웹 프락시 서버의 디스크 I/O 성능 평가

심 종 익†

요 약

웹 프락시 서버에서 디스크 I/O는 병목현상을 일으키는 주요 요소이다. 오늘날 대부분의 웹 프락시 서버는 범용 파일 시스템에서 실행되도록 설계되어 있다. 그러나 범용 파일 시스템은 웹 캐시 워크로드에서 대부분 차지하는 작은 파일들을 효과적으로 처리하지 못하기 때문에 전체 웹 프락시 서버의 성능을 저하시키게 한다. 본 논문에서는 범용 파일 시스템을 사용하지 않는 원시(raw) 디스크 I/O 기법이 웹 프락시 서버의 디스크 I/O 오버헤드 개선에 얼마나 영향을 미치는지 그 성능 잠재력을 평가한다. 이를 위해 웹 프락시 서버를 위한 저장관리 시스템인 BSMS(Block-structured Storage Management System)를 설계하고 Squid 소스에 적용시켜 웹 프락시 서버를 구현하였다. 다양한 실험 평가를 통해 원시 디스크 I/O 기법이 웹 프락시 서버에서 디스크 I/O의 성능을 크게 개선시키는 좋은 방법임을 보였다.

키워드 : 인터넷, 웹 프락시 서버, 캐싱, 파일 시스템, 원시 디스크

Performance Evaluation of Disk I/O for Web Proxy Servers

Jong-Ik Shim†

ABSTRACT

Disk I/O is a major performance bottleneck of web proxy server. Today's most web proxy servers are design to run on top of a general purpose file system. But general purpose file system can not efficiently handle web cache workload, small files, leading to the performance degradation of entire web proxy servers. In this paper we evaluate the performance potential of raw disk to reduce disk I/O overhead of web proxy server. To show the performance potential of raw disk, we design a storage management system called Block-structured Storage Management System (BSMS). And we also actually implement web proxy server that incorporate BSMS in Squid. Comprehensive experimental evaluations show that raw disk can be a good solution to improve disk I/O performance significantly for web proxy servers.

Key Words : Internet, Web Proxy Servers, Caching, File System, Raw Disk

1. 서 론

최근의 다양한 연구들은 디스크 I/O가 웹 프락시 서버의 성능을 심각하게 저해하는 중요한 요소 중의 하나임을 보이고 있다[1, 2, 3, 7]. Rousskov와 Soloviev는 디스크의 지연시간이 객체가 히트(hit)되었을 때의 응답시간에 30% 정도의 영향을 미친다는 사실을 밝혔다[9]. Mogul는 디스크 I/O 오버헤드가 캐시 히트를 높여 개선된 지연시간보다도 훨씬 크다고 하였다[3]. 오늘날 대부분의 웹 프락시 서버는 범용 파일 시스템을 기반으로 설계되었다. Harvest와 Squid[4] 그리고 CERN[5]과 같은 연구용 시스템은 유닉스 계열의 파일시스템을 사용한다. 범용 파일 시스템은 일반적인 목적의 워크로드를 위해 최적화되어 있지만, 웹 캐싱은 완전히 다른 워크로드 특성을 가지고 있다. 예를 들면, 전통적인 파일 시스템에서는 읽기 연산이 쓰기 연산보다 많지만, 웹에서는 쓰기 위주의 연산을 보여준다. 또한, 일반적인 보통 파일등은 갱신이 자주 발생되지만, 대부분의 URL들은 갱신이 거의 발생되지 않는다[6]. 웹 프락시 서버의 히트율은 30%-50% 정도로 낮기 때문에 디스크 접근의 대부분은 쓰기 연산이며, 참조되는 웹 객체의 74%는 8KB 보다 작은 크기이다[10]. 이러한 특성은 디스크를 빨리 단편화시키는 원인이 되며, 데이터를 디스크에 빠르게 쓰기 위하여 연속적인 디스크 블럭을 유지시키는 것을 어렵게 만든다. 파일 시스템은 집중적인 파일 연산 흐름(stream)에 직면하게 된다. 따라서 대부분의 일반적인 파일시스템은 캐시 트래픽을 처리하는데 매우 비효율적이다. 많은 시스템들 특히, 상업용 시스템들은 디스크 I/O 오버헤드를 개선시키기 위해 많은 노력을 기울여 왔다. NetAppliance의 NetCache[12]는 자신들이 만든 마이크로커널이 제공하는 API들을 사용한다. 특수하게 만들어진 운영체제와 파일 시스템들은 시스템의 성능을 많이 개선시켰지만, 설계하고 유지 보수하는데 매우 어렵고 비용이 많이 든다. 그리고 이식성이 매우

† 정 회 원 : 한서대학교 컴퓨터정보학과 교수
논문접수 : 2004년 10월 21일, 심사완료 : 2005년 4월 20일

좋지 않다.

본 논문에서는 기존의 파일 시스템을 사용하는 대신 웹 프락시의 디스크 I/O 오버헤드를 줄이기 위해 원시 디스크를 사용하는 기법의 잠재적인 성능을 평가한다. 지금까지 웹 프락시 서버의 디스크 I/O 성능을 향상시키기 위한 방법으로 원시 디스크 기법은 많이 연구되어지지 않았다. 상용 시스템들 중에서 잉크토미(Inktomi)의 트래픽서버(Traffic Server)는 유닉스 원시 디스크를 사용한다. 이 방법을 사용하여 프락시 서버의 성능을 매우 많이 개선시킬 수 있었다. 그러나 원시 디스크를 사용한 이러한 웹 프락시의 성능과 구조 등의 자세한 정보는 거의 발표되지 않았다. 또한, 지금까지 웹 프락시 분야에서 파일 시스템 기반의 시스템과 원시 디스크 기반 시스템간의 직접적인 비교 연구는 거의 없었다. 원시 디스크 기법은 파일 시스템 기법에 비해 많은 장점들을 가지고 있다. 이 기법은 파일 시스템 블록과 프래그먼트(fragment) 할당에 따르는 오버헤드를 피할 수 있게 한다. 커널에서 디렉토리를 트레이스(trace)하고 작은 파일을 열고 닫는 연산을 제거함으로써 작은 객체를 접근하는데 있어서 매우 유리하다. 원시 디스크 기법은 파일을 관리하는데 있어서 완전한 제어를 할 수 있기 때문에 좀 더 좋은 성능을 얻기 위한 응용에 최적화된 알고리즘을 설계할 수 있다. 이 기법은 유닉스 계열의 원시 디스크에서 사용자 응용으로써 작동된다. 일반적인 생각과는 달리 커널을 수정할 필요가 없다.

이 논문의 구성은 다음과 같다. 먼저 가장 널리 사용되는 웹 프락시 서버인 Squid와 최근에 연구된 웹 프락시 서버들의 파일 시스템 상호작용에 대하여 조사한다. 그리고 원시 디스크 기법이 파일 시스템 기반의 기법에 비해 잠재적인 성능이 우수함을 보여주기 위하여 최근의 연구들을 기반으로 BSMS(Block-structured Storage Management System)를 설계한다. 또한 이렇게 설계된 BSMS를 가지고 Squid에 원시 디스크 기반의 웹 프락시 서버(WPS-RAW)와 파일 시스템 기반의 웹 프락시 서버(WPS-FS)를 구현한다. 웹 폴리그래프(Polygraph) 성능 벤치마크[8]를 이용하여 제안된 시스템의 성능을 평가한다.

2. 기존 연구들의 캐시 구조

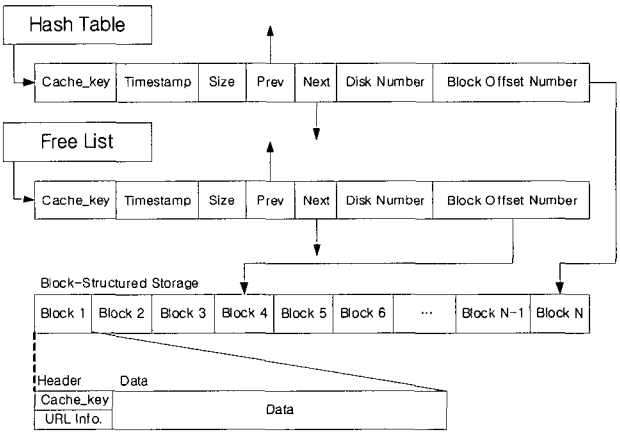
Squid는 연구 분야와 상용 서버 분야 모두에서 많이 사용되고 인정받는 캐시 소프트웨어이다. 운용체제의 기본 파일 시스템을 사용하며 모든 객체들을 개별 파일에 저장한다. 파일을 저장하기 위하여 Squid는 계층구조의 디렉토리를 사용한다. 기본 설정은 16개의 디렉토리 구조를 가지며 이러한 각 디렉토리는 다시 256개의 두 번째 디렉토리 구조를 포함하고 있다. 객체가 개별 파일에 저장되기 때문에 프락시는 각 URL이 miss 될 때마다 파일을 생성하고 삭제하여야만 한다.

파일의 생성과 삭제 연산에 따른 오버헤드를 줄이기 위하여 디스크 트래픽의 양을 많이 줄인 최근의 연구들[6, 10, 11]은 미리 정의된 임계치보다 작은 객체들을 저장하기 위하여 단일 파일을 사용한다. 공간 관리를 단순화하기 위해 파일은 고정된 크기의 블록으로 구성된다. 따라서 객체들은 실제 객체의 크기 대신 블록 단위로 파일에 저장된다. 그리고 파일로부터 객체들을 검색하기 위해 오프셋과 크기 정보를 이용한다. 이러한 방법은 파일 이름을 inode로 변환하고 이러한 inode를 유지관리하고 저장하는 오버헤드를 피할 수 있게 한다. 그러나 이러한 방법은 다양한 크기의 객체들을 저장하기 위해 고정된 크기의 블록을 사용하기 때문에 공간의 낭비가 발생한다. 따라서 최근 연구들은 공간 사용을 최적화하기 위하여 객체를 적절한 크기의 파일에 저장하는 객체-그룹핑(object-grouping) 방법을 제안하였다. 즉 하나의 블록이 필요한 객체들은 모두 같은 파일에 저장하고, 두개의 블록이 필요한 객체들은 모두 다른 파일에 저장하는 방법이다. 이러한 방법은 파일 시스템 호출의 수를 줄여줄 수 있으며, 디스크 사용을 효율적으로 최적화해 줄 수 있다.

3. 시뮬레이션 기반의 원시 디스크 I/O 평가

3.1 블록구조의 저장관리 시스템

원시 디스크를 사용하는 기법의 잠재적인 성능을 조사하기 위하여 최근의 연구들을 기반으로 블록구조의 저장관리 시스템(BSMS)을 설계하였다. (그림 1)은 BSMS의 구조를 보여주고 있다. 객체 데이터들은 블록들로 구성된 디스크 파일에 저장된다. 그리고 하나의 블록 크기는 8KB이며, 블록은 읽기와 쓰기의 기본 단위가 된다. 각 블록은 헤더 영역과 데이터 영역으로 구성된다. 헤더는 해시 테이블의 탐색 키로 사용되는 캐시 키를 포함한다. 저장된 객체들의 정보를 유지관리하기 위하여 BSMS는 주기억 장치에 메타 데이터를 갖는다. 주기억 장치에 있는 메타 데이터는 디스크에 저장된 객체 데이터들의 위치를 파악하는데 이용되는 해시 테이블과 객체의 정보를 포함하는 자료구조로 구성된다. 디스크에 저장된 각 블록은 블록 오프셋 번호와 메타 데이터에 의해 그 위치가 파악된다.



(그림 1) 블록구조의 저장 관리 시스템 구조

3.2 시뮬레이션 모델

원시 디스크의 I/O 연산 성능을 평가하기 위하여 파일 시

스템 기반의 BSMS(BSMS-FS)와 SQUID-LIKE 그리고 원시 디스크 기반의 BSMS(BSMS-RAW) 등 세 가지 시뮬레이션 모델을 구현하였다. 데이터를 관리하기 위하여 BSMS-FS는 캐시 디스크 크기의 단일 파일을 먼저 만들어야 하고, 이러한 과정은 캐시 디스크의 크기에 따라 시간이 많이 소요되는 작업이다. BSMS-RAW에서는 이러한 과정이 필요없다. 디스크나 디스크내의 파티션을 설치한 다음, mkfs나 newfs 등의 명령어로 파일 시스템을 설치하는 과정을 하지 않으면 원시 디스크 상태로 캐시 디스크를 운용할 수 있다. BSMS-RAW에서는 프로세스가 디스크와 직접 상호작업하게 되며, 디스크 입출력은 버퍼링되지 않는다. SQUID-LIKE 구조에서는 모든 요청된 객체들이 별도의 파일에 저장되도록 설계되어 있으며, Squid에서 사용되는 디스크 관리 방식을 따르며 계층형 디렉토리를 사용한다.

BSMS-FS와 SQUID-LIKE는 리눅스 운영체제의 파일 시스템인 ext2와 솔라리스 운영체제의 ufs에서 실행되며, BSMS-RAW는 리눅스와 솔라리스 운영체제의 원시 디스크에서 실행된다. 각 모델에서 사용되는 캐시 디스크의 크기는 5GB이다. 실제와 같은 워크로드를 발생시키기 위하여 프락시 성능 벤치마크 도구인 웹 폴리그래프의 클라이언트로부터 발생된 트레이스를 사용하며 초당 350 요청을 발생시켰다.

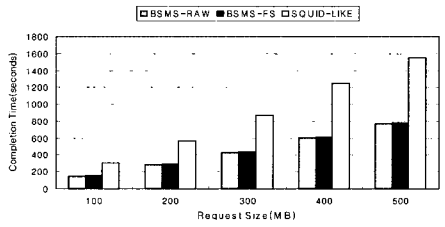
3.3 평가

3.3.1 읽기/쓰기 연산의 완료 시간

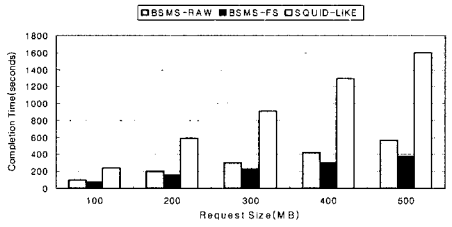
평가는 주어진 크기 만큼의 읽기와 쓰기 요청을 모두 마치는데 필요한 총 완료시간을 측정하여 이루어진다. (그림 2) (a)와 3(a)는 리눅스 운영체제에서 실행된 세가지 모델의 읽기와 쓰기 성능을 보여주고 있으며 (그림 2) (b)와 (그림 3) (b)는 솔라리스 운영체제에서 실행된 성능을 보여주고 있다. (그림 2) (a)에서 볼 수 있듯이 읽기 연산에서 BSMS-RAW의 성능이 BSMS-FS보다 약간 좋다. BSMS-RAW와 BSMS-FS 모두 SQUID- LIKE와 비교해서 읽기 연산의 오버헤드를 2배 정도 줄였다. (그림 3) (a)는 BSMS-RAW가 쓰기 연산에서 SQUID-LIKE에 비해 3배 그리고 BSMS-FS에 비해 최고 50%정도 성능을 개선시키고 있음을 보여준다. (그림 2) (b)에서 볼 수 있듯이 솔라리스 운영체제에서 BSMS-RAW는 읽기 연산에서 BSMS-FS에 비해 성능이 약간 떨어지지만, 쓰기 연산에 있어서는 (그림 3) (b)에서와 같이 성능이 2배 정도 개선된 것을 볼 수 있다. 웹 접근은 쓰기 위주의 워크로드를 보여주기 때문에 실세계에서 BSMS-RAW의 전체 성능은 BSMS-FS보다 그 만큼 더 개선될 것이다.

3.3.2 주기억 장치의 영향

BSMS-RAW와 BSMS-FS의 성능을 좀 더 조사하기 위해 시뮬레이션 모델의 가용 메모리 크기를 변화시키면서 500MB를 완료하는데 걸리는 시간을 측정하였다. 가용 메모리의 크기를 제한하기 위해서 단일 더미(dummy) 프로세스를 사용하였다. (그림 4)와 (그림 5)가 이러한 결과를 보여주고 있다. X축은 시뮬레이션에서 사용된 주기억장치의 크기를 나타낸다. X축에 있는 각 가용 메모리는 다음과 같이 계산된다: 1GB-(시뮬레이션 프로그램이 차지하는 메모리의 크기+운영체제에서 사용하는 메모리의 크기+더미 프로세스에 의해 제한된 메모리의 크기).

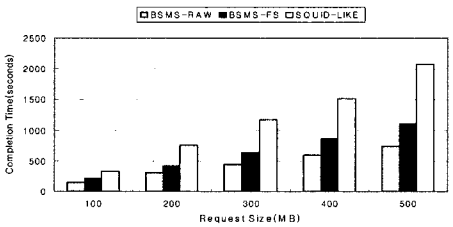


(a) under Linux OS

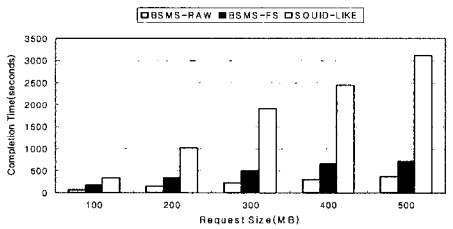


(b) under Solaris OS

(그림 2) 읽기 연산의 성능



(a) under Linux OS

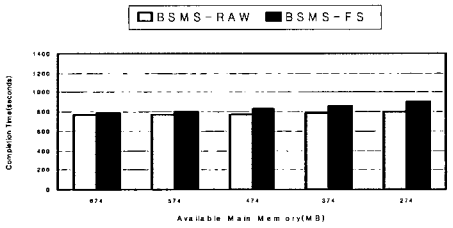


(b) under Solaris OS

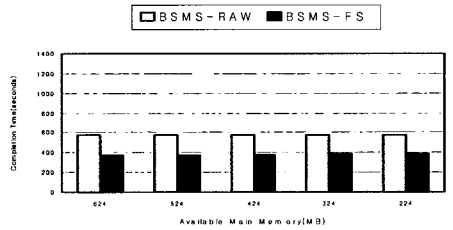
(그림 3) 쓰기 연산의 성능

(그림 4)와 (그림 5)는 BSMS-RAW가 두가지 운영체제에서 읽기와 쓰기 연산 모두 메모리 크기에 영향을 전혀 받지 않는다는 것을 보여주고 있다. BSMS-FS는 (그림 4) (a)와 (그림 5) (a)에서 보여 지는 것처럼 리눅스 운영체제에서 읽기와 쓰기 연산 모두 가용 메모리에 영향을 받고 있음을 보여준다. 사용할 수 있는 메모리의 크기가 줄어들면 BSMS-FS의 성능이 떨어진다. (그림 4) (b)는 솔라리스 운영체제에

서 읽기 연산이 가용메모리의 크기에 영향을 받지 않고 있음을 보여준다. 그러나 (그림 5) (b)에서 보여지는 것처럼 BSMS-FS는 가용 메모리가 줄어들수록 쓰기 연산의 성능이 현저하게 저하된다. 가용메모리가 624MB보다 작게 되면 BSMS-RAW는 BSMS-FS보다 성능이 2배 이상 개선된다. 일반적인 웹 프락시 서버의 히트율이 30%-50% 정도이기 때문에 50%-70%의 디스크 접근은 쓰기이다. 이러한 결과는 원시 디스크 기법이 파일 시스템 기법과 비교해서 실제 웹 워크로드 환경에서 더 많은 성능향상이 기대됨을 말해준다.

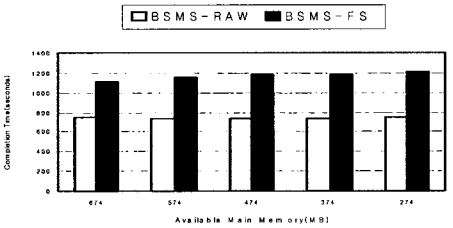


(a) under Linux OS

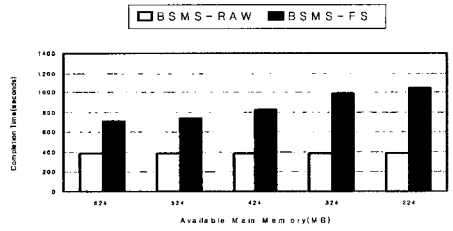


(b) under Solaris OS

(그림 4) 가용 메모리 크기별 읽기 연산의 성능



(a) under Linux OS

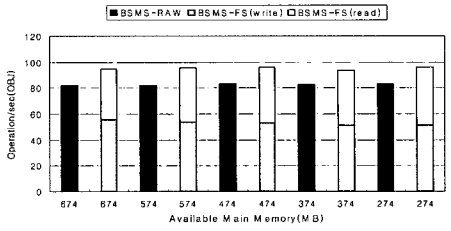


(b) under Solaris OS

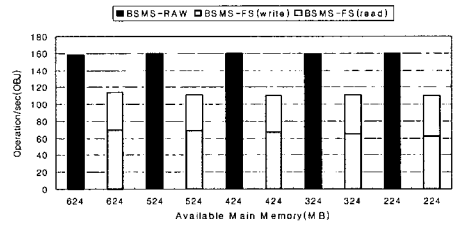
(그림 5) 가용 메모리 크기별 쓰기 연산의 성능

3.3.3 in/out 연산의 통계

앞의 두절에서 보였듯이 쓰기 연산이 읽기 연산보다 웹 프락시 서버의 디스크 I/O 성능에 많은 영향을 미치고 있다. BSMS-FS와 BSMS-RAW의 쓰기 연산 행동양식(behavior)을 좀더 이해하기 위해 유닉스 iostat 유틸리티를 이용하여 가용 메모리의 크기를 다르게 하면서 초당 쓰기 연산의 요청수를 측정하였다. (그림 6)과 (그림 7)은 리눅스와 솔라리스 운영체제에서 초당 쓰기 요청의 수를 보여주고 있다.



(그림 6) Linux 운영체제에서의 디스크 쓰기 연산 수



(그림 7) Solaris 운영체제에서의 디스크 쓰기 연산 수

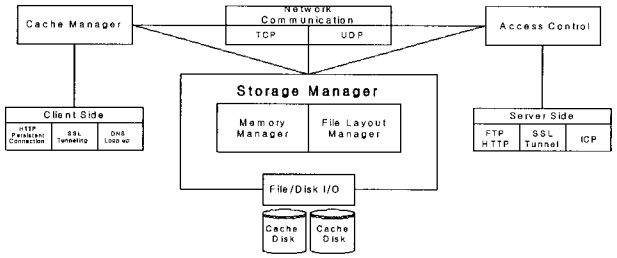
이 실험에서 BSMS-FS의 쓰기 연산의 수를 측정할 때, 우리는 쓰기 연산이 읽기 연산을 발생시킨다는 사실을 알게 되었다. 프로세스가 파일에 작은 양의 데이터를 쓰게 되면 운영체제는 디스크로부터 같은 페이지(그 페이지가 이미 주기억장치에 없는 경우)를 읽게 되고 주기억 장치 페이지에 쓰기를 한 다음, 나중에 갱신된 전체 페이지가 디스크에 쓰여 지게 된다. 이러한 것을 심각한 작은 쓰기 성능문제라 한다[6]. 쓰기 연산을 할 때 파일 시스템에서 발생되는 이러한 현상은 웹 프락시 서버에 상당한 성능 병목현상을 발생시키게 한다. 파일 시스템에서 작은 크기의 쓰기 연산 중에 발생되는 불필요한 읽기 연산 현상을 없애는 것은 매우 어렵다. 그러나 BSMS-RAW 기법에서는 (그림 6)과 (그림 7)에서 볼 수 있듯이 특별한 메커니즘을 사용하지 않고서도 이러한 불필요한 읽기 연산이 전혀 발생되지 않는다는 사실을 알 수 있다.

4. 구 현

4.1 WPS 설계

제안된 모델이 실세계에 적용할 수 있음을 보이기 위해 Squid 소스 코드에 BSMS를 적용한 두개의 사용자 레벨의 웹 프락시 서버를 구현하였다. 하나는 원시 디스크 기반으로 Squid에 BSMS를 구현한 웹 프락시 서버(WPS-RAW)이고, 또 하나는 파일 시스템 기반으로 Squid에 BSMS를 구현한 웹 프락시 서버(WPS-FS)이다. (그림 8)에서 보여진 것과 같이 구현된 시스템은 몇 가지 주요 요소들로 구성되어 있다. 클라이언트 부분(Client Side)은 새로운 클라이언트 연결을

받아들여 구문을 해석하고 처리하는 곳이다. 여기에서 요청이 hit 인지 refresh인지 miss인지를 결정한다. 하나의 TCP 연결로부터 다중 요청이 들어온다. 서버 부분(Server Side) 루틴은 프로토콜에 따라 miss가 된 캐시 요청을 다른 서버로 넘기는 책임을 갖는다. 캐시 miss는 다른 오리지널 서버나 다른 프락시 서버에 전달될 것이다. 저장 관리자(Storage Manager)는 가장 주요한 구성요소 중의 하나로 앞 절에서 설계한 BSMS가 구현된 곳이다. 캐시 공간이 부족할 때 이루어지는 교체 정책으로 LRU를 사용한다. 접근 제어(Access Control) 루틴은 몇 개의 파라메터들을 근거로 요청을 거절할 것인가 허용할 것인가를 처리하는 부분이다. 이러한 파라메터들로는 클라언트의 IP 주소, 요청된 자원의 호스트이름 그리고 요청 방법 등이 있다. 네트워크 통신(Network Communication)은 TCP와 UDP 네트워크 소켓을 통신하는 곳이다. 소켓이 열리고, 닫히고, 읽고 쓰여지는 곳이다. 캐시 관리자(Cache Manager)는 캐시 관리자가 필요로 하는 특정 정보를 접근할 수 있게 하는 곳이다.



(그림 8) 시스템의 기본 구조

4.2 실험

실험은 초당 요청 수와 응답시간으로 표현되는 단위시간당 처리량(throughput)을 측정하여 이루어지며, 이를 위해 상용 프락시 서버를 벤치마킹하는데 사용되는 산업 표준 웹 프락시 성능 벤치마크 도구인 PolyMix-3 트래픽 모델을 사용하였다. 폴리그래프 서버, 폴리그래프 클라이언트 그리고 구현된 웹 프락시 서버 등 세대가 사용되었다. 폴리그래프 서버와 폴리그래프 클라이언트는 BSD 4.3 O.S에서 20GB 디스크와 256MB의 RAM를 갖는 550MHz Intel PIII 프로세서를 사용하였다. 웹 프락시 서버는 Linux 7.1 O.S에서 32GB 디스크와 1GB RAM을 갖는 700MHz Intel PIII 프로세서를 사용하였다. 그리고 이 컴퓨터들은 독립적인 LAN을 구성하기 위해 100Mbps 스위치에 연결하였다. WPS-FS와 Squid는 리눅스 ext2 파일 시스템에서 실행하였으며, WPS-RAW는 리눅스 원시 디스크에서 실행되었다. 이 실험에 사용된 Squid 버전은 STABLE.3이다.

(그림 9~14)는 웹 폴리그래프 성능 벤치마크에 의해 자동 생성된 보고서의 일부분을 보여주고 있다. (그림 9, 10, 11)은 WPS-RAW, WPS-FS 그리고 Squid의 초당 요청수로 표현되는 성능을 보여주고 있다. (그림 12, 13) 그리고 (그림 14)는 캐시 miss, mean, hit 등 세가지 종류의 응답시간을 보여주고 있다. (그림 9)에서 보여 지는 것처럼 WPS-RAW의 초당 요청수는 220이다. (그림 10)과 (그림 11)은 Squid의 성능이 초당 76 그리고 WPS-FS의 성능이 초당 120임을 보여주고 있다. WPS-RAW의 성능이 WPS-FS에 비해 1.8배, Squid에 비해 3배 정도 빠른 것을 보여주고 있다. 이러한 결과는 3절에서 원시 디스크 기법이 디스크 I/O의 성능을 개선시킨다는 결과와 일치되는 것이다. (그림 12)에서 보여지는 것과 같이 WPS-RAW의 hit일때의 응답시간이 0.1초이고 miss일 때의 응답시간이 2.8초임을 알 수 있다. (그림 13)은 WPS-FS의 hit와 miss일때의 응답시간이 0.9와 3.1초임을 보여준다. (그림 14)는 Squid의 응답시간이 실제로 사용할 수 없을 정도로 아주 나쁜 것을 보여주고 있다. 따라서 Squid의 실제 단위시간당 처리량은 (그림 11)보다 훨씬 작을 것이다. (그림 12, 13, 14)에서 볼 수 있듯이 WPS-RAW의 세 가지 종류의 응답시간이 WPS-FS에 비해 좋다.

5. 결과 및 향후 연구

이 논문에서는 웹 프락시 서버의 디스크 I/O 오버헤드를 줄이기 위하여 원시 디스크를 사용하는 기법의 잠재적인 성능을 평가하였다. 성능 평가결과 원시 디스크 방법은 심한 웹 워크로드에서 웹 프락시 서버의 디스크 I/O 성능을 개선시키는 훌륭한 하나의 방법임을 보였다. 특히, 원시 디스크 기법은 작은 쓰기 연산에 의해 발생되는 불필요한 읽기 연산을 방지하며 가용 메모리의 용량에도 거의 영향을 받지 않는다. 이 방법은 집중적인 파일 연산 스트림을 갖는 많은 다른 응용에도 적용될 수 있을 것이다. 기존의 파일 시스템과의 상호작용 없이 원시 디스크에 객체 데이터를 관리하기 위하여 일반적인 생각과는 달리 OS 커널을 수정할 필요가 없다. WPS-RAW는 완전히 사용자 영역에서 수행되기 때문에 훨씬 이식성이 좋으며 벤더에 독립적이어서 구현하고 유지 관리하는데 비용이 적게 든다.

향후 WPS-RAW의 성능을 더욱 향상시키기 위하여 요청 객체들에 대한 지역성을 보존하고 디스크 공간을 최적화하는 많은 방법들과 다른 웹 관련 시스템에 원시 디스크 접근 방법을 적용할 수 있는 문제들에 대한 연구를 진행할 예정이다.

참 고 문 헌

[1] K. Kant and P. Mohapatra. Scalable Internet Servers : Issues and Challenges. ACM SIGMETRICS Performance Evaluation Review. Vol.28 (2000)

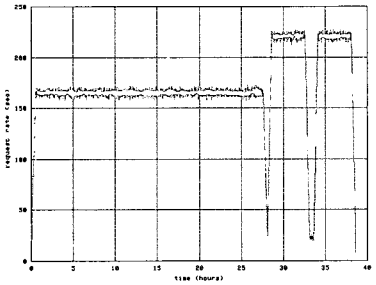
[2] C. Maltzahn, K. Richardson, and D. Grunwald. Performance Issues of Enterprise Level Web Proxies. In Proceedings of ACM SIGMETRICS (1997)

[3] J. Mogul. Speedier Squid : A Case Study of an Internet Sever Performance Problem. Login : The USENIX Association Magazine. 24(1) (1999)

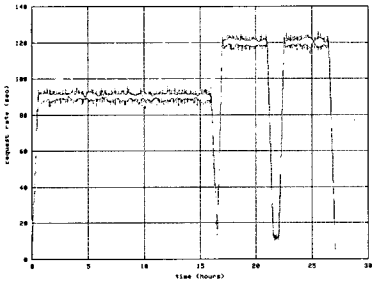
[4] D. Wessels. Squid Internet Object Cache. http://www.squid-cache.org

[5] A. Luotonen, H. Nielsen, and T. Berners-Lee. CERN httpd 3.0A. http://www.w3.org

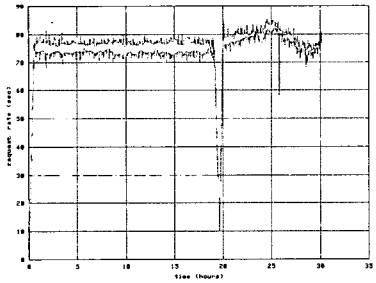
[6] E. Markatos, D. Pnevmatikatos, M. Flouris, and M. Katevenis. Web-Conscious Storage Management for Web Proxies. IEEE/ACM Transactions on Networking (TON). VolumeNumber 6 (2002)
[7] J. Wang and Dong Li. A Light-weight, Temporary File System for Large-scale Web Servers. MASCOTS (2003)
[8] Web Polygraph. Web Polygraph Proxy Performance Benchmark. http://polygraph.ircache.net
[9] A. Rousskov and V. Soloviev. On Performance of Caching Proxies. In Proceedings of the ACM SIGMETRICS Conference (1998)
[10] C. Maltzahn, K. J. Richardson, and D. Grunwald. Reducing the Disk I/O of Web Proxy Sever Caches. In Proceedings of the 1999 USENIX Annual Technical Conference(1999)
[11] V. Soloviev and A. Yahin. File Placement in a Web Cache Sever. In Proc. 10th ACM Symposium on Parallel Algorithms and Architectures(1998)
[12] P. Danzig. Network Appliance NetCache White Paper : Architecture and Deployment. http://netapp.com



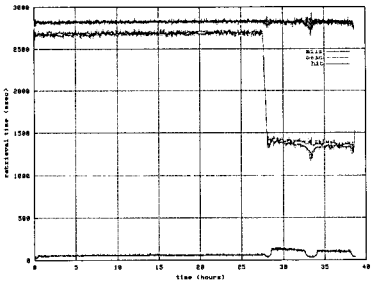
(그림 9) The Request Rate of WPS-RAW



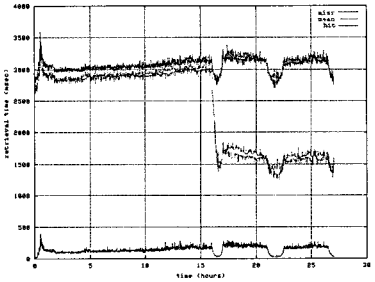
(그림 10) The Request Rate of WPS-FS



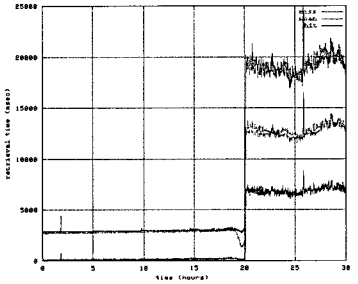
(그림 11) The Request Rate of SQUID



(그림 12) The Retrieval Time of WPS-RAW



(그림 13) The Retrieval Time of WPS-FS



(그림 14) The Retrieval Time of SQUID

심 종 익

e-mail : jishim@hanseo.ac.kr
1985년 인하대학교 전자계산학과(학사)
1987년 인하대학교 대학원 전자계산학과 (석사)
1998년 인하대학교 대학원 전자계산공학과(공학박사)
1986년~1993년 LG산전 연구소 선임연구원
1994년~현재 한서대학교 컴퓨터정보학과 부교수
관심분야 : 인터넷 통신, 데이터베이스, 실시간 시스템