웹 캐시를 위한 다중 크기 블록에 기반한 디스크 관리 기법

한국과학기술원 황인석 · 이진원 · 조동호* · 송준화

1. 서 론

인터넷은 최근 짧은 시간동안 규모와 복잡도 양면에 걸쳐 매우 빠른 성장을 보이고 있다. end-user까지 도달하는 초고속 통신망의 보급은 더욱 많은 수요를 창출하였고, 그에 따라 이들을 서비스하는 서버의 수와 그 기능의 복잡도 역시 증가하고 있다.

이와 같은 양상은 서버와 클라이언트 사이의 트래픽을 증가시키고, 서버에게 가해지는 로드가 커지면서, 클라이언트에게 전달되는 응답시간의 지연을 낳게 된다. 이러한 질적 저하를 막기 위해 많은 수의 고성능 서버가 도입되고 네트웍 망의 고속화, 광대역화가 이루어져 왔으며, 이와 같은 자원의 확충과 더불어, 컨텐츠들의 복제 및 분산을 통해 특정 서버로의 request를 분산시키고 전체적인 트래픽의 양을 감소시킴으로써 보다 빠른 서비스를 제공하는 방법, 즉 CDN이 등장하고 있다.

CDN에서는 일반적으로 주요 ISP 네트웍에 캐시 서버를 설치하고 서버들의 컨텐츠를 각 캐시들에 복제하여 분산시켜 저장함으로써, 클라이언트의 요청시 해당 회선이 연결된 ISP 네트웍에서 바로 컨텐츠를 서비스하므로 보다 빠른 응답이 가능하다.

이러한 CDN에서 분산된 정보를 저장하는 역할을 수행하고 있는 것이 바로 캐시 서버이다. 최근에는 웹 트래픽이 전체 네트웍 트래픽의 대부분을 차지하게 되면서, HTTP 요청만을 전담하여 캐시하는 웹 캐시 서버들이 널리 사용되고 있다. 웹 캐시의 역할은 기본적으로 클라이언트로부터 URL의 형태로 요청을 받은 후, 그 요청이 가리키는 객체를 서버로부터 받아서 클라이언트에게 전달하거나 또는 웹 캐시 자신의 저장소로부터 그 대상의 사본을 읽어 클라이언트에게 전달해주는 것이다. 웹 캐시의 저장 공간은 한정되어 있으므로 보다 사용 빈도가 높은 객체를 저장할수록 평균 응답 시간을 줄일 수 있다. 그래서 보다 자주 사용되는 새로운 객체를 저장할 공간을 확보하기 위해 별로 사용되지 않는 기존의 객체를 지우는 작업이 일어나게 되는데, 이를 replacement라고 한다.

CDN과 같은 환경에서 웹 캐시가 클라이언트의 요청을 받아 hit이 일어났을 때의 트래픽은 웹 캐시와 클라이언트 사이로 국한되며, 이 경우 웹 캐시의 성능이 곧 클라이언트의 응답 시간에 큰 영향을 미친다. 일반적으로 웹 캐시는 웹 서버에 비해 상대적으로 단순한 작업들을 수행한다. 웹 서버의 경우 동적 페이지 생성, 보안, 세션 관리 등 복잡한 작업들이 많은 반면 웹 캐시는 요청받은 객체를 저장소, 즉 디스크 등으로부터 읽거나 쓰는 작업들이 대부분이다. 하지만, 많은 클라이언트와 많은 서버들의 중간에 위치하면서 이들을 중계하기 때문에 단순 디스크 작업의 빈도가 매우 많다는 특징을 가지고 있다.

이러한 이유로 웹 캐시의 디스크 관리 성능이 웹 캐시의 주요 성능 병목중의 하나로 지적되고 있다. 웹 캐시는 디스크 작업의 빈도가 매우 높을 뿐만 아니라, 웹 캐시에는 독특한 디스크 액세스의 특징들이 존재하기 때문에, 웹이 널리 퍼지기 이전부터 사용되었던 기존의 파일 시스템과 같은 구조는 이러한 특수한 환경에 적합하지 않다. 웹 캐시 hit 응답시간의 30% 가량이 디스크 병목에서 기인한다는 연구 결과도 있다[1].

2장에서는 웹 캐시가 가지는 디스크 액세스의 특징들을 파악하고, 기존의 파일 시스템들이 가지는 문제점들을 도출하였다. 또한 이와 같은 문제를 해결하기 위해 제안된 기존의 방법들을 소개하고 그 장단점들을 분석하였다. 3장에서는 다중 크기의 블록에 기

* 종신회원

반한 디스크 관리 기법을 제안하고, 그 구체적인 동작 원리 및 기존의 방법에 비해 가질 수 있는 장단점들을 기술하였다. 4장에서는 실험을 통해 본 연구에서 제안한 방법이 기존의 방법에 비해 우수한 성능을 가지고 있음을 보였으며, 마지막으로 5장에서는 결론을 내리고 앞으로 할 일에 대해 언급하였다

2. 웹 캐시의 디스크 액세스 특징 및 디스크 관리에 관한 연구

2.1 웹 캐시의 디스크 액세스 특징

웹 캐시는 일반 파일 시스템들과 달리 웹 객체들만을 저장하기 때문에, 달리 평균 수~수십kB 정도의 객체 크기를 보이고 있다. 또한 그 크기 분포는 작은 크기 영역이 매우 지배적이며, 수학적으로는 일반적으로 Pareto 분포로 모델링된다[2]. 이와 같은 사실은 그림 1의 "KAIST 웹 캐시의 객체 크기 누적 분포"에서도 뒷받침 되고 있다. 그와 더불어 일반적으로 웹 캐시, 특히 프락시 캐시는 hit ratio가 50% 미만인 경우가 대부분이기 때문에 디스크 read 보다 write의 비중이 높다[3, 4, 5, 6]. 마지막으로 웹 캐시는 일반 파일 시스템과는 달리 기존의 객체를 수정하거나 추가하는 등의 연산이 일어나지 않는다는 특징을 가지기 때문에 보다 단순한 system call set을 가질 수 있다.

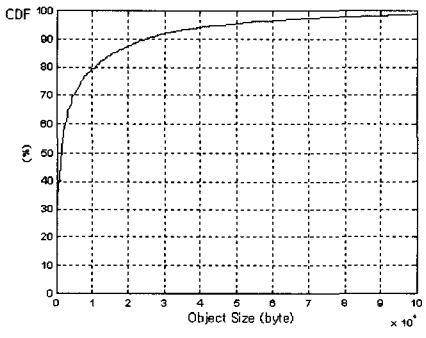


그림 1 KAIST 웹 캐시의 객체 크기 누적 분포

반면에 일반 파일 시스템들은 위와 같은 웹 캐시의 독특한 환경에 대해 최적화되어 있지 않다. 넓은 범위의 파일 크기를 지원할 수 있어야 하기 때문에 상대적으로 작은 크기의 객체들에 대한 최적화가 어려우며, 하나의 객체를 액세스하기 위하여 다단계의 테이블 검색이 필요한 경우도 있다. 또한 웹 캐시와는 달리 crash recovery가 필수적이기 때문에 crash에 대비하기 위한 복구 정보 등을 위해 부가적으로 디스크를 액세스하는 경우도 존재하며, 일반적으로 하나의 객체를 액세스하기 위해 다수의 system call이 사용됨으로써 빈번한 kernel로의 context switching을 일으킨다.

2.2 웹 캐시의 디스크 관리에 관한 연구들

웹 캐시의 디스크 병목 문제가 성능에 중요한 영향을 미친다는 것이 알려지면서 이러한 문제를 해결하기 위한 방법이 몇 가지 제안되었다. 먼저 복수의 디스크와 복수의 파티션을 이용하여 웹 캐시에 몰리는 디스크 요청을 분산시킴으로써 전체적인 성능을 향상시키기 위한 방법이 제안되었다[7]. 이 방법에서는 각 디스크와 파티션 별로 파일들을 효과적으로 배치하여 액세스에 필요한 시간을 단축하였으나, 하드웨어의 추가를 요구한다는 점에서, 주어진 자원의 성능을 최대로 끌어내지는 못하였다.

그리고 일반 파일 시스템을 사용하는 overhead를 피하기 위하여 웹 캐시 객체를 virtual memory page 별로 mapping하는 방법이 제안되었다[8]. 이 방법은 하나의 virtual memory page를 적절히 분할하여 복수의 웹 캐시 객체가 공유함으로써 효과적으로 디스크 공간을 사용하면서 복잡한 파일 시스템의 연산을 필요로 하지 않았으나, virtual memory의 주소 영역으로 캐시의 공간이 제약을 받았으며, page 크기를 넘는 객체에 대해서는 이와 같은 방법을 적용하기 어렵다는 단점이 있었다.

반면에, 디스크를 동일한 크기의 블록으로 나누고 각 웹 객체들은 서로 인접한 블록들에 걸쳐서 저장하는 방법도 제안되었다[9]. 여기서는 하나의 객체가 연속적으로 저장되므로 read, write가 빠르게 수행될 수 있었으며, 빈 블록을 관리하고 새로운 객체에게 블록을 할당하는 방법은 memory 관리에서 흔히 사용되는 free list가 도입되었다. 하지만 웹 캐시는 디스크의 사용된 공간이 증가함에 따라 빈번하게 replacement를 일으키는데, 여기서 매번 free list를 검색하고 갱신하는 작업이 일어나게 되며, free된 블록이 즉시 사용되기가 어렵기 때문에 결과적으로 디스크에 저장된 객체들 사이에 빈 블록들이 존재하는 fragmentation이 일어나게 된다. 이와 같은 fragmentation은 free list의 부담을 증가시키고 디스크의 사용률을 떨어뜨리는 결과를 가져오게 된다.

3. 다중 크기 블록 기반의 디스크 관리 시스템

여기서는 앞에서 언급한 웹 캐시의 디스크 액세스 환경에서 효율적으로 동작할 수 있는 디스크 관리 기법을 제안하였다. 본 방법에서는 무엇보다 replacement를 위한 기존 객체의 삭제와 디스크 공간의 확보과정의 단순화에 초점을 맞추었다. 일반적으로 디스크가 대부분 사용된 상황에서는 디스크 write를 수행하기 위해 replacement가 수반되는데, 특히 write가 read 보다 비중이 큰 프락시 캐시에서 이러한 replacement 과정의 단순화는 웹 캐시 속도 향상으로 이어질 수 있다. 이를 위해 본 방법에서는 기존의 객체가 삭제되면서 생성된 블록이 즉시 재사용될 수 있도록 하였으며, 여분의 빈 블록이 발생하지 않아 빈 블록들을 관리하는데 필요한 비용이 존재하지 않게 되었다.

제안된 디스크 관리 구조는 그림 2와 같다.

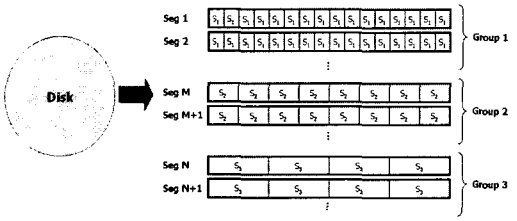


그림 2 제안된 디스크 관리 구조

그림 2에서와 같이, 디스크는 segment라는 기본 단위로 구성되며, 하나의 segment는 동일한 크기의 블록들로 구분된 연속적인 공간이다. 각 segment의 전체 크기는 모두 동일하나, segment의 블록 크기는 그림 2와 같이 segment별로 다를 수 있다. 동일한 블록 크기를 갖는 segment들은 group이라는 단위에 속하며, 전체 시스템에는 임의의 개수의 group 및 segment들이 존재할 수 있다.

본 방법이 가질 수 있는 replacement의 단순성은 기본적으로 각각의 웹 객체가 하나의 디스크 블록과 일대일 대응관계를 가진다는 것에 기인한다. 하나의 웹 객체가 복수의 디스크 블록을 사용하거나, 또는 하나의 디스크 블록이 복수의 웹 객체에 의해 공유되는 경우는 필연적으로 서로간의 할당 구조를 관리하고 갱신하는 작업의 복잡도를 상대적으로 증가시킨다. 따라서 결과적으로 가장 그 작업을 간단히 할 수 있는 것은 일대일 대응관계를 유지하는 것이다. 그런데 웹 객체의 크기는 dominant한 부분만 생각한다 하여도 1kB 미만의 크기에서부터 수십 kB 범위까지에 걸쳐있기 때문에, 한가지의 블록 크기만을 사용한다면 디스크 사용률이 심각하게 떨어지거나 지원할 수 있는 객체 크기를 매우 작게 제한할 수밖에 없다. 그래서 그림 2와 같이 다중 크기의 블록 구조를 제안하였으며, 여기서 디스크에 저장되는 웹 객체는 그 크기에 가장 알맞은 블록으로 저장된다. 예를 들어, 어떤 웹 객체의 크기를 α라고 하고, group 1과 group 2의 블록 크기를 각각 s_1, s_2라고 하자. 이때 $s_1 < \alpha \leq s_2$의 조건을 만족한다고 하면 해당 웹 객체는 오직 group 2의 블록으로만 저장이 된다. group 2의 모든 블록이 사용된 상태일 경우, group 2 내에서 replacement를 수행한다. 이 전체 과정을 그림으로 표현하면 그림 3과 같다.

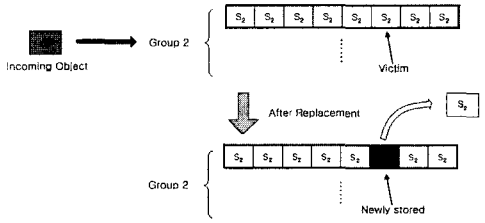


그림 3 Replacement 및 블록 할당 과정

그림 3에서 직관적으로 볼 수 있듯이, 새로운 객체에게 할당되는 블록은 기존 객체의 replacement 과정을 통해 자연스럽게 확보되며, 이 블록은 즉시 재할당됨을 알 수 있다. 즉, 블록 할당을 위한 별도의 연산이 필요치 않으며, replacement victim을 정하는 것만으로 모든 과정이 완료된다. 만일 디스크 블록과 웹 객체 사이의 일대일 대응 관계가 보장되지 못한다면 위와 같은 단순성이 유지될 수 없다.

여기서, 웹 객체의 크기로 곧 그 객체가 저장될 group이 결정되기 때문에, 해당 group의 모든 블록이 사용되었고 다른 블록에 여분의 블록이 남아있다 하더라도 다른 group의 블록을 사용할 수가 없다. 따라서 빈 블록을 확보하기 위한 replacement policy의 적용 범위가 각각의 group 별로 국한되게 되며, 각 group은 독립적으로 LRU(Least Recently Used) 또는 LFU(Least Frequently Used) list 등과 같은 자료

구조를 가지고 자신의 replacement를 수행한다. 이러한 경우 group을 지나치게 세분화할 경우 hit ratio를 떨어뜨릴 수 있기 때문에 group의 세분화를 통한 블록 사이즈의 다양화와 hit ratio 사이의 trade off가 필요하다.

또한 각 group 별로 공평한 replacement가 일어나게 하기 위해서는 segment의 적절한 분배 방법이 필요하다. 즉, 상대적으로 많은 요청이 일어나는 group에게는 보다 많은 segment를 할당해주고, 상대적으로 적은 요청이 일어나는 group에게는 보다 적은 segment를 할당함으로써 결과적으로 공평한 replacement를 일어나게 할 수 있다. 그런데 시간이 지남에 따라 각 group 별로 일어나는 요청의 비율에는 변화가 일어날 수 있고, 웹 캐시가 위치하는 곳의 환경에 따라 group 별로 요청의 비율이 다르기 때문에 run time에서 group 별 segment 수의 분배를 재조정할 수 있는 기능이 필요하다. 제안된 방법에서는 이러한 기능을 adaptation이라고 부른다. adaptation을 담당하는 소프트웨어 모듈은 주기적으로 각 group 별로 write 요청된 웹 객체들의 분포와 현재 group 별로 분배된 segment 수의 분포를 비교하여, 두 분포 사이의 불일치성이 일정 수준을 넘게 되면 segment 분배의 재조정을 시작한다. 재조정이 시작되면 먼저 현재 할당된 segment 수에 대한 요청의 비율이 가장 작은 group과 가장 높은 group을 선정하고, 전자로부터 하나의 segment를 후자로 이동시킨다. 이러한 과정을 거쳐 각 group 별로 공평한 replacement가 일어날 수 있도록 segment가 재분배되게 되며, 이 과정은 특히 웹 캐시 서버가 처음 시작되었을때, 웹 캐시의 group 별 segment 분배 상태를 실제 웹 객체 요청이 이루어지고 있는 환경과 유사하게 접근시키기 위하여 빈번히 일어나게 된다.

4. 시스템 설계 및 성능 비교 분석

4.1 시스템 설계 및 성능 측정 환경

3장에서 기술한 방법이 구현된 시스템의 구조는 그림 4와 같다. 웹 캐시의 디스크 관리 부분을 담당하는 구조이며, 상위 어플리케이션과 인터페이스를 담당하고 하위 group 별로 작업을 할당하는 dispatcher 모듈, 그리고 복수개의 group 모듈 및 각각의 group에 속하는 segment 모듈들로 구성되어 있

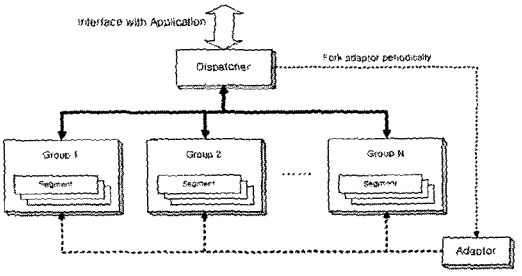


그림 4 System Block Diagram

다. 또한 주기적으로 adaptation을 위해 실행되는 adaptor 모듈이 존재한다.

기존에 제안된 방법들과 상대적인 성능 비교를 위해 웹 캐시에서 일어나는 환경과 유사한 실험이 수행되었다. 먼저 본 논문에서 제안된 방법인 그림 4의 구조, 그리고 2장에서 설명한 PMFLF에 기반한 구조, 그리고 Linux 파일 시스템을 그대로 사용한 구조가 실험 대상이 되었다. 공평성을 위해, 디스크 관리 이외의 replacement policy나 index 구조 등은 동일한 방법으로 구현하였다. 실험을 위한 입력 자료로는 실제 웹 캐시에서 얻어진 log를 이용하였는데, 이 log에는 웹 캐시가 클라이언트들로부터 요청받은 URL들이 시간 순으로 기록되어있으며, 각 URL에 해당하는 웹 객체의 크기 정보도 존재한다. 본 실험에는 www.ircache.net에서 배포하는 웹 캐시 log가 사용되었으며, 2002년 4월 2일~8일까지 1주일간의 log를 이용하였다. 본 log에서 얻어진 웹 객체들의 크기 분포는 그림 5와 같다.

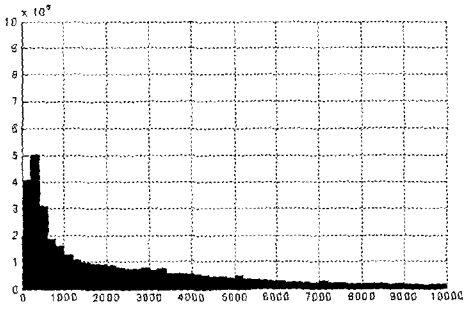


그림 5 웹 객체들의 크기 분포 (byte)

측정 대상 항목으로는, 먼저 초당 처리량이 측정되었는데, 이것은 웹 캐시의 디스크 처리 속도의 상

대적인 비교를 위한 것이다. 그리고 웹 캐시의 hit ratio 및 byte hit ratio가 측정되었는데, 이것은 여기서 제안한 방법이 가지고 있는 replacement policy 적용 범위의 제한이 미치는 영향을 알아보기 위한 것이다. 마지막으로 웹 캐시의 전체 동작 시간에 대한 CPU 점유 시간의 비율을 측정하였다. 일반적으로 OS는 특정 프로세스가 디스크 액세스 요청을 일으키면 CPU를 낭비하지 않기 위해 다른 프로세스에게 CPU 사용권을 넘긴다. 디스크의 하드웨어적 I/O 성능은 세가지 방법에 대해 동일할 것이므로, CPU가 점유된 시간의 비율이 높다는 것은 웹 캐시를 관리하는 방법의 복잡도가 높다는 것을 의미한다.

4.2 성능 비교 결과

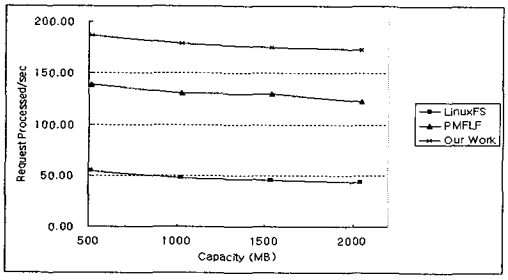


그림 6 초당 처리량

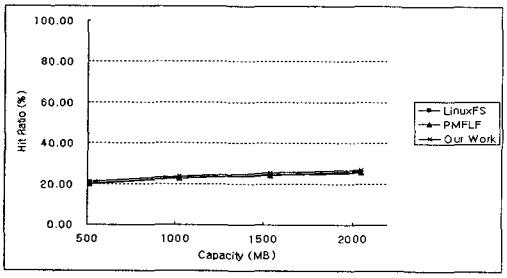


그림 7 Hit Ratio

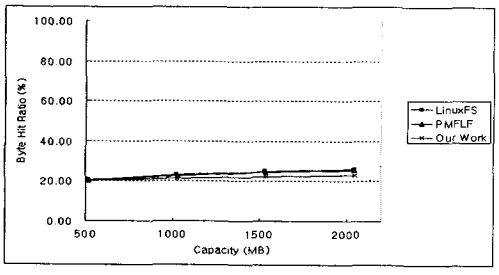


그림 8 Byte Hit Ratio

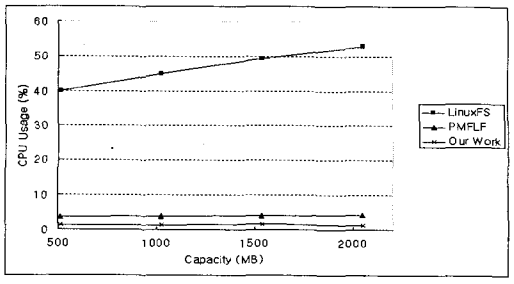


그림 9 전체 동작 시간대 CPU 점유 시간의 비율

먼저 세가지 방법에 대해 측정한 초당 처리량은 그림 6과 같다. 세로축이 초당 처리량이며, 가로축은 캐시 용량을 나타낸다. 세로축에 나타난 수치의 절대적인 값은 의미가 없으며 세 경우의 상대적인 비교에 의미가 있다. 그림에서 보면, Linux 파일 시스템을 그대로 사용한 경우가 가장 낮은 성능을 보이고 있으며, PMFLF에 기반한 방법이 그보다 약 150% 높은 성능을 보이고 있다. 여기서 제안된 방법은 PMFLF에 비해 약 30% 높은 성능을 보이고 있다. 이와 같은 결과는 다음의 그림 10이 보여주는 하나의 디스크 read 또는 write에 소요되는 시간을 측정한 결과로 설명될 수 있다. 여기서 read에 소요되는 시간은 평면적인 구조를 가지는 PMFLF에 비해 tree와 같은 계층적인 구조를 가지는 본 방법이 약간 높은 시간이 소요됨을 알 수 있다. 한편 write의 경우, 대부분 replacement를 수반하게 되며 하드웨어적으로도 read에 비해 많은 시간을 요구하기 때문에, 그림 10과 같이 read에 비해 매우 긴 시간을 가지게 된다. 그런데 이때 PMFLF와는 달리 블록 재할당을 위한 작업이 불필요한 본 방법이 약 30% 적은 시간을 필요로 함을 알 수 있다. 그림 7에서 볼 수 있듯이 hit ratio는 50%를 훨씬 밑돌고 있으며, 따라서 본 실험 환경은 write의 빈도가 read에 비해 매우 높은 환경임을 암시하고 있다. 이와 같은 웹 캐시의 환경에서 write에 소요되는 시간이 그림 10과 같이 약 30% 가량 적다는 것은, 결과적으로 그림 6과 같이 전체 성능이 PMFLF에 비해 약 30% 높게 표현되는 결과를 가져오는 것이다.

그림 7과 그림 8에서 각각 hit ratio와 byte hit ratio를 볼 수 있는데, 여기서는 세가지 방법이 거의 동일한 값을 보이고 있다. 즉 본 방법이 가지고 있는 replacement policy 적용 범위의 제한이 실제로는 큰 영향을 미치지 못하고 있다는 결과를 알 수 있다. 마

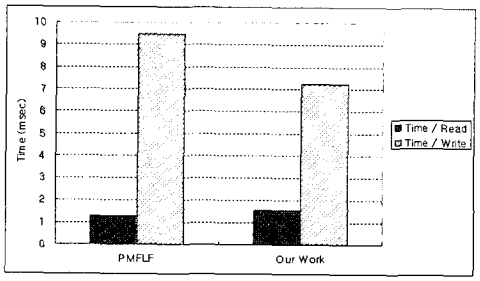


그림 10 객체당 평균 read/write 시간

지막으로 그림 9의 CPU 점유 시간의 비율을 보자.

예상했던 바와 같이, Linux 파일시스템을 그대로 사용한 경우는 빈번한 system call로 인해 많은 kernel로의 context switching을 유발하기 때문에 압도적으로 높은 CPU 점유시간 비율을 보이고 있다. 한편 나머지 두 경우는 상대적으로 매우 낮은 CPU 점유시간을 보이고 있으며, 이는 전체 웹 캐시의 동작 시간이 대부분 하드웨어적인 디스크 I/O에 소요되었음을, 즉 보다 효율적으로 디스크가 관리되었음을 의미한다. 이중에서도 본 논문에서 제안한 방법은 PMFLF에 기반한 방법에 비해 더 낮은 CPU 점유 시간 비율을 보이고 있다.

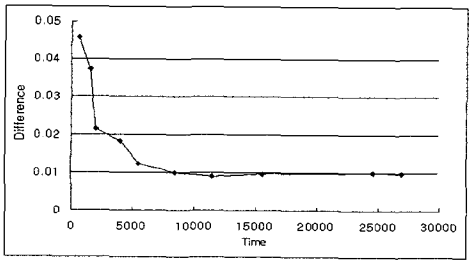


그림 11 시간에 따른 adaptation 과정

마지막으로, 그림 11은 본 방법에서 사용된 adaptation 기능이 동작하는 과정을 보여준다. 가로축은 실험 환경 속에서 웹 캐시가 동작한 이후로 흐른 시간이며, 세로축은 실제 요청된 웹 객체들의 크기 분포와 각 group별로 분배된 segment 수의 분포의 차이를 정량화한 것이다. 이 값은 두 분포를 각각 normalize한 후, 두 normalize 분포의 차이값을 구하여 제곱한 값으로, 0에 가까울수록 두 분포는 서로 일치하고 있음을 나타낸다. 예상한 바와 같이 웹 캐시가 처음 시작되었을 때는 두 분포의 불일치성이 상대적으로 높지만, 시간이 지남에 따라 빈번한 adaptation 과정을 거쳐 그 차이를 일정 수준 이하로 유지하는 상태에 도달함을 알 수 있다. 또한 전체 웹 캐시의 동작 시간에 대해 adaptation 모듈이 동작하는데 소요된 시간의 비율은 0.002% 미만으로서, 이 시간이 웹 캐시 동작에 미치는 overhead는 거의 무시할 만한 수준이다.

5. 결론 및 앞으로 할 일

본 논문에서는 최근 CDN의 등장에 따라 더욱 중요성이 높아지고 있는 웹 캐시의 성능을 향상시키기 위하여, 웹 캐시의 환경에 최적화된 디스크 관리 기법을 제안하였다. 이 방법은 비단 CDN 뿐만 아니라 웹의 전체적인 응답시간을 높일 수 있는 가능성을 제공한다. 제안된 방법에서는 웹에서 비중이 높게 분포하는 객체의 크기 범위를 지원할 수 있도록 다중 크기의 디스크 블록을 도입하였고, 웹 객체와 디스크 블록의 일대일 대응에 기초함으로써 디스크 관리에 필요한 작업량을 현저히 감소시켰고 특히 replacement 과정에서 일어나는 블록 재할당 과정을 단순화하였다. 디스크 관리 시스템은 복수의 group 및 복수의 segment로 구성된 계층 구조로 설계되었으며, replacement과정의 단순성을 유지하기 위해 새로운 객체가 자신의 크기에 알맞지 않는 group에는 저장될 수 없도록 함으로써 replacement policy의 적용 범위를 각 크기별 group 내부로 제한 시켰다. 캐시에 request되는 객체의 크기 분포에 따라 캐시의 group별 segment 할당량을 run-time에 변화시키는 adaptation 기법을 도입하였다.

그리고 기존에 제안되었던 방법과 실험을 통해 본 연구에서 제안된 방법의 우수한 성능을 보였다. 실제 웹 캐시에서 얻어진 log파일을 이용하여 실험용 request를 생성함으로써 실험의 사실성을 높였으며, 초당 처리량에 있어 본 연구에서 제안된 방법이 기존의 방법보다 30%까지 높은 성능을 보였다. 이러한 성능 차이는 예상했던 대로 replacement를 수반하는 write 과정에 소요되는 시간이 대폭 단축됨으로써 이루어 졌다. 한편, hit ratio 등의 다른 웹 캐시 성능 척도에 있어서는 제안한 방식이 기존의 방식과 거의 동일한 수치를 보임으로써 처리 속도 향상을 위해 다른 부분이 희생되지 않았음을 보여주었다.

앞의 실험 결과로부터, 본 연구에서 제안한 방법

의 replacement policy 적용 범위가 각 group별로 국한되어있지만 실제 캐시의 동작시에는 이러한 제한이 거의 영향을 미치지 못하는 것을 의미한다. 즉 이는, 통계적으로 서로 다른 group 사이에는 객체들의 popularity가 그리 연관성이 없다는 가설을 암시하고 있다. 따라서 이러한 가설을 정량적으로 입증할 수 있는 접근이 필요하다.또한 본 연구에서 제안한 adaptation이 실험을 통해 비교적 잘 동작하고 있는 것을 알 수 있었는데, 이는 웹 트래픽에서 객체들의 크기 분포가 시간에 따라 비교적 변화량이 적다는 가설을 암시하고 있으며, 이러한 가설을 정량적으로 입증할 수 있는 접근 또한 필요하다.

참고문헌

[1] A. Rousskov and V. Soloviev. On performance of caching proxies. In Proceedings of ACM SIGMETRICS, June 1998.

[2] B. Krishnamurthy and J. Rexford. WEB Protocols and Practice, pages 380-384. Addison-Wesley, 2001.

[3] S. Glassman. A caching relay for the world-wide web. In First International World-Wide Web Conference, pages 69-76. W3O, May 1994.

[4] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching proxies: Limitations and potentials. Available on the World-Wide Web at http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html.

[5] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for world-wide web documents. In ACM SIGCOMM, pages 293-305, Stanford, CA, August 1996.

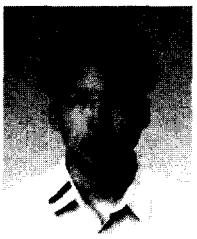
[6] M. F. Arlitt and C. L. Williamson. Web Server Workload Characterization: The Search for Invariants. In ACM SIGMETRICS '96, pages 126-137, Philadelphia, PA, May 23-26 1996. ACM SIGMETRICS, ACM.

[7] V. Soloviev and A. Yahin. File placement in a Web cache server. In Proceedings of ACM SPAA, July 1998.

[8] C. Maltzahn, K. Richardson, and D. Grunwald. Reducing the disk I/O of Web proxy server caches. In Proceedings of the USENIX Conference, June 1999.

[9] A. Iyengar, S. Jin, and J. Challenger. Efficient Algorithms for Persistent Storage Alocation. In Proceedings of the 18th IEEE Symposium on Mass Storage Systems, San Diego, California, April 2001.

황 인 석



1997~2001 한국과학기술원 전자전산학과 학사
2001~현재 한국과학기술원 전자전산학과 석사과정 재학중
E-mail:saber@nclab.kaist.ac.kr

이 진 원

1997~2001 한국과학기술원 전자전산학과 학사
2001~현재 한국과학기술원 전자전산학과 석사과정 재학중
E-mail:jcircle@nclab.kaist.ac.kr

조 동 호

1975~1979 서울대학교 전자공학과 학사
1979~1981 한국과학기술원 전기및전자공학과 석사
1981~1985 한국과학기술원 전기및전자공학과 박사
1985~1987 한국과학기술원 통신공학연구실 선임연구원
1987~1989 한국과학기술원 통신공학연구실 위촉연구원
1987~1998 경희대학교 전자계산공학과 조교수, 부교수, 교수
1989~1995 경희대학교 전자계산소 소장
1998~2001 한국과학기술원 전기및전자공학과 부교수
2001~현재 한국과학기술원 전자전산학과 교수
E-mail:dhcho@mail.kaist.ac.kr

송 준 화

1988 서울대학교 계산통계학과 학사
1990 State University of NY, Computer Science, 석사
1997 University of Maryland, Computer Science, 박사
1994~1996 IBM T.J. Watson Research Center, S/W Engineer, Digital Library Group
1996~1997 IBM T.J. Watson Research Center, S/W Engineer, Digital Media Solutions Dept.
1997~2000 IBM T.J. Watson Research Center, Research Staff Member, Parallel Commercial Systems Dept.
2000~현재 한국과학기술원 전자전산학과 조교수
E-mail:junesong@nclab.kaist.ac.kr