한 여러가지 메커니즘이 제시되었고, 이에 대한 연구가 계속 진행되고 있다[7]. 지금까지 분산 파일 시스템을 연구하는 관점에서 캐시의 일치성을 유지하기 위한 여러가지 방법들이 제시되었으나 웹 환경에서 캐시를 일치시키기 위한 프로토콜에 관한 연구는 별로 이루어진 것이 없다. 웹은 파일 접근방식에 있어서 분산 파일시스템과 근본적으로 다르기 때문에 분산 파일시스템에서 얻은 결과를 그대로 적용할 수 없다. 웹에서 기본적으로 다루는 파일의 크기가 분산 파일 시스템에서 보다 훨씬 크고, 파일에 대한 갱신도 오직 한 개의 서버에서만

이루어지기 때문에 분산 파일시스템보다 간단하다고 볼 수 있다. 웹 환경에서 캐시의 일치성 유지를 위한 방법으로는

- 생존 시간 설정(TTL: Time To Live)
- 클라이언트 폴링(polling)
- 무효화 프로토콜(invalidation protocol)

등이 있다. 생존 시간 설정은 캐시에 저장되는 각 문서파일이 얼마만큼 오랫동안 유효한지를 시간값으로 설정해두고 클라이언트에서 이 파일을 요구한 순간 이 값이 종료(expire)되었을 경우에는 원격지 서버로부터 원본 데이터를 가져오도록 하고 있다. 생존 시간 설정에 의한 방법은 완전 무결하게 일치성을 보장하지 않는다. 또한 생존 시간 값을 얼마로 정할 것인지를 결정하는 것이 쉽지 않다. 생존 시간값을 작게 잡으면 원격지 서버로부터 필요 이상으로 자주 원본 데이터를 가져올 수 있다. 이와 반대로 생존 시간값을 크게 잡으면 클라이언트가 캐시로부터 원본과 일치하지 않는 데이터를 가져올 확률이 그 만큼 증가한다. 생존 시간 설정은 비교적 구현하기 쉽다. 이 방법은 신문사에서 제공하는 기사 자료처럼 일정 주기로 갱신되는 데이터에 대해서는 효과적으로 적용할 수 있다. 클라이언트 폴링은 클라이언트가 주기적으로 원격지 서버에 폴링을 하여 캐시에 저장된 문서파일이 유효한지 확인하는 것이다. 클라이언트 폴링에서도 역시 폴링 주기를 어떻게 설정하는가 하는 것이 중요한 이슈가 된다. 일반적으로 캐시에 오래 유지되고 있는 파일일수록 수정될 확률이 그만큼 작은 것으로 나타났으며, 이를 토대로 폴링 주기를 설정하기 위해 갱신 임계값을 이용하는 방법이 제안되었다. 갱신 임계값은 파일이 캐시에 머무른 시간에 대한 퍼센트로 표시된다. 이 방법은 파일의 일치성 유무를 확인한 가장 최근 시점 이후 경과된 시간이, 갱신 임계 값과 문서파일이 캐시에 머무른 시간과 곱해서 얻은 값을 초과하지 않으면 해당 문서파일이 일치하는 것으로, 아니면 일치하지 않는 것으로 간주한다. 따라서 캐시에 존재한 시간이 짧은 파일일수록 자주 폴링을 하게된다. 생존 시간 설정이나 클라이언트 폴링방식은 캐시 서버가 클라이언트의 요구에 대해 원본파일과 일치하지 않는 파일을 내보낼 가능성이 있다. 이에 비해 무효화 프로토콜은 캐시에 저장된 복사본이 항상 원격지 서버의 원본과 일치하도록 하는데 주안점을 두고 있다. 무효화 프로토콜은 원격지 서버로 하여금 캐시 데이터를 항시 추적하도록 하고 있다. 원격지 서버는 문서파일이 갱신될 때마다 캐시에서 유지하고 있는 복사본이 더 이상 유효하지 않다는 것을 캐시 프록시 서버에게 알려준다. 따라서 원격지 서버는 자신이 제공하는 각 문서파일에 대해 어느 캐시 서버가 복사본을 가지고 있는지 추적하여 알고 있어야 한다. 무효화 프로토콜은 서버가 자신의 문서파일을 유지하고 있는 캐시서버를 모두 관리해야 하므로 비용이 많이 든다. 현재 웹에서 가장 널리 사용되고 있는 캐시는 CERN 웹 서버에 포함된 프록시용 캐시이다. CERN 캐시는 생존 시간 설정 방식을 사용한다. 즉 클라이언트가 캐시에 HTTP 요구를 한 시점에서 클라이언트가 요구한 파일의 생존 시간값이 종료되지 않은 상태이면 해당 파일을 클라이언트에게 돌려주고, 반대로 종료되었다면 원격지 서버에 "If-Modified-Since" 요구 메시지를 보내 해당 파일의 변경 유무를 확인한다. 이때 만약 파일이 변경되지 않았다면 캐시에 있는 파일을 그대로 클라이언트로

보내고 동시에 변경 유무를 확인하는 과정에서 원격지 서버로부터 새로 얻은 값으로 생존 시간값을 다시 설정한다. 그러나 파일이 변경되었다면 정상적인 요구 메시지에 의해 서버로부터 파일을 가져와 이를 클라이언트에게 돌려주고 동시에 이를 캐시에 저장한다[1, 4].

최근의 시뮬레이션 결과[7]에 따르면 캐시에 파일이 저장된 이후 경과된 시간에 따라 폴링 주기를 다르게 하는 다소 변형된 클라이언트 폴링 방식이 생존 시간 설정이나 무효화 프로토콜보다 네트워크 대역폭과 서버의 부하를 줄이면서도 일치성에 위배되는 데이터를 5% 이하로 낮출 수 있음을 보여주고 있다.

IV. 삭제 정책

캐시의 크기는 유한하다. 따라서 클라이언트가 캐시에 등록되어 있지 않은 파일을 요구하면 캐시 프록시 서버는 이를 원격지 서버로부터 가져온 후 이를 캐시에 저장하기 위해 한개 이상의 기존 파일을 삭제해야 하는 경우가 발생한다. 이때 어떤 파일을 삭제할 것인지를 결정하는 삭제 정책에 따라 캐시의 성능이 달라진다. 캐시의 성능을 결정하는 기준으로 히트율을 사용한다. 히트율은 전체 클라이언트 요구 중 얼마나 많은 요구가 캐시에 의해 이루어졌는지를 나타낸다. 캐시 프록시에서 삭제 대상 파일은 불과 수 바이트의 텍스트 파일에서부터 수메가 바이트의 비디오 파일까지 매우 다양하다. 따라서 텍스트, 오디오 및 이미지 문서 등 문서의 종류가 삭제 정책과 밀접한 관련이 있다. 삭제 정책으로는 First-In, First-Out(FIFO), Least Recently Used (LRU), Least-Frequently Used(LFU), 문서의 크기를 고려하여 LRU를 일부 변형한 LRU-MIN 및 Hyper-G 등이 나와 있다. 삭제 정책과 관련된 또 다른 이슈는 어느 시점에서 얼마나 많은 파일을 캐시로부터 삭제하는가 하는 문제이다. 삭제 정책을 실행하는 시점은 다음과 같이 세가지 경우로 생각할 수 있다.

- 요구 발생시: 원격지 서버로부터 가져온 파일이 캐시에 남아 있는 공간을 초과할 때마다 실행한다.
- 주기적 삭제: 일정 시간을 주기로 실행한다.
- 주기적 삭제 및 요구 발생시: 요구가 발생할 때마다 삭제하면서 동시에 일정시간(예를 들어 하루에 한번)을 주기로 실행한다.

또한 삭제시 얼마나 많은 파일을 삭제할 것인지를 결정하는 방법도 중요하다. 우선 요구한 만큼 삭제하는 것을 생각할 수 있다. 즉 캐시의 여유 공간이 서버로부터 가져온 파일의 크기와 같거나 초과하는 시점에서 삭제를 중지한다. 또 다른 방법은 캐시의 여유 공간에 대한 임계값을 설정하고, 일정시간을 주기로 임계값에 도달할 때까지 삭제하는 것이다. 이 방식에서는 만약 다음 번 삭제 시간 이전에 캐시가 100%로 꽉 채워진다면 즉시 앞에서 설명한 요구한 만큼 삭제하는 방식을 적용한다. 파일 삭제는 시간이 많이 소요되고 캐시 서버의 부하를 가중시킨다. 따라서 여유공간이 임계값에 도달할 때까지 주기적으로 파일을 삭제하는 것이 부하를 경감시킬 수 있다. 그러나 주기적인 파일 삭제는 클라이언트가 조만간 요구할지도 모르는 파일을 필요 이상 빨리 삭제함으로써 히트율을 떨어뜨릴 수 있다.

캐시 삭제 정책과 관련된 문제는 과연 얼마나 많은 복사된 파일을 캐시에 저장할 수 있는가 하는 데

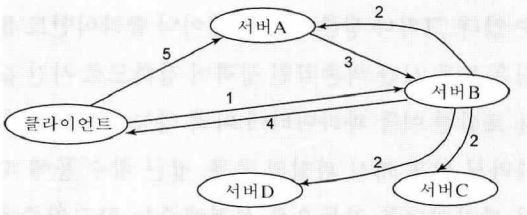
서 출발한다. 만약 캐시의 저장 능력을 무한대로 가정한다면 히트율은 최대값이 될 것이다. 그러나 현실적으로 캐시 서버가 한정된 디스크를 사용하기 때문에 삭제는 불가피하다. 시뮬레이션 연구결과 [9]에 따르면 LRU, Hyper-G 등 여러가지 삭제 정책 중에서 문서의 크기를 우선적으로 고려한 삭제 알고리즘이 가장 좋은 히트율을 얻을 수 있음을 보여주고 있다.

V. 다중 캐시

한 개의 캐시에 의존하는 클라이언트 개수가 일정수준 이상으로 증가하면 캐시 프록시에 병목현상이 발생하고, 이로 인해 캐시의 효율이 떨어진다. 또한 캐시 서버에 문제가 발생하면 이를 대체할 수 있는 캐시가 없기 때문에 캐시 이용이 불가능해진다. 이러한 문제점을 해결하기 위해 복수의 캐시 서버를 다중으로 구성하는 방법이 사용되고 있다. 이와 같이 여러 개의 캐시를 다단계로 상호 연계하여 구성할 경우 클라이언트 요구 증가에 따라 캐시를 점진적으로 확장할 수 있을 뿐 아니라 클라이언트 입장에서 볼 때 캐시의 크기가 커지므로 히트율을 높힐 수 있다.

1. 분산 캐시

분산 캐시는 독립적인 여러 개의 캐시가 상호 협력하여 일련의 클라이언트 그룹에게 공동으로 서비스를 제공하는 것으로서 서버간에는 이를 위해 상호 대화할 수 있는 프로토콜이 필요하다. 이는 곧 독립적인 캐시 서버를 사용하는 클라이언트 그룹들간에 캐시 자원을 상호 공유하는 것을 의미한다. 이 때, 보다 튼튼한 캐시 운용환경을 위해서 모든 서버는 기능적으로 동등하게 구현되며, 임의의 서버는 어느 클라이언트로부터 메시지가 도착하더라도 서비스할 수 있어야 한다. 물론 이 경우 서버간에 부하를 균형있게 분산시킬 수 있는 수단이 강구되어야 하고, 서버간에 주고 받는 트래픽으로 인해 네트워크 대역폭을 급격히 증가시키는 일이 없어야 한다. 분산 캐시의 예로 멀티캐스트에 의한 다중 캐시 구성을 들 수 있는데 이의 동작은 (그림 3)과 같다[4]. 이 방식에서 클라이언트는 HTTP 요구를 할 때 여러 개의 캐시프록시서버중 무작위로 한 개를 선택하고, 선택된 서버에 요구메시지를 보낸다. 이때 선택된 서버를 주 서버라고 하고, 만약 주 서버가 클라이언트가 요구한 파일을 갖고 있을 경우 이를 클라이언트에게 되돌려준다. 그러나 주 서버가 자신의 캐시에서 파일을 찾지 못한 경우에는 자신을 제외한 모든 캐시 서버에게 해당 파일을 갖고 있는지 확인하기 위해 멀티캐스트 방식으로 질의 메시지를 보낸다. 만약 주어진 시간안에 주 서버가 응답을 받지 못하면 주 서버는 원격지 서버로부터 원본 파일을 가져와 클라이언트에게 전달하고, 이 파일을 나중에 다시 사용할 수 있도록 자신의 캐시에 저장한다. 그러나 만약 주 서버가 다른 캐시 서버로부터 응답을 얻은 경우에는 클라이언트로 하여금 해당 서버에 다시 요구 메시지를 보내도록 지시한다. 이 방식에서는 클라이언트가 무작위로 선택한 캐시 서버가 파일을 갖고 있지 않을 경우 두 번의 요구 메시지, 즉 한 번은 주 서버에게 또 한 번은 파일을 갖고 있는 서버로 연결해야하므로 오버헤드 요인이 발생한다. 이러한 오버헤드를 줄이기 위해서 한 개의 HTML 문서파일에 inline으로 삽입된 이미지 파일들을 같은 장소에 저장하도록 한다.

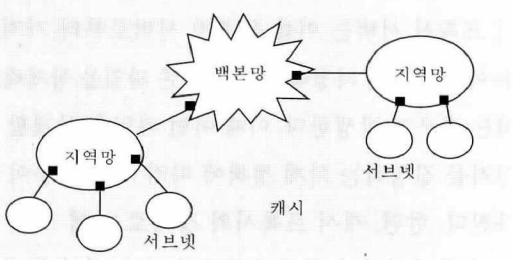


1. 무작위로 주 서버 선택
2. 주 서버가 파일을 갖고 있지 않은 경우 멀티캐스트로 파일보유 확인
3. 서버A가 주 서버에게 자신이 파일을 갖고 있음을 알림
4. 주 서버가 클라이언트에게 서버A가 파일을 가지고 있음을 알림
5. 클라이언트가 서버A에게 다시 요구 메시지를 보냄

(그림 3) 멀티캐스트에 의한 다중 캐시 동작

2. 계층적 캐시

계층적 캐시 구성에서는 자신보다 상위에 부모 캐시를, 자신과 동등한 계층에는 형제 캐시를 둔다. 형제 캐시의 역할은 캐시 서버의 부하를 분산시키는데 있다. (그림 4)는 계층적 캐시에 대한 간단한 구성의 예를 보여준다. 각 계층에 존재하는 캐시는 간단한 해석(resolution) 프로토콜을 사용하여 클라이언트가 요구한 파일을 원격지 서버로부터 직접 가져올 것인지 아니면 부모 또는 형제 캐시로부터 가져올 것인지를 결정한다. 즉 URL을 해독한 결과 클라이언트가 요구한 파일이 Common Gateway Interface(CGI) 스크립트 수행에 따라 동적으로 변하기 때문에 캐시에 저장할 수 없는 파일이거나 URL이 클라이언트와 인접한 지역에 있는 경우 해당 서버에 직접 연결하여 원본 파일을 가져온다. 계층적 캐시 구성에서는 캐시가 클라이언트가 요구한 URL을 자신의 캐시에서 찾지 못한 경우 형제 또는 부모 캐시에게 원격지 프로시져 호출을 실행하여 해당 URL을 가지고 있는지 확인하고, 그 중 가장 빨리 응답을 해준 캐시로부터 파일을 가져온다. 결국 해석 프로토콜을 사용하는 목적은 캐시이든 아니면 원격지 서버이든 자신이 찾고자 하는 파일을 가장 빨리 제공할 수 있는 캐시 서버를 찾는 것이다.



(그림 4) 계층적 캐시 구성의 예

인터넷 토폴로지는 계층적 구조로 구성되며 따라서 계층적 캐시 구성은 자연스러운 것이라는 주장과 함께 계층적 캐시는 50%에 달하는 히트율을 얻을 수 있다는 연구결과[8]가 나왔다. 또 다른 연구[7]에 따르면 인터넷 트래픽중 FTP가 가장 큰 비중을 차지하고 있으며, 만약 클라이언트가 속한 지역망과 백본망 경계지점에 FTP 캐시를 둘 경우 백본망으로 나가는 FTP 트래픽의 42%를 줄일 수 있는 것으로 나타났다.

VI. 결론 및 향후과제

웹 이용자가 폭발적으로 증가함에 따라 여러 가지 기술적으로 대처해야 할 문제점들이 대두되고 있다. 이러한 문제점들을 해결할 수 있는 가장 좋은 방법은 캐시를 이용하는 것이다. 캐시는 서버 및 네트워크 부하를 경감시키고, 사용자가 서버로부터 문서를 가져오는데 걸리는 시간을 현격히 단축시킨다. 캐시의 크기는 유한하다. 따라서 클라이언트

가 캐시에 등록되어 있지 않은 파일을 요구하면 캐시 프록시 서버는 이를 원격지 서버로부터 가져온 후 이를 캐시에 저장하기 위해 기존 파일을 삭제해야 하는 경우가 발생한다. 이때 어떤 파일을 삭제할 것인지를 결정하는 삭제 정책에 따라 캐시 성능이 달라진다. 한편, 캐시 프록시의 장애로 인해 캐시 운용이 중단되거나 클라이언트 수가 늘어남에 따라 캐시에 과부하가 걸리는 문제는 다중 캐시를 구성함으로써 해결할 수 있다. 특히 계층적 캐시를 구성할 경우 하루에 수만건 이상 접근 횟수를 기록하는 인기있는 웹 사이트의 부하를 분산시키는 효과를 얻을 수 있다. 계층적 캐시에서는 주로 최상위에 있는 캐시가 과중한 요구를 받기 때문에 최상위 서버의 성능이 무엇보다도 중요하다. 캐시 메커니즘은 기본적으로 원격지 서버에서 원본 파일을 가져다가 캐시에 저장하므로 원본 파일이 갱신되었을 때 이를 즉시 캐시에 반영하지 않으면 내용이 일치하지 않는 문제가 발생한다. 캐시의 일치성을 유지하기 위한 방법으로 생존 시간 설정, 클라이언트 폴링 및 무효화 프로토콜 등이 제안되었으며, 이들을 대상으로 완벽한 일치성 유지 유무, 구현시 용이성, 네트워크 및 서버에 주는 부담 또는 캐시 성능에 미치는 영향 등을 놓고 비교하는 연구가 진행되고 있다. 일반적으로 많이 사용되는 생존 시간 설정 방식은 네트워크에 부하를 덜 주고 구현이 용이하지만 서버에 과중한 부담을 준다. 반면에 클라이언트 폴링, 무효화 프로토콜 등도 나름대로의 이점을 가지고 있다. 무효화 프로토콜은 프로토콜이 복잡해서 구현하는데 비용이 많이들지만 완벽한 일치성을 보장한다. 클라이언트 폴링은 구성하기에 따라 무효화 프로토콜보다 네트워크 대역폭과 서버의 부하를 증가시키지 않으면서 높은 일치성을 유지할 수 있다. 그러나 생존 시간 설정이나 클라이언트 폴링은 모두 시간 의존적인 성격이 강하므로 시간 값을 포함한 이들 파라미터에 따라 성능이 좌우된다. 따라서 향후 캐시 파일의 종류, 접근 횟수 등에 따라 파라미터를 자동으로 설정해주는 알고리즘에 관한 연구가 필요하다. 캐시가 인터넷에서 대두되고 있는 여러 가지 문제점을 모두 해결할 수 있는 만병통치약은 아니다. 아직까지 정적으로 생성된 문서파일에 대해서만 캐시를 적용할 수 있다. 즉 클라이언트인 사용자의 선택에 의해 동적으로 생성되는 문서파일에 대해서는 캐시를 이용할 수 없다. 앞으로 웹을 상업적인 목적으로 이용하는 사례가 늘어나면서 이들 동적인 문서에 대한 요구가 그만큼 증가할 것으로 예상된다. 또 다른 문제로 캐시의 일치성 문제를 생각할 수 있다. 아직까지 무효화 프로토콜 외에는 캐시의 일치성을 완벽하게 보장해 줄 수 있는 방법이 나와 있지 않다. 그러나 무효화 프로토콜을 구현하기 위해서는 기존의 웹 서버와 클라이언트를 모두 수정하여 무효화 프로토콜을 지원할 수 있도록 해야 하기 때문에 현실성이 없다. 한편 현재의 HTTP 프로토콜은 클라이언트가 GET 요구를 할 때마다 서버와 TCP 연결을 성립시켜야 하기 때문에 통신상의 오버헤드가 크다. 이에 따라 한번의 TCP 연결로 여러개의 HTTP 요구를 처리하도록 하는 새로운 HTTP 프로토콜에 대한 연구가 진행되고 있고, 결국 캐시 운용에도 영향을 미칠 것으로 보인다. 따라서 지금까지 언급한 여러가지 현안 문제를 비롯해 향후 HTTP 개량에 따른 캐시 프록시 메커니즘의 수정 및 성능개선에 관한 연구가 지속적으로 이루어져야 할 것이다.

참고문헌

[1] T. Berners-Lee, "Hypertext transfer protocol - HTTP/1.0," HTTP Working Group Internet Draft, Feb. 1996.

[2] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching proxies: limitations and potentials," in *4th International World-Wide Web Conference*, Boston, Dec. 1995, pp. 119-133.

[3] A. Luotonen and K. Altis, "World-Wide Web proxies," *Computer Networks and ISDN Systems*, 27(2), 1994.

[4] R. Malpani, J. Lorch, and D. Berger, "Making World Wide Web Caching Servers Cooperate," in *4th International World-Wide Web Conference*, Boston, Dec. 1995, pp. 107-117.

[5] M. Abrams, S. Williams, G. Abdulla, S. Patel, R. Ribler, and E. A. Fox, "Multimedia traffic analysis using Chitra95," in *Proc. ACM Multimedia '95*, San Francisco, Nov. 1995, pp. 267-276.

[6] T. Berners-Lee, "Propagation, replication and caching," <URL:http://www.w3.org/hypertext/WWW/Propagation/Activity.html>, Mar. 1995.

[7] J. Gwertzman, "World-Wide Web Cache Consistency," *Proc. 1996 USENIX Technical Conference*, San Diego, CA, Jan. 1996.

[8] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell, "A hierachical internet object cache," *Proc. 1996 USENIX Technical Conference*, San Diego, CA, Jan. 1996.

[9] M. Abrams, S. Williams, G. Abdulla, C. R. Standridge, and E. A. Fox, "Removal policies in network caches for World-Wide Web documents," *Proc. ACM SIGCOMM*, Stanford, CA, Aug. 1996, pp. 293-305.