웹 캐시 서버를 위한 저수준 파일시스템 설계 및 구현

Design and Implementation of Raw File System for Web Cache Server

김 성 락* 구 용 완**
Seong-Rak Kim Young-Wan Koo

요 약

범용으로 설계된 EXT2나 UFS에 캐시 데이터를 저장하는 기법은 파일 구조가 범용이기 때문에 웹 캐시에서 요구하는 속도를 충족시키지 못한다. 본 연구에서는 웹 파일에 대한 특성을 적용하여 파일 시스템을 최적화함으로써 좀 더 나은 솔루션이 존재한다는 사실을 제시하였다. 제안한 저수준 캐시파일 시스템(RawCFS:Raw Cache File System)은 캐시된 객체의 크기와 액세스 권한 변경이 불가능하고, 오리지널 서버에 최신본이 존재한다는 사실에 기인하였다. 본 파일 시스템은 성능 평가에서 객체 단위로 개별 파일에 저장하는 기법에 비해 40% 가량 빠른 것으로 확인되었으며 캐시 서버뿐만 아니라 이미지나 HTML 페이지 같은 객체를 클라이언트에게 빠른 서비스를 위한 쇼핑몰이나 인터넷 방송국 같은 대용량 웹 서버 설계시에도 활용될 수 있다.

Abstract

The technique which stores cache data in EXT2 or UFS designed for general purpose is not suitable for satisfying the speed required for web cache due to the general purpose file system. This study shows that there is the better solution by optimizing the file system using the characteristics of web file. It is impossible that the suggested RawCFS changes the size of cached object and the access authentication, and this results from the existence of up-to-dated object in the original server. This file system is proved in the capability test that it is faster than the technique by 40% which stores in each file by object unit. This can be used in the design of high end web server such as shoppingmall or Internet Broadcasting station which should provide objects like image or HTML as well as cache server to the client for the fast service.

Keyword : Web Cache, Cache Server, Cache FileSystem

1. 서 론

웹 캐시 서버는 전용 서버에서 운영되는 소프트웨어 시스템으로 웹 이용자와 웹 서버 사이에 웹 통신을 전송하는 역할을 한다. 캐시 시스템에서 가장 중요한 요구사항은 클라이언트에게 요청한 데이터를 빠르게 전송해 주는 기술이다. 하지만 네트워크 지연과 더불어 웹 캐시 수행을 지연시키는 주요한 원인은 디스크 i/o로, 범용 파일 시스템 EXT2나 UFS에서 이들 캐시 데이터를 저장하는 기법은 범용으로 파일시스템이 설계된 탓에 웹 캐시에서 요구하는 속도를 충족시키지 못한다. 본 연구에서는 웹 파일에 대한 특성을 적용하여 파일 시스템을 최적화함으로써 좀 더 나은 솔루션이 존재한다는 사실을 제시한다. 제안한 저수준 캐시 파일 시스템(RawCFS:Raw Cache File System)은 객체의 특성을 고려하여 설계하였다. 본 액세스 기법은 성능 평가에서 객체 단위로 개별 파일에 저장하는 기법에 비해 40% 가량 빠른 것으로 확인되었다.

본 파일시스템 액세스 기법은 캐시 서버뿐만 아니라 일반 웹 서버에서도 이미지나 HTML 페이지 같은 객체를 클라이언트에게 빠른 서비스를 위한 쇼핑몰이나 인터넷 방송국 등의 대용량 웹 서버 설계시에도 활용될 수 있다.

* 종신회원 : 오산대학 인터넷정보관리과 조교수
ksr01@osan.ac.kr(제1저자)
** 종신회원 : 수원대학교 정보공과대학장, 컴퓨터학과 교수
ywkoo@.suwon.ac.kr(공동저자)

2. 관련 연구

2.1 비상용 웹 캐시 Squid

본 연구에서 살펴본 웹캐시 엔진은 비상용으로 캐싱 서버 업체에서 주로 사용하고 있는 Squid를 다루었다[1][2]. 현재 Squid는 2.6 개발버전까지 나와 있다. 초기 버전 1.0으로부터 시작한 버전은 수많은 기능 추가와 성능 향상을 위한 코드가 추가되어 있어 상업용으로 사용하기에도 손색이 없다. Squid는 다음과 같은 모듈로 구성되어 있다

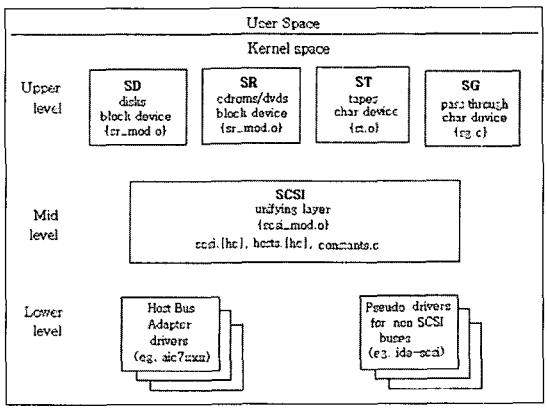
- 서버 쪽 모듈(CP 모듈, HTTP 모듈, FTP 모듈, Gopher 모듈)
  각각의 프로토콜을 처리하기 위해 사용되는 모듈들이다. 여기서는 HTTP모듈을 중심으로 다룬다.
- 클라이언트 쪽 모듈
  클라이언트의 접속을 받아들여서 요청 메시지를 분석하고 그것에 해당하는 서버 모듈을 수행시킨 후 응답을 전송하는 모듈이다.
- 저장 장치 모듈
  클라이언트의 요청에 응답하기 위해서 서버에서 얻어오거나 디스크에서 읽을 때, 메모리를 사용해야 하는 것은 당연한 일이다.
- Pass 모듈
  캐시에 저장될 수 없는 것인 경우 곧장 서버에 연결하여 클라이언트의 요청을 전달하고, 답신을 받아오는 것을 처리하는 모듈이다.
- DNS 캐시 모듈
  Squid에서는 두 개의 DNS 캐시 모듈이 존재한다. 그 중 하나는 호스트 명을 통해 IP 주소를 얻어내는 모듈인 IP cache이고 다른 하나는 IP cache와는 반대로 IP 주소에서 호스트 명을 얻어내는 모듈인 FQDN cache이다.
- unlink 데몬 모듈
  unlink 함수를 호출하게 되면 꽤 시간이걸린다. 따라서 unlink 함수를 호출하는데 드는 시간을 절약하기 위해서 파일을 지우는 역할을 하는 프로세서를 따로 만들어 파일을 삭제할 필요가 있을 때 호출하여 사용하도록 한 모듈이다.
- Redirect 모듈
  Squid에서는 Redirector라는 외부 프로그램을 사용할 수 있게 하는데, 이 프로그램은 요청된 URL로 새 URL을 만들어 내는 프로그램이다.

2.2 SG(Scsi Generic) 구조도

SCSI Sub시스템은 아래 (그림 1)과 같이 3단계의 구조를 갖는다.

최상위 레벨은 사용자 공간에 인접한 공간인 반면, 최하위 레벨은 하드웨어 레벨에 인접한 공간이다. 최상위 레벨은 사용자 레벨 인터페이스를 지원한다. 본 레벨에서 SD와 SR은 블록 디바이스용이며, ST와 SG는 문자 디바이스용으로 나뉜다. 두 번째 레벨에서는 하위레벨에 대한 표준 인터페이스와 상위 레벨에 대한 서비스를 제공한다. ioctl 함수가 본 레벨에 속하며 상위 4개의 디바이스에 대한 인터페이스를 제공한다. 최하위 레벨은 장치 드라이버(Device Driver)이다. 블록 디바이스에서 일반적인 오퍼레이션은 mount 이다. dd 명령은 블록 디바이스에서 512 바이트 또는 8192 바이트로 데이터를 읽고 쓰는데 사용된다[3].

SG는 문자형 디바이스 인터페이스를 사용하여



(그림 1) 리눅스 디바이스 드라이버의 구조도

블록 디바이스로부터 읽고/쓰는 SCSI 인터페이스이다. 일반적으로 SG는 Linux 명령어로 사용치 않고, 스케너나 CD-ROM Writer 같은 저수준 입출력을 위한 어플리케이션에서 사용된다. 모든 SCSI 디바이스는 SG를 통하여 액세스 할 수 있다.

RAW-IO 기법은 리눅스에서 블록 디바이스를 문자형 디바이스처럼 핸들링 하기 위해 리눅스에서 제공하는 바인딩 기법이다. RAW-IO는 버퍼 캐시를 거치지 않고 블록 디바이스에 직접 액세스할 수 있다. DBMS 같은 복잡하고 대규모 어플리케이션에서 사용되고 있는 기법이다.

다양한 SCSI 명령어 유틸리티들은 포맷, 불량(bad) 블록 진단, 성능평가 기능 등 SCSI 장치에 대한 진단 및 유지에 대한 기능을 제공하고 있다. 이들 유틸리티를 이용하여 파일 시스템의 설계에 앞서 EXT2 파일시스템과 저수준 파일 시스템에 대한 단위 평가를 행할 수 있어 설계에 도움을 받을 수 있다[4][5].

3. 구현

3.1 동기

캐시 서버의 성능에 가장 발목을 잡는 것은 바로 디스크이다. 램에 비해 턱없이 느린 디스크의 I/O는 많은 캐시 벤더들에게 있어서 최적화하는데 가장 많은 시간을 들이게 하는 요인이다. 보통 운영체제에서 제공하는 일반적인 목적의 파일 시스템을 그대로 쓰기에는 무리가 많다.

일례로 공개용 웹 캐시엔진 Squid가 사용하는 파일시스템으로는 기본 운영체제의 EXT2 나 UFS가 주로 이용된다. Squid의 저장장치관리(disk.c)에 대해 테스트 한 결과 EXT2에서 파일을 삭제하는데 걸리는 시간이 동일 파일을 읽는데 걸리는 시간보다 5 배 가량이 더 길었다[1][2].

웹 객체를 액세스하기 위해 Read와 Write, Remove 오퍼레이션이 대부분을 차지하는 캐시서버인 경우 본 파일시스템으로써 발생하는 성능의 저하가 매우 심각하다. 본 연구에서 설계한 파일시스템은 다음과 같은 특징을 갖는다.

웹 캐싱에서 모든 파일은 몇 가지 공통적인 특징이 있다. 객체가 일단 기록되면 그 객체의 크기는 변하지 않는다. 이것은 저수준 파일시스템의 매우 중대한 특징이다. 첫 번째 파일 기록 동작이후 수정될 수 없다. 사실, 본 시스템은 오로지 파일에 대한 3가지 오퍼레이션(create, read, delete)만 제공한다. 모든 파일은 결코 퍼미션(Permission) 변경을 허용치 않고, 오리지널 웹 서버상에 모든 파일에 대한 백업-본을 가지고 있다. 이들 중 마지막의 두 가지 점 중에서 첫 번째 것은 파일에 저장할 필요가 있는 메타 데이터 양이 적어도 된다는 것을 의미한다. 이러한 특성에 따라, 저수준 파일시스템은 많은 캐시 서버에서 사용되고 있는 EXT2보다 우수한 성능을 제공할 수 있다[3][4][5][7].

3.2 Reduced Inode Information

범용 파일 시스템에서는 파일의 여러 정보를 Inode 테이블에 유지한다. 이들 정보는 파일유형, 파일 소유자 ID, 그룹 ID, 권한, 데이터 블록 주소, 파일의 크기, 파일 생성 시간, 파일최근참조 시간, 변경시간, 링크 개수 등을 기록한다. 웹 객체의 특성을 분석한 결과 일반 범용 파일시스템과는 달리 웹 객체에 대한 갱신은 특정 프로세스에 의해서만 이뤄지며 한번 저장되면 그 크기는 다시 변경되지 않으며, 객체에 문제가 생기면 오리지널 웹서버에서 최신의 객체로 갱신이 가능하고, 해당 객체에 대한 만료시간 역시 객체내 html 헤더에 이미 저장되어 있으므로 별도로 유지할 필요가 없다.

이에 따라 설계된 웹 파일시스템에서는 웹 페이지의 특성과 이들의 액세스 유형을 고려하여 범용 파일시스템에서 유지되었던 Inode 정보를 대폭 축소하여 (그림 2)에서 볼 수 있듯이 파일시스템을 구분하는 매직넘버, 파일시스템에 저장가능한 블록개수, Inode 비트맵 시작 위치, Fragment

매직넘버	블록개수	IBM위치	FBM위치
비트맵 길이			
~			
IBMPOS (size = 블록개수 / 8bit)			
FBMPOS (size = 블록개수 / 8bit)			
페이지 크기	블록번호1	블록번호2	블록번호3
Direct pointer			
			블록번호63
간접포인터1	간접포인터2	간접포인터3	간접포인터4
Indirect pointer			
간접포인터61	간접포인터62	간접포인터63	간접포인터64
데이터 블록1			
데이터 블록2			

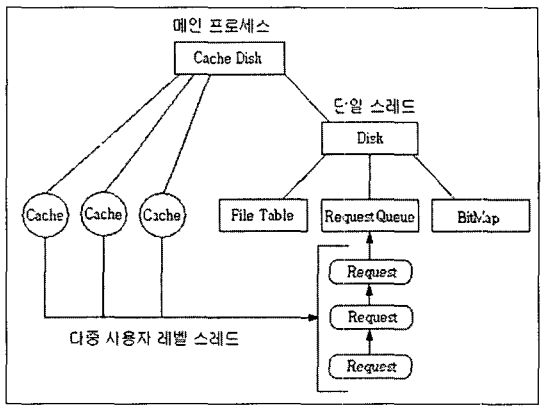
(그림 2) 저수준 파일 입출력을 위한 데이터 구조체

비트맵 시작위치, 비트맵의 길이(ibm, fbm 동일), IBMPOS와 FBMPOS는 Inode 블록과 Fragment 블록에 대한 FreeBlock을 찾기 위한 비트맵이다. 페이지 크기는 해당 객체의 크기이며 블록번호는 해당하는 객체의 나머지 블록에 대한 블록의 시작위치를 나타낸다.

종전의 파일시스템에서 데이터 파일을 액세스하기 위해서는 Inode를 위해 디스크를 액세스하고 해당하는 파일의 위치 경로를 찾아 추가의 액세스를 요구한다. 하지만 본 메커니즘은 Inode와 데이터블록이 하나의 블록에 저장되어 있기 때문에 단 1회의 블록 액세스로서 해당하는 블록을 읽을 수 있어 디스크 액세스 횟수를 줄일 수 있다.

3.3 멀티스레드 비동기 DISK IO 기법

메인 프로세스(CacheDisk)가 다중 사용자 레벨의 스레드를 생성하여 이들이 비동기적으로 파일시스템을 액세스하도록 하는 것으로, 디스크에 대한 Read/Write등의 요구들을 요청(Request) 큐에 큐잉토록 하여 처리된다. 이 메커니즘은 메인 프로세스가 캐시 스레드를 생성하고 다음 요청을



(그림 3) 멀티스레드 디스크 IO 레이아웃

```
struct FileDesc {
    long key;
    md5_byte_t digest [DIGEST_LENGTH];
    long length;
    long start;
    FileDesc* next;
};
```

(그림 4) 메모리 객체의 참조를 위한 디렉토리 구조체

기다리고 있다가 다음 요청이 들어오면 새로운 스레드를 생성하여 넘기고 또다시 다음 요청을 기다린다. 이러한 일련의 작업을 수행하다 요청했던 디스크 i/o가 완료되면 디스크 스레드가 통보를 하게 된다. 이러한 메커니즘을 사용함으로써, 메인 스레드는 이전의 요구가 끝나기를 기다릴 필요 없이 수많은 양의 요청을 큐에 제출하여 무리 없이 처리할 수 있다. 메커니즘의 레이아웃은 (그림 3)과 같다.

3.4 Directory 구조체

저수준 파일시스템은 디스크상에 한 파일에 관한 모든 필요 정보를 메모리 상주 데이터 구조에서 관리한다. 이들 구조체에는 파일 크기, 파일 시작위치, 해시테이블에 대한고속 검색을 위한 정수 키값, 파일의 이름에 대한 MD5 해시값들이 (그림 4)와 같이 파일 기술자 요소에 저장된다.

모든 Inode에 대한 검색은 메모리 액세스만으

```
int sg_create(int sg_fd, unsigned char * buff, int blocks,
off_t to_block,
            int bs, int cdbsz, int fua, int * diop)
{
    sg_io_hdr_t io_hdr;
    memset(&io_hdr, 0, sizeof(sg_io_hdr_t));
    io_hdr.dxfer_direction = SG_DXFER_TO_DEV;
        // SCSI WRITE 명령, 사용자 메모리로부터
           디바이스로 이동
    io_hdr.dxfer_len = bs*blocks; // 사용자 버퍼크기
    io_hdr.dxferp = buff;  // 버퍼포인터
    io_hdr.pack_id = to_block;  // WRITE할 블록
    write(sg_fd, &io_hdr, sizeof(io_hdr)); // SG 드라이
      브에게 명령
    read(sg_fd, &io_hdr, sizeof(io_hdr)); // SG 드라이
      브로 부터 읽기
}
```

(그림 5) create 시스템 호출 함수

```
int sg_read(int sg_fd, unsigned char * buff, int blocks,
off_t from_block,
            int bs, int cdbsz, int fua, int * diop)
{
    sg_io_hdr_t io_hdr;
    memset(&io_hdr, 0, sizeof(sg_io_hdr_t));
    io_hdr.dxfer_direction = SG_DXFER_FROM_DEV;
       // SCSI READ 명령, 디바이스에서 사용자 메모
          리로 이동
    io_hdr.dxfer_len = bs * blocks; // 사용자 버퍼크기
    io_hdr.dxferp = buff;  // 버퍼포인터
    io_hdr.pack_id = from_block;  // READ할 블록
    write(sg_fd, &io_hdr, sizeof(io_hdr)); // SG 드라이
      브에게 명령
    read(sg_fd, &io_hdr, sizeof(io_hdr)); // SG 드라이
      브로 부터 읽기
}
```

(그림 6) read 시스템 호출 함수

로 이루어지므로 디스크를 탐색할 필요가 없다. EXT2는 파일의 Inode를 검사해야하기 때문에 EXT2보다 저수준 파일시스템은 최소한 1회의 디스크 입출력을 절약한다. 메모리 파일 기술자를 사용함으로써 발생하는 한 가지 주요한 단점은 파일 위치에 대한 모든 데이터는 시스템이 손상되는 사건이 발생할 때 상실된다는 것이다. 하지만, 전통적인 파일 시스템과는 달리 디스크상에 저장된 캐시파일은 오리지널 서버에서 항상 최신의 객체를 유지하고 있다는 것이다. 처음 브라우저가 오리지널 서버로부터 데이터를 받음으로써 속도는 로컬보다는 약간 못하겠지만 최신의 갱신된 객체를 받을 수 있다.

3.5 사용자 레벨 함수

3.5.1 create

create는 파라메터로 요청한 URL, 저장할 데이터를 위한 메모리 버퍼, 버퍼 크기로 구성된다. URL은 MD5 해시를 사용하여 다른 스트링으로 변경한다. 최대 저장 가능한 파일 크기는 캐시 구성 파일의 설정값 크기에 종속적이며 create 호출 함수는 (그림 5)와 같이 정의하였다.

3.5.2 read

read가 행하는 첫 번째 오퍼레이션은 적절한 파일 디스크립터를 위한 디렉토리 해시테이블을 탐색하는 것이다. 이것을 발견하면, 디스크상에 해당하는 파일의 시작 블록과 길이를 알 수 있다. read 역시 데이터가 실제로 디스크로부터 읽혀질 때까지 기다리지 않는다. 대신에 read는 요청을 생성하고 디스크 큐에 요청을 큐잉한다.

3.5.3 remove

remove 동작은 매우 간단하고 빠르다. 파일을 디스크에서 삭제하는 작업은 오로지 디렉토리로부터 적절한 파일 기술자를 제거하고, 프리 블록으로 설정하기 위해 디스크 비트맵을 변경하는 것이다.

4. 성능평가

본 연구에서 실험은 3대의 서버와 허브, 그리고 성능 평가툴을 이용하였으며 동일한 요청에 대한 응답시간에 대한 평가와 최대 처리 가능한 유효 응답수를 평가하였다.

```
Status remove (const char *name)
 {
   return keyRemove(genMD5(name, digest));
   /* return 0 : success , error code : failure */
 }
Status keyRemove (md5_byte_t *digest)
   Status status;
   FileDesc fd

  memcpy (&fd.digest, digest, DIGEST_LENGTH);

  // 파일 테이블 엔트리 삭제 및 비트맵 클리어
 if(disk->fileTable->remove(fd)
        || disk->requestQueue->removeQueueEntry(fd)
           == OK)
 {
     pthread_mutex_lock(&disk->bitMap->mutex);
     status = disk->bitMap->clear(fd.start, fd.length);
     pthread_mutex_unlock(&disk->bitMap->mutex);
 }
 else
     status = NO_FILETABLE_ENTRY;

    return status;
 }
```

(그림 7) remove 시스템 호출 함수

(표 1) 캐시 서버의 시스템 사양

용도 : 캐시 서버	
CPU	팬티엄 1Ghz 1개
운영체제	리눅스 커널 2.4.17
하드디스크	18G × 3개 디스크(7299RPM)
운영체제(/dev/sda)	/
캐시 데이터(/dev/sdb)	/cache_data1
캐시 데이터(/dev/sdc)	/cache_data2
디스크 컨트롤러	UltraWide SCSI

4.1 시스템 환경 및 구성

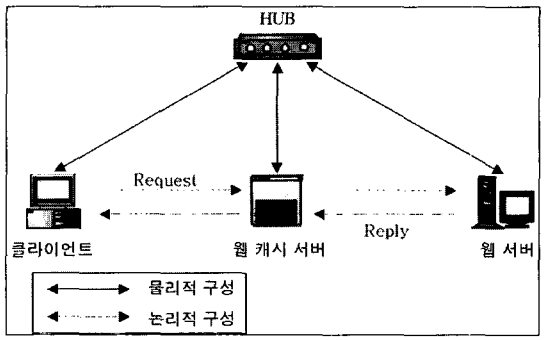
(표 1)은 캐시 서버의 시스템 사양으로 하드디스크 3개를 이용하여 1개의 하드에는 리눅스 운영체제를 위한 파티션으로 사용하였으며, 나머지 2개는 캐시 데이터를 각각 저장하기 위해 마운트(Mount)하였다. (표 2)는 클라이언트, (표 3)은 웹 서버의 시스템 사양이다. 추가로 12포트짜리 더미 허브를 이용하여 이들이 단절된 네트웍에서 외부 간섭이 없는 공정한 평가에 임할 수 있도록 구성하였다. 성능 평가 툴(Tools)은 웹 폴리그래프 2.5.4 버전을 이용하였는데, 본 시스템은 (그림 8)과 같은 구성을 갖도록 하였다.

(표 2) 클라이언트의 시스템 사양

용도 : 클라이언트	
CPU	팬티엄 850Mhz 1개
운영체제	리눅스 커널 2.4.17
하드디스크	9G × 1개 디스크(7200RPM)
디스크 컨트롤러	E-IDE

(표 3) 웹 서버의 시스템 사양

용도 : 웹서버	
CPU	팬티엄 850Mhz 1개
운영체제	리눅스 커널 2.4.17
하드디스크	9G × 1개 디스크(7200RPM)
디스크 컨트롤러	E-IDE



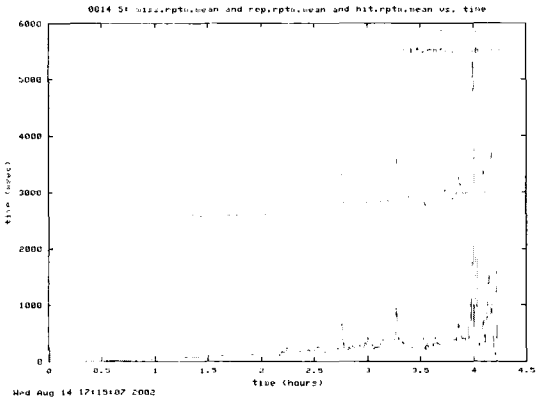
(그림 8) 성능평가를 위한 시스템 구성도

4.2 히트시간 평가

첫 번째 실험은 동일한 요청을 클라이언트에서 발생시켰을 때 캐시 서버를 거쳐 그에 대한 응답속도를 측정하였다. 본 실험으로 동일 환경에서 캐시로 얻을 수 있는 효과를 측정할 수 있다. (표 4)는 기존 시스템에서 동일 req/sec를 발생시켰을 때 응답시간과 캐시 서버에서의 히트율을 보여주고 있다.

(표 4) 동일 요청에 대한 히트율 비교

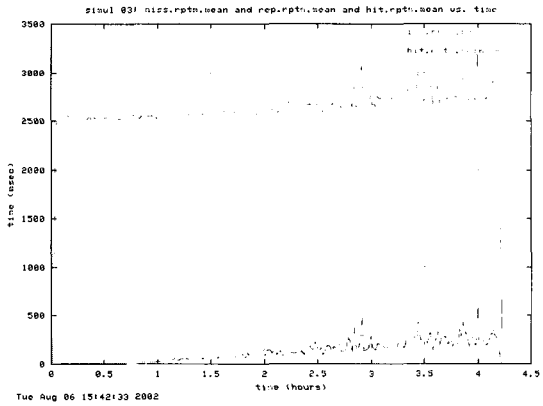
	기존 시스템	제안 시스템
Throughput	105.20 rep/sec	
Response time	1270.09 msec	1155.32 msec
- misses	2649.87 msec	2521.42 msec
- hits	132.97 msec	33.88 msec
Hit Ratio	54.82 %	54.92 %



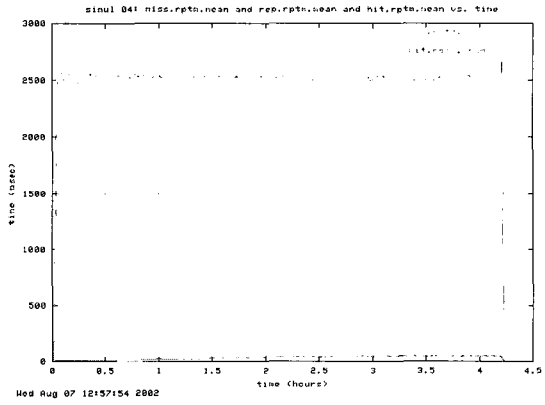
(그림 9) 요청 110 req/sec일 경우 응답시간 그래프(기존시스템)

(표 4)에서 보는 바와 같이, 동일한 요청인 105 req/sec에 대해 제안한 시스템이 약간 개선된 것을 볼 수 있으며, 캐시 미스(Cache Miss)일 경우의 시간은 캐시에서 데이터를 찾지 못하고 오리지널 웹서버로부터 객체를 읽어오는데 걸리는 시간을 의미하므로, 기존 시스템과 제안한 시스템간에 뚜렷한 격차를 보이질 않는다. 이에 반해 히트(Hit)일 경우는 캐시 서버에서 요청한 객체를 발견한 경우로 제안한 시스템의 파일 시스템에서 객체를 읽기 때문에 빠른 서비스로 클라이언트에게 서비스를 해줄 수 있었음을 알 수 있다. 히트율은 전체 웹 폴리그래프에서 공식적으로 발생시킨 히트율 55%와 유사한 값이 나와 평가는 유효함이 증명되었다.

(그림 9)와 (그림 10)은 110 req/sec과 105 req/sec에서 볼 수 있듯이 미스시간에서 급격히 응답시간이 증가함을 알 수 있다. 특히 시뮬레이션 시간이 마무리되는 시점에 가까워서는 응답시간이 급격히 나빠짐을 보여주고 있다. 이로써, 요청



(그림 10) 요청 105 req/sec일 경우 응답시간 그래프(기존시스템)



(그림 11) 요청 105 req/sec일 경우 응답시간 그래프(제안시스템)

수준이 시스템 한계치를 벗어나면 웹 폴리그래프에서 허용한 허용치를 초과하여 정상적으로 시뮬레이션을 완료하지 못해 리포트를 생성할 수 없다. 기존 파일시스템을 사용할 경우 시스템의 한계치는 110 req/sec 임을 결론 지을 수 있다.

(그림 11)은 평균 응답시간이 전체 평가시간에서 굴곡 없이 서비스되고 있음을 나타내고 있어, 기존 시스템보다 높은 시스템 처리율을 가질 수 있다는 것을 알 수 있다.

5. 결론 및 향후 연구과제

본 논문에서는 웹 캐싱의 특성을 다각도로 분

석하여 설계한 저수준 캐시 파일 시스템은 캐시된 객체의 크기와 액세스 권한 변경이 불가능하고, 오리지널 서버에 백업본을 반드시 소유하고 있다는 사실에 기인하였다. 오리지널 서버상에 백업본을 가지고 있다는 사실은 모든 메타 정보를 메모리에 저장하고 검사점 작업의 필요성이 없다는 것이다. 이는 캐싱 시스템에 이상이 있을 때 오리지널 서버로부터 데이터를 얻을 수 있기 때문이다. 이러한 특성으로 Direct 디스크 i/o에 의한 디스크에 액세스하는 어플리케이션에서 직접 디스크 버퍼(Disk Buffer)를 읽어옴으로써 Kernel에서의 디스크 캐시로 인한 오버헤드를 줄일 수 있으며, 웹 컨텐츠의 특성을 이용, 즉 컨텐츠를 생성하여 기록하는 시점에서 데이터 크기를 고정시킬 수 있는 특징을 반영하여 디스크 블록을 최대한 인접하게 위치시켜 디스크 액세스 속도를 개선할 수 있다. 이러한 설계를 바탕으로 성능 평가를 한 결과 EXT2보다 약 40% 가량의 성능 향상을 보였다. 본 캐시 파일 시스템 설계의 주 목적은 범용 파일 시스템과의 호환성은 없지만 웹 객체상에 있어 고속 운용을 제공하는데 있기 때문에 다른 유형의 파일 시스템과 호환성을 가질 필요는 없다.

향후 연구과제로는 디스크에 대한 늦은 동기화로 인해 발생될 수 있는 오류를 최대한 예방할 수 있는 방안과 점차 증가하고 있는 VOD와 같은 멀티미디어형 데이터의 스트리밍 서비스에서도 성능을 발휘하도록 확장된 연구가 필요하다.

참 고 문 헌

[1] 프로그램 세계 2001년 4월호 최철호, "웹 캐시 서버 Squid 완전 분석" pp. 192~217.

[2] 프로그램 세계 2001년 4월호 최철호, "인터넷 터보 엔진 웹 캐시" pp. 182-191.

[3] Douglas Gilbert, http://tldp.org/HOWTO/SCSI-Generic-HOWTO, May, 2002.

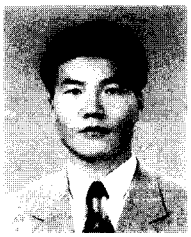
[4] Douglas Gilbert, http://www.torque.net/sg/p/scsi-generic_long.txt, March, 2001.

[5] Heiko Eieldt, http://www.torque.net/sg/p/original/SCSI-Programming-HOWTO.txt, May, 2002.

[6] http://kldp.org/Translations/html/The_Linux_Kernel-KLDP/tlk9.htmlan kldp.org, 1999.

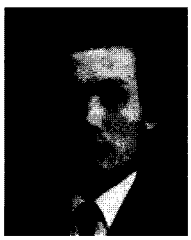
[7] Duane Wessels. Squid Internet Object Cache., http://www.squid-cache.org, May1996.

◐ 저 자 소 개 ◑



김 성 락

1984년 울산대학교 전자계산학과 졸업(학사)
1989년 한양대학교 대학원 전자계산학전공 졸업(석사)
2003년 수원대학교 대학원 전자계산학과 졸업(박사)
1996년~현재 : 오산대학 인터넷정보관리과 조교수
관심분야 : 분산 및 운영체제, 멀티미디어, 웹 시스템 등
E-mail : ksr01@osan.ac.kr



구 용 완

1976년 중앙대학교 전자계산학과 졸업(학사)
1980년 중앙대학교 대학원 전자계산학과 졸업(석사)
1988년 중앙대학교 대학원 전자계산학과 졸업(박사)
1983년~현재 수원대학교 정보공과대학장, 컴퓨터학과 교수
관심분야 : 분산 및 운영체제, 실시간 시스템, 시스템네트워크 관리, 멀티미디어, 인터넷 등
E-mail : ywkoo@.suwon.ac.kr