2001년도 대한전기학회 하계학술대회 논문집 2001.7.18-20

커널레벨에서의 웹가속기 개발에 대한 연구

고성준, 박종규, 민병조, 김학배
연세대학교 전기·전자공학과

Developement of a Web Accelerator in the Kernel

Soungjun Ko, Jyounggue Park, Byungjo Min, and Hagbae Kim
Dept. of Electrical and Electronic Engineering, Yonsei University

Abstract - 웹가속기(web accelerator)는 웹서버와 동일한 컴퓨터에서 동작하여 웹서버의 서비스 속도를 향상시키는 것을 주된 목적으로 한다. 이는 캐싱서버와는 달리 별도의 머신을 필요로 하지 않고 커널레벨에서 클라이언트의 요청을 가로채어 처리하므로 일반적인 Apache와 같은 사용자 응용 프로그램보다 더 빠르게 서비스를 수행할 수 있다. 정적페이지(static page)와 동적페이지(dynamic page)를 처리할 수 있고, 커널레벨에서 동작하므로 일반적인 웹서버가 지니는 Multi-thread 구조의 속도상 overhead와 메모리 복사와 디스크 접근에서 일어나는 자원 낭비를 줄임으로써, 웹서버의 응답속도(response time) 및 초당처리요청수(request/sec)를 개선시킬 수 있다.

1. 서 론

인터넷 사용자의 수는 최근 몇년사이에 엄청나게 늘어났다. 그에 따른 네트워크의 발전도 급성장을 거듭해왔고, 수많은 사람들이 인터넷을 통해 정보를 공유하고 이용하게 되었다. 그러나 너무나 많은 정보의 양때문에 사용자들은 적은 시간안에 더 많은 정보를 얻어야 할 필요성이 생겼다. 그에 따라 인터넷 트래픽은 급격하게 증가하였고 네트워크에 걸리는 부하는 커졌다. 이러한 상황에서 조금 더 빠른 웹 서비스를 위한 여러 가지 기술들이 등장하게 되었고 그에 따라 인터넷 트래픽의 대부분을 차지하는 HTTP를 효율적으로 관리하기 위한 방법들이 등장하였다. 그 대표적인 것이 캐시이다. 웹캐시는 상대적으로 속도가 느린 디스크의 접근을 최소화하여 클라이언트의 요청에 대한 응답 데이터를 메모리에 저장하고 관리하여 클라이언트의 요청이 있을 때 디스크의 내용에 접근하기 전에 캐시의 내용을 살펴서 해당 데이터가 있으면 바로 응답을 함으로써 속도를 크게 향상시킬 수 있는 기술이다. 이러한 캐시의 성능을 높이기 위해 다양한 기술[5]이 연구되어지고 있다. 실제로 곧 사용되거나 앞으로 활용될 기술을 보면 커널 I/O을 최적화한다든지, 캐시 전용 운영체제를 사용한다든지, 클라이언트가 어떤 객체를 요청할지 미리 예측하여 능동적으로 캐싱을 하는 컨텐츠 프리페칭(contents prefetching) 등이 있다. 본 연구에서 구현한 커널단에서의 웹가속기(이후로 커널가속기라 칭함) 역시 캐시의 방식을 충분히 이용하고 있으며 앞에서 언급한 여러 기술들과 연동하여 더 나은 성능을 얻을 수 있다는 특징이 있다.

기존의 웹서버 머신들이 웹 서비스를 제공하기 위해 사용하는 Apache와 같은 응용프로그램들은 빠른 속도의 서비스보다는 여러 가지 다양한 웹 요청에 대해 유연하게 대처하고자 하는 목적이 더 크다. 갈수록 다양해지는 웹요청에 대해 많은 검사과정이 필요하게 된다. 그리고 클라이언트의 요청에 응답하기 위해 요청된 데이터는 여러 계층의 소프트웨어적인 단계를[2] 지나가야 하기 때문에 같은 내용을 여러번 복사할 필요가 있다. 예를 들어 파일시스템과 응용프로그램 사이라든지, 데이터 전송시 시스템의 운영체제나 디바이스 드라이버의 여러과정을 통과해야 할 때라든지와 같은 반복된 복사에 따른 overhead가 발생하게 됨은 명백하다. 이러한 문제를 해결하기 위해 본 논문에서 구현한 커널가속기는 커널레벨[2]에서 웹 요청을 처리하게 된다. 그렇게 함으로써 메모리 복사에 따른 overhead를 줄이고 캐싱의 방법을 충분히 활용하여 디스크의 접근횟수도 최대한 효율적으로 줄이게 되어 속도의 향상을 얻을 수 있게 된다. 즉 Apache와 같은 응용프로그램이 웹요청을 처리하기 전에 미리 커널레벨에서 그 내용을 가로채어 처리함으로써 더 나은 성능향상을 얻는 것이다. 또한 기존의 웹가속기가 정적페이지만 처리할 수 있는 것에 반해 동적페이지까지도 처리하게 된다.

본론에서는 웹가속을 위한 방법중 캐싱서버의 종류[1]와 캐싱에 사용되는 여러 기술중 커널 I/O 최적화, 캐시 전용운영체제, 컨텐츠 프리페칭(contents prefetching)과 같은 주요 기술들에 대해 간략히 알아보고 커널가속기의 구현 양식에 대해서 언급한다. 그리고 응용프로그램과의 성능 비교를 위해 응답속도와 초당처리요청수의 시뮬레이션 결과를 살펴본다.

2. 본 론

2.1 캐시서버의 종류와 동작방식

이 장에서는 일반적으로 사용되어지고 있는 웹 가속을 위한 캐시서버에 대해 살펴본다. 캐시서버는 웹서버의 커널안에서 동작하는 커널가속기와는 다르게 하나의 독립적인 서버를 형성하게 된다. 그러므로 커널가속기와 캐시서버를 연동하여 사용하면 더 나은 성능을 얻을 수 있을 것이다.

2.1.1 프록시 캐시(Proxy Cache)

프록시 캐시는 클라이언트로부터 HTTP 요청을 받아서 요청한 데이터가 캐시에 있으면 처리해 주는 방식이다. 캐시에 없으면 원래 서버에서 데이터를 얻어 사용자의 요청을 처리하게 된다. 일반적으로 같은 네트워크 상의 많은 사용자들의 요청들을 처리할 수 있도록 네트워크 종단에 위치하도록 하나, 실제적으로는 사용자나 서버의 위치와 관련 없이 인터넷상 어느 곳에서나 위치할 수 있다. 그러나 프록시 캐시는 single point of failure가 존재해서 프록시 캐시가 고장이 나면 이를 사용하는 사용자에게는 네트워크가 마비된 것처럼 보이게 되는 단점이 있다. 그리고 만약 서버에 문제가 발생하여 사용할 수 없게 되었을 때 사용자들이 일일이 웹 브라우저에서 설정을 다시 해줘야만 하는 불편이 있다.

2.1.2 투명 캐시(Transparent Cache)

투명 캐시는 프록시 캐시와는 달리 네트워크 설정에 대한 변경이 필요 없기 때문에 짧은 시간에 설치를 할 수 있고 이러한 이유로 사용자들에게는 캐시가 투명하게 보이게 된다. 투명 캐시는 라우터 레벨과 스위치 레벨, 두 가지 위치에 배치될 수 있는데, 스위치 레벨를 이용, 여러 대의 캐시를 배치해서 부하분산(load balancing)

을 할 수도 있다. 라우터 레벨에서는 라우터가 클라이언트의 요청을 받아서 어떠한 정책을 이용하여 특정한 투명 캐시에게 전달해 주는 방식을 사용한다. 투명 캐시 역시 같은 네트워크 상의 많은 사용자들의 요청을 처리할 수 있도록 대부분 네트워크 종단에 위치한다.

2.1.3 리버스 캐시(Reverse Cache)

이 방법은 앞의 두 캐시서버와는 달리 사용자쪽이 아닌 서버쪽에 설치한다. 그렇기 때문에 리버스 캐시는 특정한 소수의 서버의 데이터만을 캐시하고 그 특성으로 리버스라는 이름을 가지게 되었다. 사용자가 서버에 오는 길목 중간에 캐시서버를 설치함으로써 서버의 부하를 줄일 수가 있고 사용자는 보다 빨리 콘텐츠를 전달받을 수가 있다. 캐시서버는 일반 웹 서버보다 콘텐츠 전달능력이 뛰어나기 때문에 사용자에게 빠른 속도로 콘텐츠를 전달할 수가 있다. 리버스 캐시는 프록시 캐시나 투명 캐시 같은 클라이언트 측면의 캐시와는 완전히 독립적인 캐시이다. 따라서 리버스 캐시와 프록시/투명 캐시 등을 연동해서 사용한다면 훨씬 더 나은 성능을 얻을 수 있다.

2.2 캐시의 성능향상을 위한 기법들

보다 빠르고 완벽한 웹가속기의 성능을 얻기 위해 여러 가지 연구[1]가 진행되고 있다 그중 최근 사용되고 있는 기술을 간략히 언급한다.

2.2.1 커널 I/O 최적화

메모리에 비해 상당히 느린 디스크의 I/O의 효율을 높이기 위해 파일 시스템을 캐시의 처리 구조에 맞게 최적화하는 것으로 속도 향상을 가져올 수 있다. 보통 운영체제에서 제공하는 일반적인 목적의 파일 시스템을 그대로 쓰기보다는 파일 시스템을 웹캐시에 적합하게 만들어 객체들을 처리할 때 I/O를 최소화하여 성능 향상을 가져올 수 있다. 디스크 I/O를 최적화 하기 위해서 디스크상의 객체 배치와 객체의 데이터 구조, 그리고 캐시의 내부 처리에서 디스크에 대한 접근 횟수등을 고려한다.

2.2.2 캐시 전용운영체제

이 방법은 본 논문에서 구현한 커널가속기와 같은 맥락이라 할 수 있다. 웹 전용의 운영체제를 만들어 최적의 서비스를 구현하는 것이다. 다시말해서 일반적인 운영체제는 캐시만을 돌리는 서버를 생각할 때 불필요하거나 비효율적인 측면이 많다. 그러므로 대부분의 운영체제들이 다중처리를 위해 채택하고 있는 쓰레드 기반의 접근 방식보다는 이벤트 기반의 접근 방식을 채택하여 lock으로 인한 대기시간을 없애고 빠른 처리를 할 수 있게 하도록 하는 것등과 같은 방법을 연구하는 것이다.

2.2.3 컨텐츠 프리페칭(Contents Prefetching)

컨텐츠 프리페칭은 멀리 떨어져 있는 서버로부터 클라이언트의 요청을 예측하여 그 내용을 웹캐시에 미리 가져온다는 뜻이다. 컨텐츠 프리페칭을 구현하기 위한 가장 중요한 포인트는 역시 클라이언트의 요청을 예측하는 알고리즘이다. 한 방법으로 프리페칭은 로컬기반과 서버 힌트 기반으로 구분할 수 있는데, 로컬기반의 방식은 웹캐시 자체에서 클라이언트들의 요청 패턴들을 분석해서 무엇을 미리 가져 올 것인가를 고르는 방식이다. 반면 서버 힌트 기반의 프리페칭은 서버가 자체적인 통계자료 같은 것을 이용하여 그것을 기반으로 프리페칭 할 내용을 정하는 방식이다. 서버 힌트 방식의 단점은 서버로부터 클라이언트로의 힌트에 대한 정보와 관련, 서로 통신을 주고 받아야 할 필요가 있기 때문에 클라이언트와 서버사이의 종합이 다소 어려운 면이 있다.

2.3 커널가속기의 구조

2.3.1 전체구조(커널쓰레드&모듈)

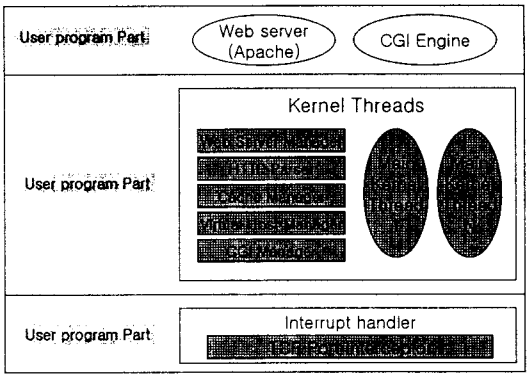

그림 1. 커널가속기의 전체구조

커널가속기는 Linux OS상에서 구현된 위치에 따라 크게 Interrupt part, Kernel part[3]로 나뉘어 질 수 있으며, 이는 Apache 등의 사용자 프로그램 부분과 연동되어 동작하게 된다. 네트워크를 통해서 클라이언트의 요청이 전달되게 되면, Interrupt part의 TCP port interceptor[4]가 이 데이터를 가로채어 먼저 커널가속기에게 전달되도록 하므로 Apache 등의 사용자 프로그램 웹서버 보다 항상 앞서 처리 할 수 있게 된다. Kernel thread 형태로 동작하는 커널가속기는 CPU 개수와 같은 수의 Main thread가 돌면서 HTTP 요청을 해석하는 HTTP parser, 응답할 데이터를 메모리 상에서 관리하는 Cache manager, Virtual hosting을 지원하는 Virtual host manager, 사용자 프로그램 부분의 웹 서버를 관리하는 Web Server Manager, CGI를 커널에서 처리할 수 있게 관리하는 CGI Manager의 라이브러리를 이용하여 전달 받은 클라이언트의 요청을 해석 및 처리하게 된다. 클라이언트의 요청은 여러가지 조건에 따라 커널가속기가 자체적으로 처리할 것인지 또는 메모리 상에 저장할 것인지, 사용자 프로그램 부분의 웹서버에게 이양해야 하는 것인지 등을 신속하게 판단하여 이에 따른 후속 프로세스를 준비하고 클라이언트에게 응답을 하게 된다. 커널가속기는 앞에서의 설명처럼 커널내에서 동작하게 되므로 이에따라 속도 향상은 물론이고 필요에 따라 커널의 다른 소스들을 쉽게 이용할 수 있게 된다.

2.3.2 데이터 전달 구조

클라이언트로부터 전달된 요청은 일반적인 웹 서버가 열어놓은 TCP 포트로 전달되게 된다. 커널가속기는 사용자 프로그램 부분의 웹 서버가 전달 받을 클라이언트의 요청을 커널에서 먼저 처리하여 속도를 개선하면서도 Apache등의 웹 서버가 그 사실을 모르도록 하는데 초점을 맞추고 있다. 클라이언트는 일반적으로 웹 서비스를 위해 사용하는 TCP의 80 포트로 요청을 보내게 된다. 이 요청은 interrupt part의 TCP Port Interceptor가 가로채어 커널가속기가 열어놓은 TCP 포트로 전달하게 된다. 이 과정으로 커널가속기는 모든 클라이언트의 요청을 가져올 수 있게 되고, 이에 따른 적절한 해석 및 동작을 수행한 후에 웹 서버가 열어놓은 TCP 포트를 통해 클라이언트에 응답을 하게 된다. 일부의 경우 커널가속기가 처리하지 못 할 요청은 웹 서버에게 넘겨 주지만 응답은 역시 커널가속기를 통해서 하게 된다.

이렇게 interrupt part에서 간단히 TCP 정보를 해석하여 처리하는 구조는 클라이언트와 사용자 프로그램

부분의 웹 서버 모두 설정의 변경 없이 커널가속기가 Linux 커널의 모듈로서 삽입될 수 있게 하며, 클라이언트의 요청을 User part까지 전달하지 않고 Kernel part에서 처리함으로써 성능 및 속도의 향상을 이룰 수 있도록 한다.

2.3.3 캐싱구조

클라이언트로부터 요청이 들어오면 커널가속기는 응답할 데이터가 메모리에 저장되어야 할 것인지를 판단하고, 만일 저장 가능하다면 같은 데이터가 이미 저장되어 관리되고 있는지를 살펴본다. 저장되어 있는 경우라면 바로 읽어 들여 응답을 하고, 저장되어 있지 않으면 디스크에 저장되어 있는 데이터를 이 후 사용할 네트워크의 특성에 알맞은 형태로 변환하고 필요한 항목들을 미리 추가하여 메모리에 저장한 후 응답을 하게 된다.

이와 같은 과정은 처음 한 번의 경우는 디스크를 접근하게 되지만 클라이언트의 요청이 진행됨에 따라 동일한 요청이 많아지고 결국은 대부분의 경우 저장된 메모리를 사용하게 되며 속도의 지속적인 향상이 이루어질 수 있다.

2.3.4 HTTP 1.1 지원

HTTP 1.1 프로토콜의 핵심적인 변화요소는 persistent connection 및 pipelining requests 기법이다. 기존 HTTP 1.0까지의 프로토콜은 하나의 TCP 커넥션에 하나의 HTTP 요청을 보냄으로써 동작하는데, 클라이언트가 빈번한 요청(html, jpg, gif 등등)을 보내는 경우, 실제의 데이터 송수신보다 TCP 커넥션을 맺는데 서버 및 네트워크의 자원과 시간을 많이 낭비하여, 효율적인 네트워크 대역폭을 조정하는데 어려움이 있었다. 하지만 HTTP 1.1 이후부터는 HTTP 프로토콜의 헤더를 이용하여, 하나의 커넥션이 끝난 이후에도, 서버쪽의 socket을 종료하지 않고, 같은 클라이언트의 다음 요청을 같은 socket으로 처리할 수 있게 되었다. 즉 이로 인해 실제적으로 TCP 커넥션에서의 초기화 과정을 생략할 수 있고, 시스템의 자원 낭비 및 네트워크의 시간지연요소를 줄일 수 있다. 또한 이러한 persistent connection을 이용하여, 클라이언트의 요청을 pipeline을 이루게 함으로써 하나의 TCP 커넥션에 여러 개의 요청을 보내고, 서버는 이를 해석하여, 적절하게 응답해 줄 수 있게 되어, 전체적으로 TCP 커넥션에 걸리는 시간과 대역폭을 절약할 수 있다.

2.3.5 CGI 지원

커널가속기는 CGI API가 구현되어 동적인 데이터를 처리할 수 있다. 일반적인 경우 사용자 프로그램 부분의 웹 서버가 클라이언트의 요청을 해석하여 사용자 환경에서 CGI 데이터를 처리할 수 있도록 하지만 이러한 과정을 모두 커널에서 수행함으로써 동적인 데이터 중 CGI의 처리 속도를 향상시켰다.

2.4 시뮬레이션 결과(hit수/sec)

(a) 초당처리요청수의 경우 apache만으로 웹 서비스를 하였을 경우, 약 초당 1000개의 요청을 수행하였으며, 커널가속기를 탑재하여, 웹가속 및 캐싱 서비스를 하였을 경우, 약 초당 5000개의 요청을 수행하였다.(그림 2)

(b) Throughput(Mbytes/sec)의 경우 apache만으로 웹서비스를 하였을 경우, 약 초당 5(Megabytes/sec)의 throughput을, 커널가속기를 탑재하여, 웹가속 및 캐싱 서비스를 하였을 경우, 약 초당 30(MegaBytes/sec)의 결과를 내었다.(그림 3)

본 커널가속기의 테스트는 zdnet의 Webbench 3.0을 사용한 것이며, 각각 giga bytes lan 환경에서 총 24대의 test client 서버를 통해 이루어졌고 초당처리요청수와 throughput의 경우 각각 500%, 600%정도의 성능이 향상되었다.

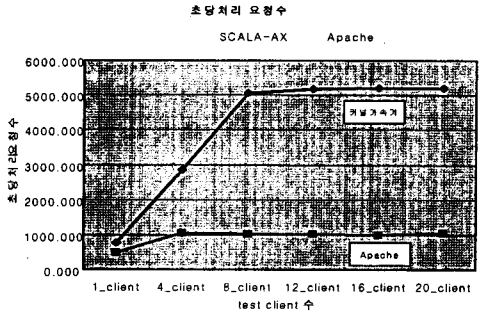

그림 2. 클라이언트의 변화에 따른 초당처리요청수

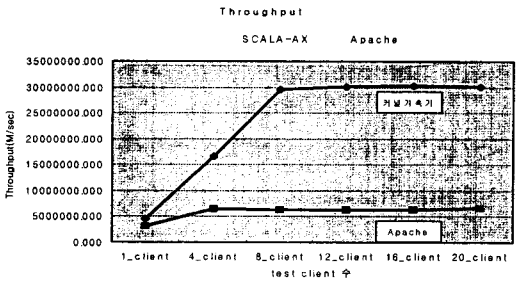

그림 3. 클라이언트의 변화에 따른 Throughput

3. 결 론

지금까지 웹가속을 위한 여러 가지 방법과 커널가속기의 구현을 위한 여러 가지 내용들을 살펴보았다. 시뮬레이션의 결과에서도 알 수 있듯이 속도면에서 월등한 결과가 나옴을 알 수 있다. 커널가속기는 웹서버안에 구현하여 따로 비용을 소비하지 않고 캐시서버와 같은 속도 향상의 결과를 가져옴을 알 수 있었다. 그리고 캐시서버와 연동하여 각각 사용하였을 경우보다 더 나은 결과를 얻을 수도 있을 것이다. 본론에서도 언급하였듯이 커널가속기는 정적페이지뿐만 아니라 동적페이지도 어느정도 처리할 수 있지만 점점 더 다양해지고 복잡해지는 동적페이지를 완벽히 처리하는 데에는 아직 약간의 문제가 있다. 물론 커널에서 처리할 수 없는 내용은 응용프로그램으로 처리를 넘기는 방법을 취하고 있다. 커널 레벨에 있기 때문에 필요한 자원에 쉽게 접근하여 사용할 수 있다는 장점이 있는 반면, 잘못 사용시에는 전체 시스템에 큰 타격을 줄수도 있다. 이점에 유의하여 앞에서 언급한 웹가속을 위한 여러 가지 기술들을 계속해서 커널레벨에 추가하는 연구가 계속되어야 할 것이다.

[참 고 문 헌]

[1] Barish, G; Obraczke, K, "World Wide Web caching:trends and techniques", IEEE Communications Magazine, Volume: 38 Issue: 5, Page(s): 178-184, May 2000

[2] Levy-Abegnoli, E; Iyengar, A; Junehwa Song; Dias, D, "Design and Performance of a Web Server Accelerator", INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Volume: 1, Page(s) 135-143, 1999

[3] Daniel P. Bovet; Marco Cesati, "Understanding the Linux Kernel", O'REILLY, January 2001

[4] Michael Beck 외 5명, "Linux Kernel Internals", Addison-Wesley, Page(s) 227-278, 1997

[5] 이경희 외3명, "인터넷 Speed Up! 웹 캐시", 프로그램세계, 4월호, pp 180-191, 2001