TUX(Threaded linUX) 웹서버 분석을 통한 웹서버 성능향상 방안

구본준, 박종규, 이상문, 김학배
연세대학교 전기전자공학과

A Webserver Performance Improving Scheme Based on Analysis of TUX(Threaded linUX) Webserver

Bonjun Koo, Jongyu Park, Sangmoon Lee, and Hagbae Kim
Dept. of Electric and Electronic Engineering, Yonsei University

Abstract - 인터넷 인구와 데이터량의 증가로 인해 초고속 정보인프라의 구축에도 불구하고 사용자들의 서비스에 대한 만족도는 여전히 충분하지 않다. 따라서 인터넷에서 고성능과 고가용성을 보장하는 것은 매우 중요한 이슈가 되고 있고, 이러한 욕구를 충족하기 위한 웹서버 가속기, 여분의 하드웨어, 로드 밸런싱, 동적 데이터의 효율적 관리 등과 같은 웹서비스의 많은 기법들이 활발히 연구되고 있다[1]. 이러한 웹서비스 관련 인프라 기법들 중에서 리눅스 운영체제의 커널단에서 실행되면서 고성능 웹서비스를 할 수 있도록 구현한 TUX (Threaded linUX) 웹서버의 분석을 통한 웹서버 성능향상 방안에 대해 고찰하여 성능 및 안정성을 개선 할 수 있는 시스템 설계 및 구현 방안을 제시한다.

1. 서 론

인터넷 관련 산업은 초고속 정보통신망 구축 등 정부의 지속적인 관심과 투자로 세계적 수준의 정보인프라가 구축되고 있으며, 또한 인터넷을 이용하는 사용자 수, 웹 서비스를 공급하는 사이트 수, 네트웍상의 데이터량도 최근 들어 폭발적으로 증가하고 있다. 웹 상에서 사용자들이 체감하는 큰 지연과 늦은 응답속도는 사용자들에게 심각한 불만을 주고 있다. 사실상 전송속도 문제는 사용자들에게 가장 직접적으로 느껴지는 것이며, 시간지연에 대한 사용자들의 지루함이나 불만은 민감한 주제이다. 이러한 늦은 응답 속도에 대한 원인은 사용하고 있는 통신망의 전송능력 한계 및 수많은 사용자들에 의한 트래픽의 과다 등 네트웍 분야의 문제도 있을 수 있지만 웹서비스를 하고있는 웹서버 쪽의 시스템 성능에도 큰 영향을 받는다. 이처럼 웹에서의 성능 문제는 웹 사용자가 늘어날수록 더욱 민감하고 중요한 문제가 되고 있다. 따라서 최근 들어 웹서버에 대한 관심이 증대되고 있으며 여기에 대한 연구가 활발히 진행되고 있다[2]. 웹의 성능과 고가용성은 많은 수의 요청을 받는 웹사이트에서 더욱 중요하다. 단순히 웹의 성능을 높이기 위해서 새로운 하드웨어를 추가하여 클러스터링을 구현하거나 로드밸런서와 리얼서버의 중간에 캐싱서버를 배치시킴으로써 자주 요청되는 페이지를 캐싱하여 서비스하는 것도 웹의 성능을 높이는 방법이 될 수도 있을 것이다[3][4]. 그러나 본 연구에서는 웹서버 자체에 관점을 두고 일반적인 웹서버에 대해서 알아보고 웹서버의 성능향상과 확장성을 위해 웹서버 기능을 운영체제 안에 포함시킨 TUX 웹서버가 사용하고 있는 기술에 대해서 자세히 분석하여 시스템 성능 및 안정성을 개선시킬 수 있는 방안을 제시하고자 한다.

2. 본 론

2.1 웹서버의 개념

웹서버는 네트웍에서 네스케이프나 익스플로러 같은 웹 브라우져에게 콘텐츠를 제공하는 일을 하는 어플리케이션이다. 이것은 웹서버가 http://www.x.com/index.html 형식의 웹페이지 요청을 받았을 때 서버의 로컬 파일에 URL(Uniform Resource Locator)을 매핑시킨다. 여기서 index.html은 호스트 파일시스템에 존재하는 파일이다. 그런 다음 웹서버는 디스크에서 파일을 로드하고 네트웍을 통해 유저의 웹브라우져에 데이터를 전송하게 된다. 이것의 흐름도는 [그림1]과 같다

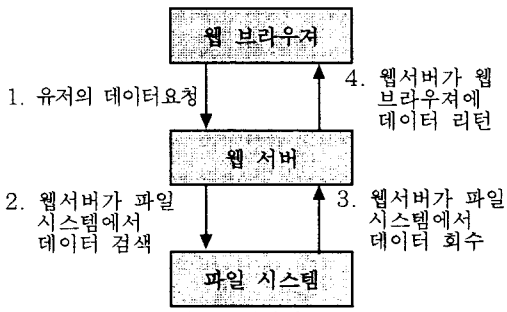

그림 1. 웹서비스 흐름도

위의 상황은 매우 간단한 정적 콘텐츠 전송에 기반한 원리일 뿐이다. 그러나 이것을 동적 콘텐츠에 확장하면 가장 일반적이면서 대표적으로 사용되는 것이 CGI (common Gateway Interface)를 이용하는 것이다. [그림2]는 웹브라우져가 동적 페이지를 요청했을 때 CGI 프로그램을 이용하여 동적으로 생성된 페이지를 서비스하는 흐름을 보여준다.

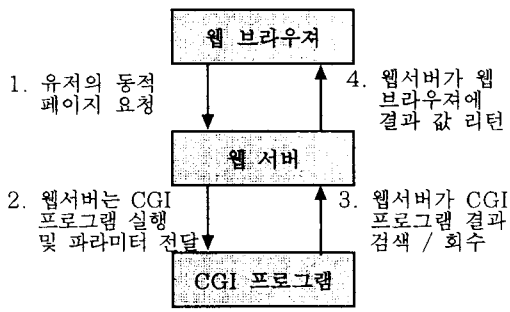

그림2. 동적 페이지 웹서비스 흐름도

웹서버는 브라우져로부터 네트웍 커넥션을 보장하고, 디스크에서 콘텐츠를 검색할 수 있어야 하며, 로컬 CGI 프로그램을 실행시킬 수 있어야 하고, 클라이언트에 데이터를 전송할 수 있어야 하고, 가능한 한 속도가 빠르면 좋은 웹서버라고 할 수 있을 것이다[5].

현재 시중에 나와 있는 웹서버의 종류는 Alibaba, AOLserver, Apache, GoAhead, iPlanet Web Server Enterprise Edition, iServer, Microsoft IIS v4.0, Microsoft IIS v5.0, Microsoft PWS,

Microsoft Site Server, NCSA HTTPd, Netscape Enterprise, Netscape FastTrack, OmniHTTPd Pro, Roxen Challenger, Roxen Web Server, Sambar Server, Zeus 등 수 많은 종류가 있다[2]. 이중에 대표적인 것으로 아파치와 IIS를 들 수 있는데 이것들은 모두 어플리케이션 레벨에서 실행되는 것으로서 각 계층으로 데이터를 전송할 때 데이터 복사의 반복과정을 통해 쓰레드를 처리하기 때문에 많은 오버헤드가 발생하여 속도 면에서 성능 한계를 가지고 있다[6][7][8]. 그러나 다음에 설명할 TUX는 커널레벨에서 동작하는 프로그램이므로 속도 면에서 우수한 성능을 보장한다.

2.2 TUX(Threaded linUX) 웹서버

TUX[9] 웹서버는 Threaded linUX의 약자로서 리눅스 커널 개발자인 Monlar에 의해서 개발되었으며 리눅스 운영체제의 커널단에 구현한 고성능 HTTP서버이다. TUX는 하나의 HTTP 프로토콜 계층에서 웹서버의 오브젝트를 캐시하기 위해 Linux 커널에 포함되었고 성능 향상을 위해 SMP환경에서 복잡한 비동기 요청을 처리하기 위한 멀티 쓰레딩, 커널의 네트워킹 계층과 웹서버의 직접 연결, 정적 콘텐츠 전송속도를 높이기 위해 커널 네트워킹 계층의 응답을 캐시, 유저 레벨에서 복잡한 동적 콘텐츠를 만들기 위해 유저 레벨 어플리케이션을 위한 다양한 인터페이스 제공 등의 일반적인 기술을 사용한다. 데이터의 복사와 checksum을 만드는데 발생하는 부하를 줄이기 위해, zero-copy 기술을 통해 커널 주소공간에 캐시 될 수 있게 하였고, CGI 스크립트는 커널에 의해 바로 호출 될 수 있으며, 반환 값은 네트웍 계층으로 바로 전송된다. 전체적인 구성은 [그림3]과 같다

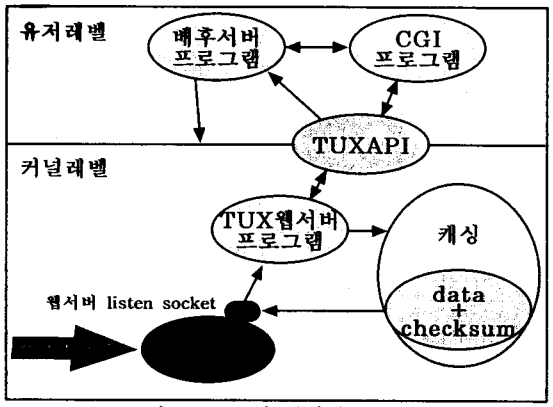

그림3. TUX의 데이터 처리과정

2.2.1 TUX 웹서버의 기본적인 특징

TUX는 서버의 웹문서 루트에 마운트된 ext2, NTFS, DOS FAT등 로컬 파일 시스템의 파일들을 전송하는 역할을 한다. TUXAPI라는 간단하고 안전한 프로그래밍 인터페이스를 통해서 커널 모듈에서 콘텐츠를 동적으로 생성할 수 있다. 이 인터페이스들은 C 언어로 작성된 커널 모듈이 커널 환경 안에서 쉽게 파일 등의 리소스에 접근할 수 있도록 한다. TUX는 TUXAPI를 통해서 외부 CGI를 동작시킨다. 외부 CGI는 커널 밖에서 동적 콘텐츠를 생성하기 위해 기존의 웹서버에서 보통 fork, exec에 의해 시작되었던 스크립트를 말하는 것이다. CGI 스크립트는 사용자 레벨 CGI라 불리는 사용자 레벨의 조작을 통해서도 시작될 수 있다. 이것은 새로 만들어진 tux() 시스템 콜에 의해서 사용자 프로세스가 커널 웹서버와 상호작용 할 수 있게 한다.

프로토콜은 HTTP 버전 1.0 과 1.1의 기본 기능을 지원하고 khttpd처럼 응용레벨에서 동작하는 아파치 등의 배후 서버가 필요하지 않다. 하지만 선택적으로 TUX가 인식하지 못하는 HTTP 헤더 정보를 갖는 요청에 대해 배후 서버가 응답할 수 있도록 하였다. 그러나 배후서버는 TUX와는 완전히 독립된 작동을 하므로 빠른 소켓 전환 메커니즘에 의해 이 과정을 인식하지 못한다.

TUX의 설정은 커널을 만드는 과정에서 시작하고 커널 컴파일 옵션에서 "Networking options" 안의 TCP/IP Networking"에 TUX를 위한 새로운 옵션들을 조작하여 설정한다

서버의 로그는 바이너리 형식으로 만들어진다. TUX 개발자는 이 방법이 ASCII 형식으로 하는 것보다 I/O 대역폭이나 디스크 공간을 적게 차지하는 것으로 생각하였다. 만약 로그 파일의 해석이 필요한 경우에는 별도의 유틸리티를 사용하면 W3C 형식의 포맷으로 전환할 수 있다. 각각의 로그 기록은 http_req_t 구조체의 모든 정보를 담고 있으며, 각 CPU의 쓰레드는 로그 데이터를 매 초 혹은 95% 이상 찼을 때 지움으로써 과도한 데이터의 축적을 방지한다.

커널의 HTTP 계층과 유저 레벨 사이의 전환은 tux() 시스템 콜에 의해서 이루어진다. 사용자 프로세스가 TUX 웹서버를 동작시키고, tux() 시스템 콜은 유저 레벨 작업과 커널 네트웍 사이의 인터페이스로 사용되어 사용자 프로세스 자신을 웹서비스 쓰레드로 인식하도록 하고, 커널로부터의 작업을 받아들인다.

2.2.2 정적 콘텐츠의 가속

외부로 나가는 네트웍 패킷의 데이터를 복사하고 checksum을 계산하는 작업은 작은 크기의 파일들을 계속적으로 전송해야 하는 웹서버에 있어 가장 큰 부하량을 차지한다. 이 작업들에 의한 부하를 줄이기 위한 전형적인 방법은 앞으로 들어올 클라이언트의 요청에 대비해 이미 한 번 이상 서비스된 오브젝트를 캐싱하는 것이다. 캐시된 checksum 값을 효율적으로 유지하기 위해서 데이터는 파일 시스템이나 특정한 어플리케이션 레벨이 아니라 커널 네트웍 버퍼에 캐시한다. TUX는 캐시된 데이터와 코어 내에서 아이노드로 나타나는 커널 데이터 구조체를 참조하여 이 기능을 수행한다.

리눅스 커널은 네트웍 디바이스를 통해 입출력되는 데이터를 sk_buff 구조체를 이용해 관리한다. TUX는 sk_buff의 동적 변화를 관리하는 리눅스 네트웍 계층의 기능을 강화하여, 버퍼 프레그먼트를 관리하고 sk_buff를 사용한 I/O의 분산/집적을 위해 skb_frag_t라는 새로운 데이터 구조체를 이용한다.

```
struct skb_frag_struct {
        unsigned int csum;
        int size;
        struct page *page;
        int page_offset;
        void (*frag_done)(struct sk_buff *skb,
skb_frag_t *frag;
        boid *data;
        void *private;   };
```

그림 4. 버퍼 프래그먼트 관리를 위한 데이터 형

모든 sk_buff는 네 개까지의 버퍼 조각을 포인팅하고 있다. 각각의 조각은 시스템 페이지 크기를 가질 수 있고, 각 조각의 checksum은 독립적으로 관리되므로 서버는 각각의 조각에 대한 checksum을 새로 계산할 필요 없이 사용할 수 있다. 다음은 캐시된 checksum의 데이터 구조이다.

```
struct scumcache_struct{
        int size;
```

```
int packet_size;
skb_frag_t *array;  };
```
그림 5. checksum 캐시 오브젝트

- array : 프래그먼트의 배열
- size : 프래그먼트 배열 안에 있는 패킷 수

scumcache_struct 구조체는 보통 sk_buff와 달리 임의의 수의 버퍼 프래그먼트를 포인팅 할 수 있다. 따라서 임의의 복잡한 네트웍 버퍼의 길이와 내용을 관리하는 것이 가능하다. 클라이언트의 요청에 대해 응답할 때 서버는 이전에 캐시된 부분들을 모으고 checksum을 더하면 된다. 이것은 분산/집합 I/O의 추가적인 복사 작업을 미연에 방지할 수 있게 한다.

코어 안에 있는 아이노드를 checksum 캐시에 캐시된 데이터에 연결하기 위해서 lookup_urlo()에서 URL 오브젝트가 생성된다. 이것이 캐시된 데이터에 URL을 링킹한다. TUX는 디렉토리 엔트리 캐시를 검색하여 URL 오브젝트를 얻는다. 디렉토리 엔트리 캐시는 리눅스가 동작하는 하드웨어의 물리 메모리 사이즈에 의해 결정되는 해시 테이블에 의해 관리되는데, 일반적으로 캐시 안의 엔트리는 재사용 되지 않고, 프리 메모리가 존재하는 한 캐시 크기가 커진다. 메모리가 부족하게 되면 페이지 얼로케이터는 캐시의 디렉토리 엔트리를 줄이는 작업을 시작한다. 이 작업으로 코어 안의 아이노드와 페이지 캐시 안의 일련의 데이터도 제거할 수 있다.

요약하면 TUX는 커널의 네트웍 계층에서 checksum과 함께 응답의 일부 또는 전체를 캐시할 수 있도록 하는 새로운 데이터 구조체들을 만들고, 캐시된 웹 오브젝트들은 커널의 LRU(Least Recently Used) 알고리즘을 통해 디렉토리 엔트리 캐시의 일부분처럼 관리된다.

2.2.3 동적 콘텐츠의 가속

TUX는 두 개의 새로운 인터페이스로 동적 콘텐츠의 생성을 가속한다. tux() 시스템 콜에 의한 사용자 레벨의 인터페이스와 커널 레벨의 인터페이스가 있다.

TUX는 커널 모듈들이 동적 콘텐츠를 생성하는 것을 지원하는 TUXAPI를 제공하는데, 이 동적 API는 간단하고 보안은 중요하지 않으면서 자주 사용되는 요청을 위한 것이고, 복잡하고 느린 요청들은 사용자 공간에서 처리된다. 이런 모듈들은 "**http**://server.your.domain/**module**?**argument**" 와 같은 특별한 URL에 의해 호출된다. 여기에서 "**module**" 은 불려질 동적 모듈의 고유 이름이고, "**argument**"는 서버에서 모듈로 전달되는 text 형식의 인자를 나타낸다.

모든 HTTP 동적 모듈은 엔트리 포인트와 모듈 속성들을 정의하는 tcapi 템플릿을 정의한다. 템플릿은 다음의 형식을 지닌다.

```
struct tcapi_template_s{
        char * vfs_name;
        char *version;
        int (*query) (http_req_t *req);
        int (*send_reply) (http_req_t *req);
        int (*log) (http_req_t *req, char*long_buffer);
        void (*finish) (http_req_t *req);  };
```
그림 6. 동적 모듈 API template

- vfs_name : 모듈 이름
- version : 모듈에 의해 지원되는 인터페이스의 버전
- 다음은 모듈에 구현된 가상 함수들이다
  - query : 클라이언트로부터의 새로운 요청을 처리하기 위해 서버로부터 호출
  - http_req_t : 모듈이 반환할 파일 이름. 이것은 요청한 클라이언트에게 전송
  - send_reply : 서버는 이것에게 응답을 만들게 함
  - log : 특정한 모듈에 따른 로그 메시지 생성
  - finish : 서버가 커넥션을 끊을 때 호출

인터페이스는 모듈 프로그래머가 사용하기 쉽도록 몇 가지 다른 구조체를 제공하는데, 이것은 파일, URIs, 페이지, 파일 시스템 디렉토리 엔트리, 서버의 문서 경로 등을 나타내는 디스크립터들이다. 모듈 프로그래밍을 간략화하기 위해 제한된 기능의 시스템 콜과 유사한 API가 제공된다. 이것은 모듈이 안전하게 서버의 다큐먼트 루트에서 파일들의 열기, 닫기, 검색하기, 읽기, 쓰기 등을 할 수 있게 하고, 버퍼 혹은 파일을 직접 클라이언트에게 전송, 파일 크기와 수정 시간 변경, 프리 heap 메모리 할당/해제, 새 쓰레드 실행, 클라이언트의 IP 주소 회수 등의 작업을 하도록 한다.

요약하면 TUXAPI는 커널 모듈이 동적 콘텐츠를 생성하도록 하는 커널 모듈 프로그래밍 인터페이스이다. 이것은 fork, exec, 파일관리, 웹 오브젝트 캐시를 비동기적으로 존재하게 하거나 네트웍 커넥션 관리 등을 수행하는 작은 시스템 콜 비슷한 인터페이스를 제공한다. TUXAPI 인터페이스를 사용하는 커널 모듈은 외부 CGI 스크립트를 호출할 수 있게 한다. 따라서 이것을 통해서 CGI를 실행시켜 동적 콘텐츠를 가속시킨다.

3. 결 론

이상에서 살펴본 바와 같이 TUX 웹서버는 캐싱서버와 같은 다른 하드웨어의 추가 설치 없이 리눅스 운영체제의 커널단에서 동작하면서 정적, 동적 데이터 모두를 안전하고 효율적으로 처리하고, 자주 사용되는 웹서버 기능을 커널의 네트워킹 스택에 좀더 가까이 배치시켜서 성능과 확장성 면에서 상당히 발전된 기술을 구현하였다. TUX는 네트웍 계층에서 캐시된 데이터를 유지하여 들어오는 네트웍 이벤트들을 직접 웹서비스함으로써 기존의 웹서버보다 더 빠르게 클라이언트에게 응답할 수 있도록 하였으며, 웹서버가 HTML 파일을 사용자에게 보내기 직전에 데이터 값을 포함시켜 보내주는 SSI (server-side include), 커널 내에 있는 동적 컨텐츠를 위한 API 등과 같은 발전된 특징을 보여주고 있다. 이와 같은 TUX의 발전된 기능에 기존의 웹 성능향상을 위해 많이 사용되는 로드밸런싱에 의한 클러스터링의 구현이나 캐싱서버와 같은 기능을 접목시킨다면 더 큰 성능향상을 기대 할 수 있을 것이다. TUX의 향후 과제로 프락시 캐시와 호환성 문제, 유저레벨 인터페이스에서의 보안 관련 문제 등을 해결하면 더욱 향상된 성능과 안정된 시스템으로 웹서비스를 할 수 있을 것이다.

[참 고 문 헌]

[1] Arun Iyengar, Jim Challenger, Daniel Dias, Paul Dantzig, "High-performance web site design techniques", IEEE, march · april, pp17-26, 2000

[2] http://webcampare.internet.com/webservers.html

[3] Junehwa Song, "Design alternatives for scalable webserver accelerators", IEEE, 0-7803-6418-X, pp184-192, 2000

[4] A.Mourad and H. Liu, "Scalable Webserver Architecture", IEEE, Second Symposium, pp12-16, 1997

[5] http://webcampare.internet.com/webbasics

[6] http://httpd.apacche.org/dosc

[7] http://www.microsoft.com/korea/windows2000/guide/server/overview/default.asp

[8] http://www.microsoft.com/korea/ntserver/nts/default.asp

[9] ftp://ftp.redhat.com/pub/redhat/tux/