웹 프록시 캐쉬에 대한 운영체제 지원 성능의 측정과 분석

(Measurement and Analysis on Operating System Supports for Web Proxy Cache)

백 윤 철 † 추 연 준 ††
(Yuncheol Baek) (Yeonjun Chu)

요 약 오픈 소스 웹 관련 소프트웨어는 주로 UNIX 기반 운영체제 상에서 작동하며, 사용하는 운영체제에 따라 웹 시스템 전체의 성능에 영향을 미칠 수 있다. 본 논문에서는 운영체제가 웹 서비스를 얼마나 잘 지원하는 가를 분석하기 위해 소스 코드 수준의 시간 측정 도구를 만들고, 오픈 소스 웹 프록시 캐쉬 소프트웨어인 Squid가 요청하는 운영체제 시스템 호출의 수행 시간을 측정하였다. Squid는 2.4.STABLE1을 사용하였으며, 기반 운영체제로는 역시 오픈 소스인 Linux 2.4.2와 Solaris 8을 사용하여 시스템 서비스 성능 차이를 비교하였다. 이 결과 전체적으로 Squid를 지원하는 데에는 Linux 2.4.2가 Solaris 8보다 나은 것으로 나타났다. 실험 결과는 오픈 소스 소프트웨어를 이용하여 웹 서비스를 구축하는 경우, 소프트웨어 선택을 위한 판단 자료로 사용될 수 있고, 웹 관련 소프트웨어를 지원하는 운영체제 서비스의 성능 개선을 위한 자료로도 유용하게 사용될 수 있다.

키워드 : 웹 프록시 캐쉬, 성능 평가, 운영 체제, 리눅스, 솔라리스, 오픈 소스

Abstract Open source web service softwares are usually running on UNIX-based operating systems. Choosing an operating system can affect web system performance. In this paper, we describe a source - code-level time measurement tool and measure a service time of each system call that Squid - open source web proxy cache - makes. We run Squid 2.4.STABLE1 on Linux 2.4.2 and Solaris 8, and we compare the measured time. As a result, Linux 2.4.2 performs better than Solaris 8. It can be used as a guide for selecting system software on building web service using open source software, and also be used as a basis for enhancing the operating system performance of supporting web service software.

Key words : web proxy cache, performance evaluation, operating systems, Squid, Linux, Solaris, open source

1. 서 론

월드 와이드 웹(WWW: World Wide Web)은 가장 대중화되었으며 전체 인터넷 트래픽 중 가장 많은 비중을 차지하고 있는 인터넷 서비스이다. 지금 이 시간에도 WWW 클라이언트와 서버의 수가 늘어나고 있을 정도로 노드의 증가율이 엄청나고 서비스를 이용하는 인구도 기하급수적으로 늘어났다. WWW 서비스를 위해서는 기본적으로 웹 문서를 제공하는 웹 서버가 있어야 하고 이를 검색하는 클라이언트 소프트웨어가 있어야 한다. 웹 서비스는 한 서버의 같은 웹 문서에 대해서 여러 클라이언트가 요청을 하고 서버는 이를 반복적으로 서비스를 해 준다는 트래픽 특성이 있다. 또 많은 경우에 한 클라이언트가 똑같은 문서를 반복적으로 요청하기도 한다. 이러한 형태의 통신 방식은 인기가 있는 사이트에 대한 요청의 폭주를 낳게 하며 인터넷 대역폭의 낭비를 가져오기도 한다. 이러한 WWW 트래픽으로 야기되는 인터넷의 병목 현상을 타개하기 위해 대두된 해결책이 웹 문서의 캐싱(caching)이다. 이것은 원격 사이

· 본 연구는 2002학년도 상명대학교 자연과학연구소 연구비 지원으로 이루어졌음.
† 정 회 원 : 상명대학교 소프트웨어학부 교수
bycker@sangmyung.ac.kr
†† 비 회 원 : 생크정보통신 연구원
yjchu@oslab.sangmyung.ac.kr
논문접수 : 2001년 10월 23일
심사완료 : 2002년 4월 16일

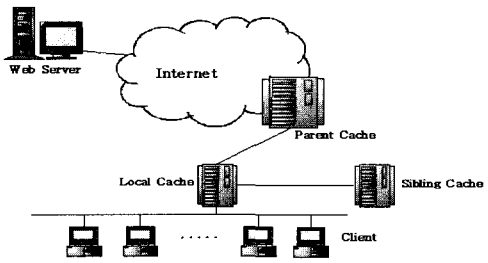

그림 1 WWW 시스템

트에 있는 웹 문서에 대한 반복적인 요구를 흡수하기 위해 인터넷의 각 요소마다 프록시 캐쉬를 배치하여 웹 문서를 지리적으로 가까운 곳에 위치하도록 하는 방법이다[1]. 그림 1은 웹 서버와 클라이언트, 웹 프록시 캐쉬가 WWW을 구성하고 있는 모습을 보여준다.

웹 서버와 웹 프록시 캐쉬 서버로 사용되는 소프트웨어는 대단히 종류가 많으며 크게 오픈 소스 소프트웨어와 소스가 공개되지 않는 상용(commercial 또는 proprietary) 소프트웨어로 나누어진다. 오픈 소스 소프트웨어 중 대표적인 것이 Apache[2], Squid[3] 등이며 상용 소프트웨어 - 대부분 하드웨어와 함께 설계, 구현, 판매되므로 상용 시스템이라 함이 더 적당함 - 로는 Microsoft IIS[4], Inktomi[5], Akamai[6], Cisco[7] 등이 대표적이다. 그러나 Netcraft[8]에 의하면 세계적으로 조사된 3,000만여 개의 웹 서버 중 약 60%가 오픈 소스 소프트웨어인 Apache를 쓰고 있는 것으로 나타났다. 웹 프록시 캐쉬의 경우에는 Squid가 가장 널리 사용되는 소프트웨어이며 오픈 소스 소프트웨어로 구축되는 프록시 캐쉬의 거의 대부분이 Squid를 사용한다고 볼 수 있다. 이러한 오픈 소스 웹 소프트웨어들은 대부분 Linux, Solaris 등 UNIX기반의 운영체제 상에서 작동하고 있다.

본 논문에서는 대표적인 웹 프록시 캐쉬 소프트웨어인 Squid에 대한 운영체제의 지원 성능을 분석하였다. Squid는 웹 서버와 클라이언트 사이에 커넥션을 맺기 위해 운영체제의 네트워크 관련 시스템 호출을 사용하며 최근에 참조된 웹 오브젝트를 읽거나 저장하기 위해 파일 시스템 관련 시스템 호출을 빈번하게 사용하는 소프트웨어이다.

성능 측정 실험은 다음과 같이 구성된다.

① 프로그램 모듈의 수행시간을 측정하는 도구를 만든다.

② Squid 소스에 시간 측정 도구에서 제공하는 API를 포함하도록 패치(patch)한다.

③ 웹 프록시 캐쉬의 벤치마킹 도구로 널리 이용되는 웹 트래픽 생성기인 Web Polygraph[9]를 통해 만들어진 요청을 패치된 Squid로 하여금 처리하게 한다.

④ 수행 시간 측정 도구를 통해 나온 로그를 바탕으로 운영체제의 시스템 호출에 걸리는 시간을 계산한다.

본 논문은 다음과 같이 구성된다. 2장에서는 웹 시스템의 성능과 관련한 연구를 살펴보고, 3장에서는 운영체제의 서비스와 관련이 있는 Squid의 작동에 관해 기술하고, 4장에서는 Squid에 패치하여 시간을 측정할 목적으로 작성된 성능 측정 도구를 기술하며, 5장에서는 실험 결과와 분석을 제시하고, 6장에서 결론을 맺는다.

2. 관련연구

웹과 관련한 시스템 소프트웨어 - 웹 서버, 웹 프록시 캐쉬 등의 성능과 관련하여 많은 연구가 진행되었다. 먼저 성능 측정 방법에 대한 것들로 Banga 등은 웹 서버의 성능 측정을 위해 보다 실제적인 HTTP 요청을 만들어 내는 방법을 연구하였다[10]. Barford 등도 웹 워크로드를 만들어 내는 SURGE라는 도구를 만들고 로드의 증가에 따른 네트웍과 서버의 성능을 측정하였다[11]. 웹 캐쉬 프록시 서버의 응답 성능 측정을 위한 트래픽 생성 도구는 1장에서 언급한 Web Polygraph가 대표적이다. 웹 캐쉬 프록시는 클라이언트의 요청을 받아 필요한 오브젝트를 원격 서버에서 끌어오는 일을 하는 것이므로 Web Polygraph가 HTTP 요청을 발생시키는 클라이언트의 역할과 오브젝트를 공급하는 원격 웹 서버의 역할을 동시에 수행한다[9]. 위의 연구들을 토대로 웹 서비스 관련 서버들의 성능을 측정 또는 모델링하고 메커니즘 측면의 개선 사항을 제시하는 연구들이 있으며 [12][13][14] 등이 여기에 속한다. 웹 프록시 캐쉬에 있어서는 캐쉬의 교체 정책이 하나의 큰 연구 주제이며 교체 방법들 간의 성능 비교 또는 새로운 교체 방법 제안과 성능 평가에 관련한 연구가 상당히 많이 진행되었다. 그 중 Squid에서 제공하는 캐쉬 교체 정책들의 성능은 [15]에서 잘 비교 설명되었다.

웹 관련 서버 자체뿐 아니라 이를 지원하는 운영체제와의 상관 관계 속에서 성능의 개선 요소를 찾는 연구도 있는데 [16]에서는 고성능의 웹 서버를 구현하기 위해 제안되어 있는 방법들 - 효율적인 socket 관련 함수를 적용하는 것, 시스템 내에서 메모리간의 복사를 줄이는 것, TCP를 위반하지 않으며 연결당 오버헤드를 최소화하는 것 등에 대해 종합적인 실험을 하고 그 효용성을 논하고 있다. [17]도 그 중 하나로 고성능의 네트워크 서버를 위해 운영체제 수준에서 지원해야 할 점들을 언급하고 있다. [18]에서는 기존 UNIX 파일 시스템의 버퍼 캐쉬 메커니즘이 웹 프록시 캐쉬를 위해 적합치 않은 방법임을 시사한다. 본 연구도 이들과 맥을 같

이 하는 것으로 웹 프록시 캐쉬와 이를 지원하는 운영체제와 관계를 주요 관심사로 한다.

3. Squid의 작동 흐름

Squid는 Harvest 프로젝트의 산물로 웹 클라이언트에게 FTP, gopher, HTTP 데이타 오브젝트를 제공하는 웹 프록시 캐쉬이며 모든 클라이언트의 요청을 하나의 non-blocking, I/O driven 프로세스로 처리한다는 특징을 가지고 있다[3]. Squid는 소스가 공개되어있어 각종 UNIX 플랫폼 상에서 광범위하게 사용되고 있으며, 특히 미국 전역을 계층적 캐쉬로 엮는 NLANR (National Laboratory for Applied Network Research)의 IRCache 프로젝트에서도 사용되고 있다[19].

본 논문에서는 Squid 2.4.STABLE1 소스 코드를 가지고 클라이언트로부터 웹 오브젝트를 요청 받는 단계에서 시작하여 캐쉬된 웹 오브젝트가 있을 경우 (HIT인 경우) 파일 시스템을 접근하여 클라이언트에게 오브젝트를 보내거나, 없을 경우 (MISS인 경우) 웹 서버에 오브젝트를 요청한 후 받은 오브젝트를 클라이언트로 보내고 한편으로는 이것을 캐쉬하는 처리 과정 흐름을 분석하였으며, 이에 사용되는 운영체제의 시스템 호출을 그림 2와 같이

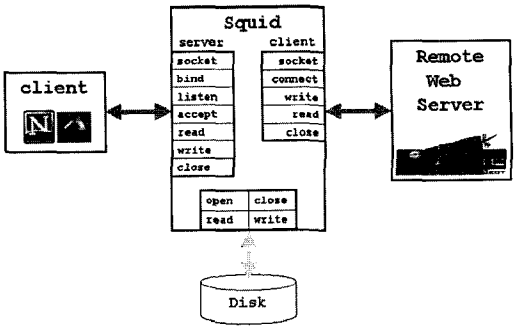

그림 2 Squid의 동작에 사용되는 운영체제의 시스템 호출

정리하였다. 그림 2에서 보는 것처럼 웹 프록시 캐쉬는 클라이언트의 요청에 대해서는 서버로 작동하고 원격 웹 서버에 대해서는 클라이언트로 작동한다. Squid의 운영체제 시스템 호출 사용은 다음과 같이 요약될 수 있다.

① Squid는 운영체제의 시스템 호출인 socket(), bind(), listen() 등을 사용하여 socket을 만들고 클라이언트의 요청을 기다린다.

② 클라이언트의 요청이 오면 이를 accept()하고 그 내용을 read()하여 URL을 추출한다.

표 1 Squid의 함수 호출

	sys call	Squid내 함수 호출
server 역할시	socket()	main()→mainInitialize()→severConnectionOpen()→clientHttpConnectionsOpen()→comm_open()→socket()
	bind()	main()→mainInitialize()→severConnectionOpen()→clientHttpConnectionsOpen()→comm_open()→commBind()→bind()
	listen()	main()→mainInitialize()→severConnectionOpen()→clientHttpConnectionsOpen()→comm_listen()→listen()
	accept()	main()→comm_poll()→comm_poll_http_incoming()→httpAccept()→comm_accept()→accept()
	read()	main()→comm_poll()→clientReadRequest()→read()
	write()	main()→comm_poll()→commHandleWrite()→write()
	close()	main()→comm_poll()→clientReadRequest()→comm_close()→close()
client 역할시	socket()	main()→comm_poll()→clientReadRequest()→clientAccessCheck()→aclCheck()→aclCheckCallback()→clientAccessCheckDone()→redirectStart()→clientProcessRequest()→clientProcessMiss()→fwdStart()→peerSelectCallback()→fwdConnectStart()→comm_open()→socket()
	connect()	main()→comm_poll()→commConnectHandle()→comm_connect_addr()→connect()
	write()	main()→comm_poll()→commHandleWrite()→write()
	read()	main()→comm_poll()→httpReadReply()→read()
	close()	main()→httpReadReply()→comm_close()→close()
file system 접근	open()	main()→comm_poll()→httpReadReply()→fwdComplete()→storeComplete()→storeSwapOut()→storeSwapOutStart()→storeCreate()→storeUfsCreate()→file_open()→open()
	read()	main()→comm_poll()→clientReadRequest()→clientAccessCheck()→aclCheck()→aclCheckCallback()→clientAccessCheckDone()→redirectStart()→clientProcessRequest()→storeClientCopy2()→storeClientFileRead()→storeReaddiskHandleRead()→read()
	write()	main()→comm_poll()→httpReadReply()→storeSwapOut()→storeSwapOutStart()→storeWrite()→storeUfsWrite()→diskHandleWrite()→write()
	close()	main()→comm_poll()→httpReadReply()→fwdComplete()→storeComplete()→storeSwapOut()→storeClose()→storeUfsIOCallback()→file_close()→close()

③ URL에 해당하는 오브젝트가 캐쉬에 있는 것이면 파일 시스템에 대한 open(), read(), write(), close()를 통해 server 측으로 가지고 온다.

④ 해당 오브젝트가 캐쉬에 없으면 client측의 socket(), connect() 등을 통해 네트워크 연결을 만들고 write(), read()를 통해 원격 웹 서버로부터 원하는 오브젝트를 가지고 온다.

⑤ 해당 오브젝트 혹은 오브젝트를 구할 수 없는 경우에는 에러 메시지를 write()를 통해 클라이언트에게 보내주고 socket을 close() 한다.

표 1은 우리가 관심이 있는 운영체제 시스템 호출이 나오게 되기까지 Squid 내부 함수 호출 과정을 보여준다.

4. 성능 측정 도구

성능 측정 도구를 사용하여 측정하고자 하는 것은 Squid가 동작하는 운영체제 상의 네트워크 관련 시스템 호출과 파일 입출력 관련 시스템 호출의 수행 시간이다.

Squid는 하나의 프로세스로 이루어져 있으며, 1024개의 파일 기술자(file descriptor)를 열어두고 이들의 상태를 주기적으로 폴링(polling)하는 방식으로 작동한다. 이들은 각각 네트워크 통신 또는 파일의 입출력을 위해서 사용된다. 파일의 입출력에 있어서는 전체 웹 오브젝트를 한꺼번에 읽거나 쓰는 것이 아니고 일부(4096 Btye)에 대해서만 작업을 하고 다른 개방된 파일 기술자들의 상태를 조사하기 위해 또 다시 폴링하게 된다. 이렇게 일부에 대해서만 읽거나 쓰게 함으로써 특정 요청이 Squid의 서비스를 독점하는 것을 막아준다. 따라서 각 파일 기술자마다 사용되는 함수를 구별하여 시간을 누적 측정할 수 있도록 아래와 같이 자료구조를 구현하였다.

```
struct PerFile {
        int fd;
        char *desc;
        char *name;
        struct timeval open;
        struct timeval read;
        struct timeval write;
        struct timeval close;
}SpendTime[1024];
```

위 자료구조를 사용하여 파일 관련 시스템 호출들이 수행되기 전에 시스템의 현재 시간을 얻어 저장한 후, 호출을 끝내고 돌아왔을 때 다시 시스템 시간을 얻어 그 차이를 구하였다. 이때 각 파일에 대해 사용되어진 함수에 따라 시간을 구별하여 측정하였으며 생성된 함수가 사용한 시간에 대한 정보는 파일을 닫을 때 출력하도록 하였다. 위에서 언급한 바와 같이 캐쉬된 파일을 읽거나 쓸 경우 Squid가 여러 번 동일한 함수를 호출하기 때문에 각 파일마다 누적하는 기능을 하도록 구현하였다. 파일 입출력 관련 시스템 호출 수행시간을 측정하기 위해서 아래와 같은 함수를 사용한다.

```
void setStartTime(int fd, char *desc, char *name)
        : 시스템 호출 전에 시스템 시간을 기록
void setEndTime(int fd, char *desc, char *name)
        : 파일 기술자, 호출의 종류 등을 구별하여 호출에 걸린 시간을 계산
```

파일 관련 시스템 호출 전에 시스템 시간을 기록하는 함수 setStartTime()은 C 표준 라이브러리 함수 gettimeofday()을 사용하여 현재 시스템의 시간을 얻어오도록 하였다. 그리고 파일 관련 시스템 호출 바로 다음에는 함수 setEndTime()를 사용하여 시스템의 현재 시간을 기록하고 이전에 저장해 두었던 호출 시작 시간과의 차이를 구하여 시스템 호출의 종류에 따라 구별하여 저장하도록 하였다. 그리고 특별히 세 번째 파라미터의 값이 close일 경우 저장해 두었던 각 파일 관련 시스템 호출들이 사용한 시간을 출력하여 로그를 만들도록 하였다. 이상의 두 함수에서 사용되는 파라미터들의 용도는 다음과 같다. 첫 번째 파라미터 fd는 위에 정의한 자료구조 SpendTime의 인덱스로 사용하여 각 파일 기술자를 구별할 수 있게 하였으며, 두 번째 파라미터인 desc는 캐쉬된 파일의 절대 경로를 전달하도록 하였다. 마지막으로 name은 파일 관련 시스템 호출들을 구분할 수 있도록 사용하였다.

네트워크 관련 시스템 호출의 수행 시간을 측정하기 위해서는 아래와 같은 함수를 사용한다.

```
void setStartSystemCallTime(void)
        : 시스템 호출 전에 시스템의 시간을 기록
void calcSystemCallTime(char *name)
        : 시스템 호출의 종류에 따라 호출 이후의 시간을 측정하고 사용 시간을 계산
```

네트워크 관련 시스템 호출인 socket(), bind(), listen(), accept(), close(), connect()의 경우에는 파일과는 달리 수행 중에 중단되는 일이 없기 때문에 누적 측정해야 할 필요가 없다. 따라서 함수 setStartSystemCallTime()를 이용하여 호출 전에 현재 시간을 얻고 함수 calcSystemCallTime()를 이용하여 호출 후에 현재 시간을 얻어 그 차이를 계산하여 즉시 출력하도록 하였다.

5. 실험 결과와 분석

본 논문의 실험에서 사용한 하드웨어와 소프트웨어의 사양은 표 2, 3과 같다. 운영체제로는 두 대의 시스템에

표 2 실험에 사용된 하드웨어 사양

프로세서	800Mhz Intel Pentium III Dual
주메모리	512MB
HDD	30GB SCSI
외부 캐쉬	256KB
NIC	Ethernet 100Mbps

표 3 실험에 사용된 소프트웨어 사양

운영체제	Solaris 8 Intel Platform
	Linux Kernel 2.4.2
프록시 캐쉬 서버	Squid 2.4.STABLE1
트래픽 생성기	Web Polygraph 2.7.3

Solaris 8과 Linux Kernel 2.4.2를 각각 설치하고 Squid를 수행시켰다. 트래픽 생성 소프트웨어인 Web Polygraph는 또 다른 시스템에서 동작시켰다. 우리는 이 실험에서 제 1회 캐쉬 성능 비교(first cache-off)[9]에 사용되었던 워크로드(workload)인 PolyMix #1을 사용하였으며, 이것은 응답 크기의 지수분포 평균이 13KByte이고 캐쉬할 수 없는 응답이 20%이며 서버의 지연이 평균 3초, 표준편차 1.5초이고 문서 적중률(document hit ratio)은 55%인 워크로드로 클라이언트들이 초당 5개의 요청을 하는 경우와 초당 10개의 요청을 하는 경우로 나누어 각 시스템에 설치된 프록시 캐쉬 서버를 테스트하였다. [12]에 의하면 웹 캐쉬 계층

표 4 Squid 2.4.STABLE1에 대해 5개/sec 요청을 했을 경우의 시스템 호출 수행시간

		Solaris 8			Linux		
		call 회수 (개)	call 처리시간 (μsec)	평균 call처리시간 (μsec/개)	call 회수 (개)	call 처리시간 (μsec)	평균 call처리시간 (μsec/개)
file	open	12464.60	12747540.00	1022.69	12203.00	1154222.40	94.58
	close	12464.60	724651.00	58.14	12202.60	148770.20	12.19
	read	7868.20	2823406.20	358.84	7567.00	976454.80	129.04
	write	4539.60	3487230.20	768.18	4558.20	1121295.20	245.99
network	socket(server,client)	8167.40	1079798.80	132.21	8227.00	212842.80	25.87
	close(server,client)	26247.80	3878308.00	147.76	26302.60	841808.60	32.00
	bind(server)	2.00	138.80	69.40	2.00	39.20	19.60
	listen(server)	1.00	58.60	58.60	1.00	32.20	32.20
	accept(server)	18099.00	5203174.20	287.48	18098.60	593375.80	32.79
	read(server)	18099.00	1664708.00	91.98	18097.40	303329.20	16.76
	connect(client)	8179.20	1054448.60	128.92	8248.80	386326.60	46.83
	read(client)	41729.60	2835982.00	67.96	47107.20	1003498.40	21.30
	write(server,client)	95190.00	5843834.60	61.39	96420.60	2884864.60	29.92
합계				3253.55			739.09

표 5 Squid 2.4.STABLE1에 대해 10개/sec 요청을 했을 경우의 시스템 호출 수행시간

		Solaris 8			Linux		
		call 회수 (개)	call 처리시간 (μsec)	평균 call처리시간 (μsec/개)	call 회수 (개)	call 처리시간 (μsec)	평균 call처리시간 (μsec/개)
file	open	26180.80	52452389.00	2003.47	26057.60	2729217.20	104.74
	close	26180.80	1701902.80	65.01	26057.60	343152.20	13.17
	read	15251.80	4357569.40	285.71	14473.80	1812975.80	125.26
	write	10868.40	14290981.40	1314.91	11445.60	2964509.40	259.01
network	socket(server,client)	18151.40	2508377.80	138.19	18760.20	541337.80	28.86
	close(server,client)	54565.20	8185024.40	150.00	55019.60	1828493.80	33.23
	bind(server)	2.00	139.20	69.60	2.00	38.20	19.10
	listen(server)	1.00	57.60	57.60	1.00	31.60	31.60
	accept(server)	36450.00	10504843.40	288.20	36327.80	1205065.40	33.17
	read(server)	36449.60	3246299.20	89.06	36355.00	595169.40	16.37
	connect(client)	18183.00	2483005.00	136.56	18929.60	922424.20	48.73
	read(client)	95467.80	6824033.00	71.48	117688.20	2694690.80	22.90
	write(server,client)	196503.40	12548965.40	63.86	206616.20	6527620.20	31.60
합계				4733.65			767.73

중 최고 수준(root level)에 해당하는 웹 프록시 캐쉬들은 피크(peak)시간에 15~20개/sec 정도의 요청을 받으며, ISP(Internet Service Provider)들의 웹 프록시 캐쉬는 8~12개/sec 정도, 학교, 연구소 등의 웹 프록시 캐쉬는 5~7개/sec 정도의 피크 시간 요청을 받는다고 하였으므로 본 논문에서 실험한 초당 5개의 요청은 단위 기관의 피크 시간 부하에 해당하며 초당 10개의 요청은 ISP의 피크 시간 부하에 해당한다고 볼 수 있다.

실험은 위 4가지 경우에 대해 5회씩 총 20회 실시되었으며, 각 실험은 1시간씩 진행되었다. 5회의 실험 결과를 평균하여 얻어진 결과는 표 4, 표 5와 같다.

실험 대상이 된 두 운영체제는 모든 결과에서 시스템 호출의 수행 시간에 상당한 차이를 보이며 Linux 2.4.2가 Solris 8보다 전반적으로 우수한 것으로 나타났다. 표 6은 어떤 클라이언트의 요청이 캐쉬에 없고 원격 서버로 가서 오브젝트를 가져오는 경우에 필요한 모든 시스템 호출 시간을 합산한 것이다. 물론 오브젝트의 크기에 따라 여러 번의 read(), write()가 필요하지만 단순한 비교를 위해 1회로 계산하였다.

표 6 시스템 호출 수행시간의 합 (단위 μsec)

	5개/sec 요청	10개/sec 요청
Solaris 8	3253.55	4733.65
Linux 2.4.2	739.09	767.73

표 6을 통해 보면 Solaris 8이 Linux 2.4.2에 비해 초당 5개의 요청을 하는 경우에는 4.4배, 초당 10개의 요청을 하는 경우에는 6.2배 정도 시스템 호출에 대한 총 서비스 시간이 많이 걸리는 것으로 나타난다.

표 4, 5의 데이타를 살펴볼 때 각 항목 중 가장 큰 차이를 보이는 것이 파일 시스템에 대한 open()으로 표 4에서는 10배, 표 5에서는 무려 20배의 차이를 보인다.

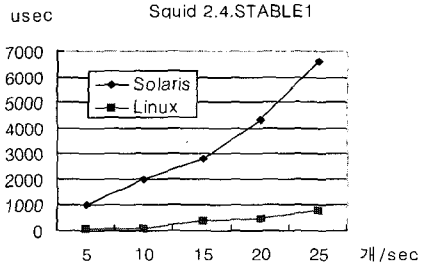

그림 3 파일 시스템 open() 시스템 호출에 걸리는 시간 비교

이는 Solaris의 UFS 기반 파일 시스템과 Linux의 EXT2 기반 파일 시스템의 구조 차이에서 기인 한 것으로 보인다. 그림 3은 파일 시스템에 대한 open() 시스템 호출에 대해 보다 명확히 비교해 보기 위해 수회의 실험을 추가하여 얻어낸 결과이다.

여기서 보면 클라이언트의 요청률 증가에 대해 Linux의 경우에는 비교적 완만한 성능 저하는 보이지만 Solaris의 경우에는 급격한 시간의 지연을 보이는 것을 알 수 있다.

6. 결론 및 향후 연구

본 논문에서는 웹 서비스를 제공하는 시스템 소프트웨어들을 Linux, Solaris 8 등 UNIX 기반의 오픈 소스 운영체제가 얼마나 잘 지원하는 가를 알아보기 위해 소스 수준에서 시간을 측정하는 도구를 개발하고 이를 오픈 소스 웹 프록시 캐쉬인 Squid에 패치하고 이에 Web Polygraph에서 생성된 트래픽을 가하여 각 시스템 호출에 관한 수행 시간을 얻어내고 이를 운영체제 별로 비교하였다.

결과적으로 Linux 2.4.2가 Solaris 8보다 전반적으로 시스템 호출 수행 효율이 우수한 것으로 나타났다. Solaris 8은 파일 시스템 관련, 네트워크 처리 관련 시스템 서비스의 전반 적인 효율을 보다 제고하여야 할 필요가 있으며, 특히 파일 시스템 부분에 있어 open() 시스템 호출 지연에 대해서는 시급히 개선이 요구된다고 하겠다.

추후 본 논문의 결과를 바탕으로 보다 부하가 크게 걸리는 상황에 대한 비교가 필요하리라고 보이며, 웹 시스템 소프트웨어 자체의 성능 평가를 위해서는 소스 코드의 모듈 별 시간 측정이 필요하다. 운영 체제의 경우에도 소스 수준의 분석을 통해 시스템 호출 성능에 영향을 주는 설계상의 요인을 보다 구체적으로 규명하는 일이 필요하다. 이러한 작업을 통하여 웹 서비스를 효과적으로 지원하는 운영체제 또는 웹 관련 서버와 운영체제가 통합된 고효율 서버의 설계와 구현에 한 걸음 다가갈 수 있을 것이다.

참 고 문 헌

[1] Abrams M., C. R. Standrigde, G. Abdulla, S. Williams and E. A. Fox, "Caching Proxies: Limitations and Potentials," *Proceedings of Fourth International World Wide Web Conference*, December 1995.

[2] The Apache Software Foundation, http://www.

apache.org
[3] Squid Web Proxy Cache, http://www.squid-cache.org
[4] Microsoft Corporation, http://www.microsoft.com
[5] Inktomi Corporation, http://www.inktomi.com
[6] Akamai Technologies, Inc., http://www.akamai.com
[7] Cisco Systems, Inc., http://www.cisco.com
[8] http://www.netcraft.com/survey
[9] Web Polygraph, http://www.web-polygraph.org
[10] Banga, G. and P. Druschel, "Measuring the Capacity of a Web Server," *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Moterey, California, December 1997.
[11] Barford, P. and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proceedings of the joint international conference on Measurement and modeling of computer systems*, pp.151-160, Madison, June 1998.
[12] Rousskov, A., "A Performance Study of the Squid Proxy on HTTP/1.0," *WWW Journal 99*, 1999.
[13] Maltzahn, C., K. Richardson and D. Grunwald, "Performance issues of enterprise level web proxies," *Proceedings of ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, Seattle, June 1997.
[14] Kim, J., H. Bahn, K. Koh and Y. Baek, "Modeling of retrieval latency for proxy cache simulation," *Electronics Letters*, Vol.37, No.3, IEE, February 2001.
[15] Dilley, J. and M. Arlitt, "Improving Proxy Cache Performance: Analysis of Three Replacement Policies," *Internet Computing*, Vol. 3, No. 6, IEEE, November 1999.
[16] Nahum, E., T. Barzilai and D. Kandlur, "Performance Issues in WWW Servers," *Proceedings of the international conference on Measurement an Modeling of Computer Systems*, pp.216-217, May 1999.
[17] Banga, G., P. Druschel and J. Mogul, "Better Operating System features for faster network servers," *Proceeding of the Workshop on Internet Server Performance*, 1998.
[18] Baek Y., S. Son and K. Koh, "Analysis on the UNIX Buffer Cache Mechanism of World Wide Web Proxy Server," *Proceedings of the 16th IASTED International Conference Applied Informatics*, Garmisch-Partenkirchen, Germany, Feb. 1998.
[19] IRCache Project, http://www.ircache.net/

백 윤 철
1988년 서울대학교 계산통계학 이학사. 1990년 서울대학교 전산과학 이학석사. 1995년 서울대학교 전산과학 이학박사. 1996년 ~ 현재 상명대학교 소프트웨어학부 부교수. 관심분야는 시스템 소프트웨어, 실시간 시스템, 미디어 시스템

추 연 준
1997년 수원 장안전문대학 전자계산학과 졸업. 2000년 상명대학교 전자계산학 이학사. 2002년 상명대학교 전자계산학 이학석사. 2001년 ~ 2002년 정보통신대학원 부설 정보통신교육원 강사. 2002년 ~ 현재 씽크정보통신 연구원. 관심분야는 Web Caching, Embedded Linux