웹 캐시 서버의 디스크 성능 향상에 대한 연구

이 윤[0] 이대성 김기창
인하대학교 전자계산공학과
(youn[0],blue)@super.inha.ac.kr
kchang@super.inha.ac.kr

A Study on Improving Disk Performance On the Web Cache Server

Youn-Lee[0] Daesung-Lee Kichang-Kim
Dept. of Computer Science and Engineering, Inha University

요 약

인터넷 사용자에게 빠른 응답시간을 제공하기 위한 대안으로 웹 캐시 서버가 사용되고 있다. 그러나 웹 캐시 서버는 클라이언트의 처리 요청의 집중으로 인해서 디스크 I/O 병목현상이 유발되어 성능이 저하되는 문제점이 있다. 기존의 캐시서버 디스크 성능 개선 방안으로는 다중쓰레드, RAID 기술의 적용이 제안되었으나 효율성과 비용측면의 문제점을 갖고있다. 본 논문은 기존의 웹 캐시 서버, "스퀴드"의 오브젝트 저장구조와 디스크 접근방식을 변경하여 효율적이고 경제적인 디스크 성능 개선 방안을 제시하고자 한다.

1. 서 론

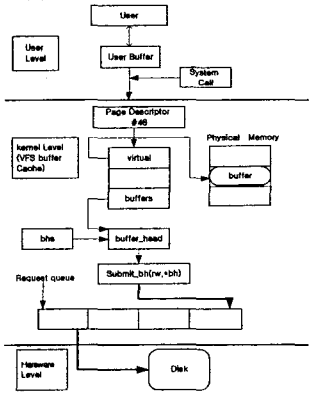
현재 실용화된 모든 캐싱 알고리즘들은 오브젝트들을 디스크에 저장하는 공통된 방식을 취한다. 그러므로, 캐시 서버에 대한 클라이언트의 요청이 많아질수록 디스크에 저장되어야 할 오브젝트들이 더 많아진다. 이로인해 길어진 디스크 큐(Queue)는 더욱 많은 디스크 I/O 요청을 일으키게 되고, 결과적으로 서비스 응답시간의 지연을 초래한다. 따라서, 웹 캐시 서버의 성능 저하의 주된 원인은 디스크 I/O 병목현상이라 하겠다. 실제로, 클라이언트의 높은 서비스 요청에 의한 디스크 I/O 병목현상은 버퍼메모리, 파일기술자 등의 부족을 초래하여 캐시 서버의 디스크 I/O 성능을 심각하게 저하시키는 문제점이 보고되어 있다[1].지금까지, 디스크 I/O 성능을 개선하기 위한 방안으로 다중쓰레드와 RAID를 웹 캐시 서버에 적용하는 방법이 연구되어졌으나, 다중쓰레드를 사용한 스퀴드(Squid)의 경우, 클라이언트의 요청이 폭주하면 디스크 I/O에 관여하지 않는 쓰레드가 나타날 때까지 서버가 대기해야 하는 문제가 발생한다[2].웹 캐시 서버의 디스크 성능향상을 위해 디스크 RAID level 0(striping)를 사용하는 경우, 성능 향상을 위해 더 많은 디스크를 사용할수록 사용 중인 디스크에 문제가 발생할 확률도 함께 높아지며, 고가의 비용이 드는 문제점이 있다[3].

본 연구는 효율적이며 경제적인 디스크 I/O 성능 향상 방안으로 '블록 단위 오브젝트 저장'과 'raw device 접근을 통한 디스크 I/O'를 제안하며, 구현 및 성능 평가를 위해 웹 캐시서버의 대명사격인 스퀴드를 사용한다. 본 논문의 2장에서는 스퀴드의 디스크 I/O 동작과 오브젝트 저장방식의 문제점을 지적하고 그 개선을 위한 기존의 연구를 소개하며, 3장에서 본 논문이 제안하는 개선된 캐시 서버의 설계에 대해 설명한다. 4장에서 실험결과와 그에 대한 고찰을, 5장에서 결론을 제시하고 향후 연구과제에 대해 논한다.

2. 관련 연구

2.1. 스퀴드 에서의 디스크 I/O [Linux 기반]

스퀴드에서 기존 파일시스템(Linux Ext2 File System)을 기반으로 한 디스크 I/O 동작은 디스크 에 직접적으로 이루어지는 것이 아니라 운영체제의 버퍼링을 통하여 이루어지게 된다[4]. User(스퀴드) 오브젝트 읽기 요청을 디스크에 보내면, 리눅스 운영체제는 우선 버퍼캐시에서 User가 요청한 논리적인 디스크 블록이 있는지 조사한다. User가 요청한 논리적 디스크 블록이 발견되지 않는 경우, 캐시미스가 발생하는데, 이때 리눅스는 버퍼캐시 내에 4Kbyte의 페이지 프레임을 할당하고 디스크 디바이스 드라이버에 디스크 I/O 작업을 요청하여 캐시미스가 발생한 블록을 디스크로부터 읽어들인 후 User 가 요청한 크기 만큼 사용자 버퍼에 복사한다. 또한 User가 오브젝트 쓰기 요청을 할때도 이와 비슷한 과정으로 버퍼캐시를 거치게 된다. [그림 1]은 리눅스 시스템에서 지원해주는 파일시스템을 기반으로 한 전반적인 디스크 동작원리를 설명하는 그림으로, 가상파일시스템에서의 페이지 프레임에 필요한 자료구조와 Submit_bh()에 의해서 디스크 I/O 요청이 만들어져서 디스크에 보내져 디스크 I/O 작업이 이루어지는 것을 볼수 있다.

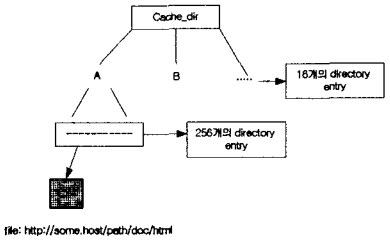


[그림 1]스퀴드의 디스크 I/O동작원리(Ext2 파일시스템 기반)

버퍼캐시는 그 용량이 한계에 이르면 리눅스 시스템의 Cache Replacement 정책에 의하여 새로운 공간을 확보하게 되며, 이 때 부가적인 디스크 I/O가 일어나 시스템의 부하가 높아진다는 문제가 있다. 스퀴드에 클라이언트의 요구가 폭주할 경우 이 문제는 더욱 심각하다.

2.2 스퀴드에서의 오브젝트 저장방식의 문제점

스퀴드에서는 [그림 2]과 같이 계층화된 디렉토리 구조에 파일을 생성하여 오브젝트를 저장하고 있다.



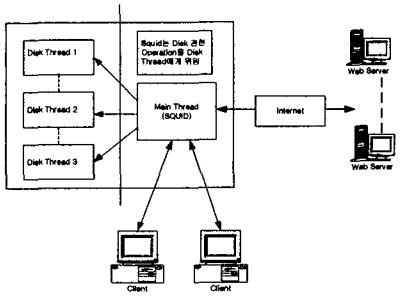
[그림 2]스퀴드의 오브젝트 저장형태

[그림 2]에서 보는 오브젝트 저장형태의 단점은 각각의 오브젝트를 파일에 저장해야 하기 때문에 파일을 열고,열어진 파일에 오브젝트를 쓰고,오브젝트를 파일에 쓴후에는 파일을 닫아야 하며 또한 저장된 오브젝트에 대해서 캐시히트가 발생되면 스퀴드는 클라이언트에게 서비스하기 위해서 오브젝트가 저장되어 있는 파일을 읽기 위한 동작을 수행하여야 하므로 파일에 관련된 많은 시스템콜이 호출된다.이러한 시스템콜 들은 스퀴드의 디스크 I/O 성능을 저하시킬수 있으며, 또 다른 문제점은 오브젝트들을 저장하기 위한 파일이 생성될시 파일을 생성하기 위해 필요한 물리적인 디스크의 블록들이 연속적으로 할당되지 않을경우 디스크의 탐색시간을 많이 요구하므로 디스크 I/O 성능을 저하시킨다.

2.3.변형될 캐시서버에 적용될 디스크 접근방식에 관한 연구

2.3.1 Thread를 이용한 디스크 접근 방식

웹 캐시 서버인 스퀴드 프로그램의 경우 다중쓰레드를 사용해 디스크 I/O 연산을 효과적으로 수행해 성능을 개선시키고 있다. 다중쓰레드를 사용한 스퀴드의 동작원리는 [그림 3]과 같다.

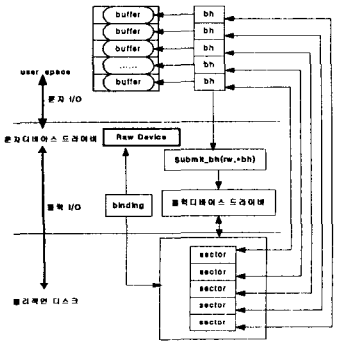


[그림 3]쓰레드를 이용한 디스크 접근 방식

[그림 3]에서 볼수 있듯이 쓰레드를 이용한 스퀴드는 캐시 미스 시에 웹 서버로부터 받은 패킷을 자신의 자식 쓰레드를 이용해 디스크 쓰기 연산을 위임 시키고 있다 또한 캐시히트 시에도 자신의 자식 쓰레드를 이용해 캐시된 데이터를 읽어 온후 스퀴드가 클라이언트에게 서비스하도록 하고 있다.이러한 디스크 접근 방법은 웹 캐시 서버의 병목 현상을 일으키는 디스크 연산을 자식쓰레드에게 위임함으로서 성능을 개선하려 하고있다.

2.3.2 Raw device를 이용한 디스크 접근방식

Raw device는 User와 하드디스크간에 문자단위 I/O를 가능하게 해주는 가상적인 문자 디바이스 드라이버이며 이러한 문자 디바이스 드라이버 파일에 들어있는 파일 Operation 즉 raw_open,read,write()등의 raw시스템콜 들을 이용해 User가 스퀴드가 기동되기전 디바이스 드라이버에 바인딩 된 물리적인 디스크에 연산을 행하게 된다.[그림 4]는 Raw device가 스퀴드에 적용될 때 전반적인 디스크 I/O 동작방식이다.이 그림에서 보는 바와 같이 데이터 전달시 파일 시스템을 거치지 않으므로 버퍼링(User와 커널메모리 페이지간에 데이터 복사과정)을 하지 않고 직접적으로 User프로세스 주소공간과 물리적인 디스크간에 data를 직접 전달하는 과정을 볼수있다.Raw device는 버퍼헤드(bh)들을 만들어서 user 공간에 할당 되어져 있는 buffer라는 구조체에 직접 매핑시켜 주는 역할을 하며 buffer(512byte:디스크 I/O의 기본단위)를 관리하는 각각의 버퍼헤드들이 Submit_bh()에 의하여 요청이 만들어져서 디스크 I/O 작업이 이루어지는 과정과 또한 raw device와 물리적인 디스크 간에는 블록단위 I/O를 하지만 User는 raw device를 이용해서 문자단위 I/O를 하는 과정을 볼수 있다.

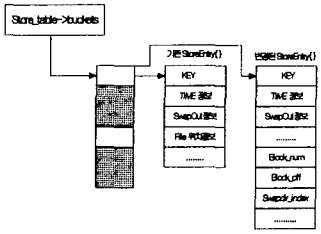


[그림 4]raw device를 통한 디스크 I/O 동작방식

3.변형된 캐시 서버의 설계

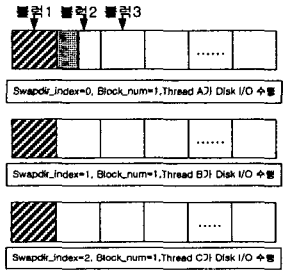
본 논문에서 제시하고자 하는 변형된 캐시 서버는 스퀴드 에서의 디스크 저장방식과 디스크 접근방식을 변형한 것이다. 디스크 저장방식은 변형된 캐시 서버 기동시 각각의 디스크를 하나의 파일로 간주하여 열었으며,Open된 상태에서 큰파일에 연속적으로 할당된 블록(크기 4kbye)에 저장, 스퀴드에서 디스크 I/O 성능저하의 원인이 되었던 시스템콜과 디스크 탐색시간을 줄였으며,디스크 접근 방식으로는 raw device를 적용하였으며 또한 멀티 쓰레드를 이용한 병렬처리를 하여 데이터 액세스 시간을 줄여 캐시 서버의 디스크 I/O 성능을 개선하려고 하였다.스퀴드 에서는 캐시 되어있는 혹은 앞으로 캐시될 오브젝트 마다 StoreEntry{} 구조체가 메모리에 할당되며 Uri와 Method(GET)를 MD5를 이용하여 해싱(Hashing),StoreEntry{}구조체의 KEY 값을 생성한다.변형된 캐시 서버에서는 블록 단위 오브젝트 저장이므로 클라이언트가 요구

하는 오브젝트를 디스크에 read/write시 요구하는 블록위치, 블록옵셋,물리적인 디스크의 위치를 변경된 캐시 서버가 파악해야 하므로 기존의 스퀴드의 StoreEntry{} 구조체를 변경하였다.[그림 5]는 기존 스퀴드의 StoreEntry{} 구조체와 변경된 캐시서버의 StoreEntry{} 구조체이다.



[그림 5]StoreEntry{} 구조체

[그림 6]은 변형된 캐시서버 에서 변형된 StoreEntry{} 구조체를 이용한 물리적 디스크 I/O 형태[물리적인 디스크는 3개(각각의 디스크의 블록크기를 4kbyte로 분할함) 15kbyte 오브젝트를 저장하는 경우]이다.



[그림 6]변형된 캐시서버에서의 디스크 I/O

5.실험 결과 및 고찰

본 실험은 본 논문에서 제시한 raw device를 적용한 변형된 캐시 서버(오브젝트 저장방식:블록단위 저장)와 오브젝트 저장방식이 파일단위 저장인 기존 스퀴드와의 비교 실험이다.이 실험은 동일한 시스템환경(운영체제:Linux, HDD:(4)IDE 하드디스크 16Gbyte,메모리:1Gbyte, Cpu:pentiumIII 700MHZ)에서 하였으며,공식적인 캐쉬 서버 벤치마크 프로그램인 Polygraph를 사용하였다.

	기존 스퀴드	변형된 캐시서버
Throughput	150.30rep/sec	250.75rep/sec
Response time -misses: -hits:	1412.45msec 2844.01msec 240.76msec	1355.67msec 2816.79msec 152.53msec
Hit ratio	54.99%	55.20%
Errors	0.00%	0.00%
Duration	4.00hour	4.00hour
Phases	top2	top2

[그림 7]Executive Summary(실험 결과 요약표)

디스크 성능의 신뢰도는 오브젝트 히트시 응답시간에 의존한다 [그림 7]에서 보면 오브젝트 히트시 변형된 캐시 서버의 응답시간이 기존의 스퀴드의 응답시간의 두배 이상 단축되었으며,또한 디스크 Throughput의 수치도 기존의 스퀴드의 1.5배 이상 상승했다.그러므로 변형된 캐시서버의 디스크 성능이 기존 스퀴드의 디스크 성능보다는 더 우수하다고 입증되었다.

6.결론 및 향후 연구과제

본 논문에서 변경된 캐쉬서버는 기존의 웹캐시서버(스퀴드)의 문제점을 파악하여 변형된 캐시 서버를 설계,구현 하였으며,그 결과 디스크 I/O 성능이 개선되었으며,향후 연구 과제로는 사용자의 서비스 요청에 대해 보다 빠른 응답시간을 제공하도록 캐쉬 서버의 히트 율을 높여야 하며 일반적인 서버들에서 나타나는 Network I/O 대기시간을 개선할 수 있는 연구들이 필요하다.

참고 문헌

[1]Alex Rousskov, valery Soloviev, "Eliminating the I/O bottleneck in Large Web Caches", Computer Science Department North Dakota State University

[2]Sachin More Alok Choudhary, "MTIO, A MULTI-THREADED PARALLEL I/O SYSTEM", in Proceeding of 11th International Parellel Proceeding Symposium , pp 368-373, April 1997.

[3]David A Patterson, Garth Gibson, and Randy H Katz, "A Case for Redundant Arrays of Inexpensive Disks(RAID)" in SIGMOD Conference pp 109-116, 1988.

[4]조유근, 최종무, 홍지만, "리눅스 매니아를 위한 커널 프로그래밍", 교학사