Journal of Digital Contents Society Vol. 9 No. 4 Dec. 2008(pp. 653-659)

웹 서비스를 위한 효율적인 캐쉬 관리 전략의 설계 및 성능 평가

문진용*

요 약

최근 인터넷의 폭발적인 인기로 인해 인터넷 객체의 캐쉥 기법이 매우 중요한 문제가 되었다. 인터넷에서의 캐쉥은 전통적인 캐쉥 기법과 여러 면에서 다르다. 특히 일반적인 캐쉥 알고리즘은 인터넷에 적합하지 못한데, 이는 크기가 서로 다른 객체들을 함께 처리함으로써 발생하는 작은 객체들의 불이익에서 기인한다. 본 논문에서는 인터넷 캐쉥을 위해 설계된 기존의 기법들을 살펴보고 새로운 알고리즘을 제안한다. 그리고 여러 가변 길이 대체 알고리즘에 대한 성능 평가의 결과를 제시하고, 객체들을 크기에 따라 구분하여 처리함으로써 바이트 적중률을 향상시키는 방안을 도출한다.

Design and Performance Analysis of an Efficient Cache Managing Strategy for Web Services

Jin-Yong Moon*

Abstract

With the recent explosion in using of the Internet, the problem of caching web objects has gained considerable importance. Caching on the Internet differs from traditional caching in several ways. Especially, the conventional caching algorithms are not well suited for the Internet caching. The poor performance is mainly due to its unfair treatment of small objects since all the objects are treated the same even though they differ in size. In this paper, I give an overview of caching policies designed for web objects, and provide a new algorithm of my own. I also have performed trace-driven simulations about variable-size replacement algorithms, and derived a new algorithm to improve byte hit-ratio by classifying objects based on their sizes.

Keywords : Byte Hit-Ratio, Caching Algorithm, Performance Evaluation

1. 서론

국내·외로 대중적인 인기를 확보하여 인터넷의 사용량이 급속도로 증가하고 있는 실정이다. 인터넷 사용자와 서비스 제공자의 기하급수적인 증가에 따라 동일 객체에 대한 중복 요청이 네트워크 대역폭의 상당부분을 차지하여 불필요하게 낭비되며, 일부 인기있는 서버로 부하가 집중되어 클라이언트의 지연시간이 길어지거나 접속 불능이 되는 상태까지 발생하고 있는 실정이다. 이러한 현상은 인터넷의 급격한 성장에 따라 심각한 장애로 인식되고 있다[1][2].

그 결과로 현재 인터넷 캐쉥에 대한 연구가 활발히 진행되고 있다. 캐쉥은 자주 요청되는 인터넷 객체를 저장함으로써 인터넷 성능을 전반적으로 향상시킨다. 캐쉥은 인기 있는 서버의 부하와 전체 네트워크의 트래픽을 감소시키며 인터넷 사용자에게는 캐쉥된 문서로 빠른 응답시간을 제공한다[2][4][7].

캐쉥은 네트워크의 여러 지점에서 구현할 수 있다. 본 논문은 인터넷 캐쉬 시스템에서 적용할 수 있는 일반적인 주기억장치 캐쉬 대체 알고리즘에 관한 것이다. 따라서 이 결과를 인터넷 서버, 프락시 서버(Proxy Server), 그리고 클라이언트 브라우저에 적용할 수 있다.

※ 제일저자(First Author) : 문진용
접수일:2008년 09월 18일, 완료일:2008년 11월 27일
* 극동정보대학 방송영상미디어과
jmoon@kdc.ac.kr

캐싱되는 위치는 다르지만 어떻게 한정된 인터넷 캐쉬 공간을 효율적으로 사용할 것인가라는 공통된 문제를 가지고 있다. 따라서 인터넷 캐싱 기법의 문제는 어떻게 한정된 저장 공간을 효과적으로 관리하는가에 달려있다. 그러므로 대체 알고리즘의 성능은 캐싱 기법의 중요한 성능 요소가 된다. 이를 위해, FIFO(First In First Out), LRU(Least Recently Used), LFU(Least Frequently Used), 그리고 SIZE와 같은 여러 대체 알고리즘들이 연구되어 왔다[2][3][9].

인터넷 캐싱은 파일 시스템, 가상 메모리 시스템과 같은 전통적인 대체 기법과는 다르다. 인터넷 캐싱에서는 가변 크기의 인터넷 객체를 지원해야 한다. 예를 들어, 인터넷 객체의 크기는 수 B(Byte)에서 수십MB(Mega Byte)까지 매우 다양하며 각 객체의 인기도 또한 매우 가변적이다[1]. 그 결과로 인기가 없는 매우 큰 객체의 삽입으로 인하여 크기는 작으나 매우 인기 있는 수많은 객체들이 제거되어야 하는 상황이 발생할 수 있다. 이 경우 인터넷 캐싱의 효율은 매우 떨어지게 된다.

본 논문에서는 인터넷 캐쉬를 관리하는 새로운 알고리즘을 제시한다. 모든 객체들을 하나의 캐쉬에 관리하는 것이 아니라 객체들의 크기에 기반하여 저장하는 부분을 논리적으로 달리하는 새로운 캐싱 기법을 제안한다. 따라서 각 부분에서 관리하는 객체들의 크기는 다소 동일해져서 기존의 알고리즘을 적용하는데 큰 기여를 하게 된다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 인터넷 캐싱 기법들에 대해 살펴보고, 3장에서는 제안한 알고리즘을 상세한다. 4장에서는 로그 파일 분석 및 소개된 알고리즘들의 성능을 평가 및 그 결과에 대해 논한다. 마지막으로 5장에서는 현재까지 수행되어온 본 논문의 성과에 대해 정리하고 앞으로의 개선방향에 대해 기술한다.

2. 관련 연구

2.1 구성 요소 및 동작 원리

인터넷 캐싱 시스템의 객체들은 클라이언트 브라우저, 프락시 서버, 그리고 원본 서버에 캐쉬될 수 있다. 클라이언트는 요청한 문서가 브라우저의 캐쉬에 존재하지 않으면 프락시 서버에 요청한다. 클라이언트로부터 요청을 받은 프락시 서버는 요청된 객체가 캐쉬되어 있는지를 검사하고 만일 존재하면, 클라이언트에 서비스를 제공한다. 만일 존재하지 않으면, 프락시 서버는 원본 서버에 요구하여 전송받은 객체를 자신의 캐쉬 공간에 저장한 후 클라이언트에 제공하는 일반적인 작동구조를 가진다[5].

현재 몇 개의 시제품이나 프로토타입이 나와 있는 상황이다. 이중 하비스트(Harvest) 캐쉬 서버는 콜로라도(Colorado) 대학에서 만들어진 시스템이다. 부모와 형제를 갖는 계층적 구조를 가지며 이들 간의 상호협력은 UDP(User Datagram Protocol)를 기반으로 한 ICP(Internet Cache Protocol)로 이루어진다. 현재 하비스트는 상업적인 목적으로 안정성과 성능을 대폭 향상시킨 하비스트 제품군과 좀더 지능적인 캐쉬를 구현하기 위해 새로운 기능을 계속 추가하며 연구를 위해 원시코드가 공개된 스퀴드(Squid) 인터넷 캐싱 시스템으로 구분된다.

2.2 대체 알고리즘

현재 인터넷 캐쉬를 위해 연구된 기존의 대체 알고리즘들은 다음과 같이 크게 3가지로 분류할 수 있다. 본 논문에서는 제안된 알고리즘과의 성능 평가를 하여 본다.

첫째는 LRU, LFU와 같은 전통적인 대체 알고리즘들을 가변크기 객체라는 특수성을 갖는 인터넷 캐싱 분야에 응용·적용하는 것이다. LRU 확장은 크기가 서로 다른 객체를 처리하기 위해 기존의 LRU 기법의 확장이라 할 수 있다[3, 5]. 이 기법은 새롭게 들어오는 인터넷 객체의 여유 공간을 마련하기 위하여 가장 최근에 사용되지 않은 객체부터 삭제하는 방식이다. 이렇게 하면, 삭제되는 객체는 0개에서부터 수많은 객체들이 캐쉬에서 삭제될 수 있다. 이 방식은 실제적으로 매우 낮은 성능을 보인다는 것을 4장에서 알 수 있다.

둘째는 키에 기반한 대체 알고리즘들이다. SIZE는 크기가 가장 큰 객체를 교체하는 기법이다. 인터넷 캐쉬에서는 하드웨어 캐쉬나 파일 시스템 캐쉬와는 다르게 캐쉬의 단위가 인터넷 객체이기 때문에 캐쉬할 객체의 크기가 가변적이

다. 따라서 크기가 큰 하나의 객체가 크기가 작은 여러 개의 객체들을 교체하는 경우가 발생한다[2]. 이 기법은 캐쉬된 객체 중에서 크기가 가장 큰 객체를 먼저 교체함으로써 이러한 문제를 개선한다.

LRUMIN은 LRU의 변형으로 LRU를 인터넷 캐쉥에 적용할 때 새로운 객체보다 큰 객체에만 LRU를 적용하여 여러 개의 작은 문서가 교체되는 것을 방지하는 기법이다[3]. 예를 들면, 캐쉬에 새롭게 들어오는 객체의 크기를 S로 하고 S보다 큰 여유 공간이 없다고 하면 다음과 같은 순서로 처리한다. S/2보다 큰 객체들 중에서 LRU로 제거하고, S/4보다 큰 객체 중 LRU 방식으로 여유 공간이 생길 때까지 이와 같은 기법으로 계속해서 제거한다.

셋째는 전송비용, 객체의 유효기간 등과 같은 비용에 기반한 알고리즘이다. GD(Greedy Dual)-Size 알고리즘은 전송비용/객체크기가 가장 낮은 객체를 캐쉬 공간에서 삭제한다.

3. 인터넷을 위한 캐쉥 알고리즘

본 장에서는 인터넷 캐쉥에서의 설계시 고려되어야 할 사항에 대해 살펴보고, 새로운 캐쉥 알고리즘을 제시한다.

3.1 고려사항

인터넷 객체의 캐쉥은 공간 지역성을 이용할 수 없다는 특징을 갖는다. 인터넷에서는 각각 객체 단위로 캐쉥됨으로 이러한 공간 지역성에 의한 캐쉬 적중은 기존의 파일 시스템과는 달리 전혀 발생하지 않는다. 인터넷 캐쉥은 전적으로 같은 문서에 대해 약간의 시차를 두고 발생하는 시간 지역성에 의존하게 된다.

3.2 제안된 알고리즘

분할된 캐쉬의 사용은 메모리와 처리기 사이의 대역폭을 증가시키기 위해 이미 컴퓨터 구조 분야에서 사용되고 있다. 예를 들어, 명령어와 데이터를 위한 부분을 나눈 캐쉬는 현재 대부분의 처리기에서 사용하고 있다. 이렇게 하면, 명령어 캐쉬가 데이터 캐쉬에 비해 높은 적중률을 갖는다고 조사되고 있다[8].

본 논문의 접근 방법은 인터넷 객체의 크기에 기반한 분할된 인터넷 캐쉥 기법이며 제안된 알고리즘은 (그림 1)과 같다. 분리된 캐쉥 기법의 원리는 참조의 지역성을 유지하며 각 분할된 부분 캐쉬의 적중률을 높이는 방법이다. 각 부분에 저장된 인터넷 객체는 같은 등급의 객체들 간에만 대체된다. 작은 크기의 인터넷 객체의 참조 지역성이 캐쉬에 없는 등급이 다른 크기의 큰 객체의 삽입으로 혼란 받는 일이 없어진다. 이 기법을 이용하면 각 분할된 부분의 캐쉬는 시간적 지역성이 크게 개선된다.

캐쉬에서의 분할된 캐쉬의 수, 각 분할 캐쉬의 크기, 객체를 지원하는 크기의 상한과 하한 값의 설정, 그리고 각 부분에서 사용되는 대체 알고리즘이 설계에 관련된 매개변수이다. 본 논문에서는 이를 위해 4.2절과 같이 실제 사용되고 있는 인터넷 서버의 로그를 이용하여 부하의 특성을 분석하여 다음과 같이 설정하였다.

캐쉬의 구조는 하나의 캐쉬를 논리적으로 3부분으로 나누고 각각을 MIN, NOR, MAX로 하였다. 각 분할 캐쉬의 크기는 MIN을 1/10, NOR을 2/10, 그리고 MAX를 7/10로 나누었다. 이는 하나의 통합된 캐쉬를 분할하여 사용하고자 할 때, 어떤 비율로 구분할 것인가는 중요한 문제이다. 당연히 바이트 적중률을 향상시키기 위해서는 인기있는 크기의 객체에게는 많은 부분을 할당하고, 그렇지 않은 부분은 적게 할당해야 한다. 이를 위해 본 논문에서는 사용자가 요청하는 회수와 전송된 객체의 크기 비율의 산술 평균을 구하여 중요도를 결정하였다.

각 부분이 지원하는 인터넷 객체의 상한과 하한의 범위는 로그 파일의 분석에 따라 인터넷 객체 중에서 가장 많은 요청이 있는 객체의 평균 크기인 1~10KB(Kilo Byte)의 인터넷 객체를 NOR이 지원하며, MIN은 1KB보다 작은 객체들을 관리하며, 그리고 MAX는 10KB보다 큰 객체를 지원한다고 설정하였다. 이를 통해, 각 부분에서 관리하는 객체들의 크기는 다소 동일해져서 기존의 알고리즘을 적용하는데 큰 기여를 하게 되며 본 논문에서는 각 분할 캐쉬는 새로 삽입되는 객체의 여유 공간을 마련하기 위해 LRU 대체 알고리즘으로 작동한다.

```
object request on a client;
if (requested object ∈ cache) then
service requested object;

else{
confirm object size;
if (requested object 〈 1KB) {
select victim with LRU algorithm in MIN cache;
delete victim from cache;
append requested object in MIN cache;
}
if (1KB ≦ requested object ≦ 10KB) {
select victim with LRU algorithm in NOR cache;
delete victim from cache;
append requested object in NOR cache;
}
else if(requested object > 10KB) {
select victim with LRU algorithm in MAX cache;
delete victim from cache;
append requested object in MAX cache;
} }
```

(그림 1) 제안된 알고리즘

본 논문의 알고리즘은 전술한 바와 같이 인터넷 서버 캐쉬, 프락시 서버, 그리고 인터넷 클라이언트인 브라우저가 관리하는 캐쉬에 적용할 수 있는 일반적인 주기억장치 캐쉬 대체 알고리즘에 관한 것이며 캐쉬의 동작구조는 다음과 같다.

캐쉬 관리자는 캐쉬에 저장된 인터넷 객체의 리스트를 관리한다. 사용자의 특정 객체에 대한 요구가 들어오면, 캐쉬 관리자는 요구된 객체를 리스트에서 검색한다. 만일 적중하면 요구된 객체를 인터넷 캐쉬로부터 클라이언트로 전송한다. 만일 적중하지 않으며, 관리자는 원본이 있는 인터넷 서버에 요청한다. 전송받은 객체는 그 크기에 따라 등급에 맞는 분할 캐쉬에 저장되게 된다. 만일 전송받은 인터넷 객체를 위한 여유 공간이 없다면, 바로 그 분할 캐쉬 부분에서 여유 공간이 생길 때까지 LRU 알고리즘으로 삭제할 객체를 선택하게 된다.

4. 로그 분석 및 성능 평가

이 장에서는 로그 분석 및 관련 연구에서 살펴본 기존의 LRU 확장, LRUMIN, SIZE들을 본 논문에서 제안한 알고리즘과 실제 운영 중인 인터넷 서버에서 수집된 요청 기록에 대해 적용한 성능 평가 방법과 그 결과에 대해 기술한다.

4.1 실험 환경 및 가정

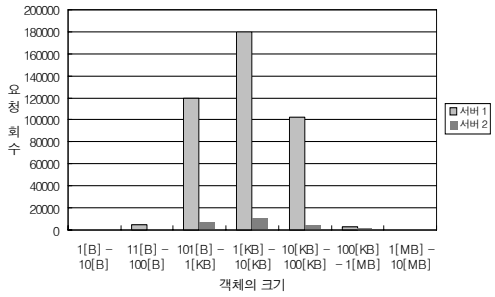
이 실험은 다음과 같은 실험 환경 및 가정에서 수행되었다. 우선, 각 객체를 저장 공간에 저장하는 방법에 따라 발생할 수 있는 단편화 현상은 전혀 없는 것으로 한다. 그리고 각 인터넷 객체의 변경은 전혀 없는 것으로 가정하였다. 이는 객체 일관성 유지 기법이 대체 알고리즘의 결과에 영향을 주는 것을 방지하기 위함이다.

본 논문에서 사용된 로그는 본 대학에서 공개적으로 운영하고 있는 메인 인터넷 서버(이하 서버 1)와 학과 홈페이지 서버(이하 서버 2)의 2008년 2월 한달 간의 요청 기록이다. 서버 1은 주로 학교 홍보, 수강 신청, 메일 서비스, 게시판, 그리고 교수의 학점 입력 등의 용도로 사용되고 있으며, 서버 2는 학과 소개 및 홍보, 알림판, 그리고 교수, 학생, 연구실들의 홈페이지 서버로 주로 이용되고 있다. 이들 인터넷 서버는 운영체제로는 RedHat Linux v9(Kernel v2.4.5)를 사용하고 있으며, 데이터베이스는 MySQL v 4.0.12를 이용한다. 웹 서버는 아파치 웹 서버 v 1.4를 사용하고 있다.

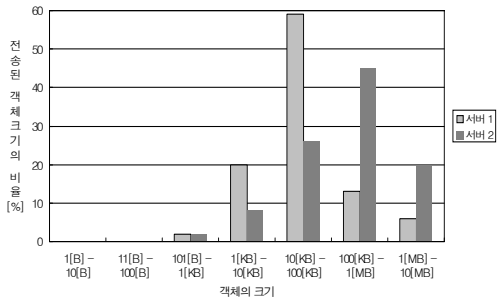
4.2 로그 파일 분석

서버 1과 서버 2의 요청 회수는 각각 약 70만 회와 4만 5천회이며, 총전송된 인터넷 콘텐츠의 양은 각각 약 3GB와 550MB의 요청이 포함되어 있다. 일일 평균 데이터 전송량은 각각 235,617KB와 17,654KB였다.

(그림 3)은 객체 크기별로 총전송된 전송량의 비율이다. 객체 크기별 요청 회수를 나타낸 (그림 2)와 비교하면 상대적으로 인기가 없는 큰 객체가 네트워크 트래픽의 상당부분을 차지함을 알 수 있다. 예를 들어, 1MB~10MB범위의 인터넷 객체들의 요청 회수는 각각 60회와 23회로 아주 미약하였으나 총전송된 크기에서 차지하는 비율은 각각 6%와 20%로 아주 높음을 알 수 있다. 그리고 가장 인기있는 객체의 크는 1~10KB 범위의 중간 정도 크기이다. 지나치게 작거나 큰 객체는 자주 요청되지 않음을 알 수 있다.



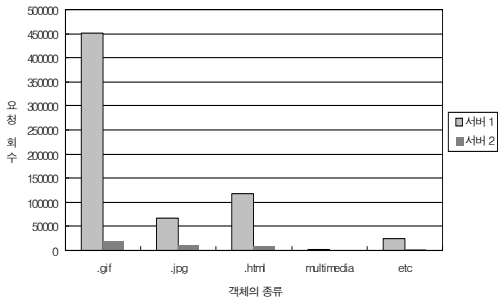
(그림 2) 객체 크기별 요청 회수



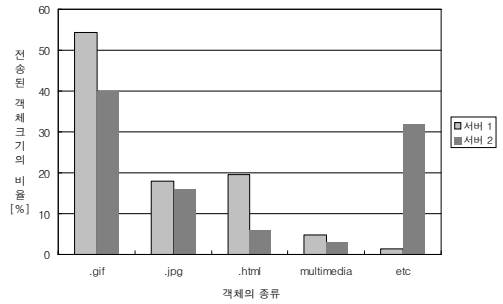
(그림 3) 객체 크기별 총전송량 비율

객체 크기의 분포를 이해하기 위해 실제 서비스된 콘텐츠를 구성하는 객체 포맷의 종류를 조사하였다. 이를 통해, 서버에 대한 사용자의 접근 패턴을 알 수 있다. (그림 5)는 객체 종류별로 총전송된 크기 비율로 요청 패턴을 분석하여 보면 mp3, mid, swf, mpeg 등과 같은 멀티미디어 포맷 데이터의 경우 요청 회수는 (그림 4)와 같이 각각 1033회와 101회로 아주 미약했으나 전송 비율에서 차지하는 비율은 서버 1과 서버 2에서 각각 약 5%와 3%로 아주 높음을 알 수 있다. 이것은 작은 객체들을 많이 요청함으로써 네트워크 트래픽이 증가할 수 있지만, 크기가 큰 객체들은 몇 번만 요청하면 통신량이 많이 증가함을 의미한다. 특별히, 서버 2의 경우 zip, cgi, hwp, exe와 같은 기타 포맷의 요청 회수는 1088회로 아주 작았으나 총전송량에서 차지하는 크기 비율은 31%로 상대적으로 높았다.

지금까지의 부하 분석이 다른 서버의 작업 부하와 정확하게 일치한다고 단정할 수는 없다. 특정 인터넷 서버를 이용하는 사용자의 경향이나 서버의 역할에 따라 다르기 때문이다. 하지만, 본 논문에서 사용된 로그에서 나온 결과들이 유사하다는 것을 볼 때, 다른 서버의 결과도 일반적으로 인기있는 파일은 중간 크기 정도의 객체이고, 크기가 상대적으로 작거나 큰 객체들의 요청 빈도는 낮은 정규 분포 형태가 되리라 예상된다.



(그림 4) 객체 포맷별 요청 회수



(그림 5) 객체 포맷별 총전송량 비율

4.3 성능 평가 결과

본 논문에서는 성능평가를 하기 위해서 SUN Sparc 에서 운영체제로는 Solaris 8을 사용하였고, 컴파일러는 gcc 3.2.2를 사용하였다. 일반적으로 인터넷 캐쉬 대체 알고리즘의 성능 평가의 척도로는 적중률, 바이트 적중률(Byte hit-ratio), 사용자 응답 시간, 그리고 전체 비용 등이 사용되고 있다. 인터넷 객체의 크기가 가변적인 성격을 가지므로 바이트 적중률은 기존의 캐쉥 알고리즘의 성능 척도와는 다르게 인터넷 캐쉥 시스템의 중요한 인수이다. 그러므로 바이트 적중률

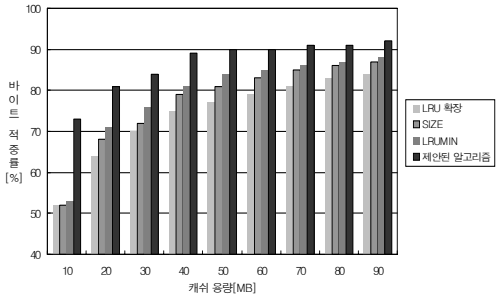
의 향상은 네트워크의 트래픽의 감소와 응답 시간의 감소에 중요한 영향을 준다.

바이트 적중률은 식 (1)과 같이 사용자가 요청한 전체 크기에서 적중한 객체 크기 양의 비율로 구해진다. 인터넷 캐쉬에서 바이트 적중률은 캐쉬를 사용함으로써 얻어지는 네트워크의 트래픽 감소와 응답 시간 향상을 나타내는데 사용된다. 하지만 인터넷의 특수성 때문에 응답 시간의 향상은 예측할 수는 있으나, 정확하게 정량화 하는 것은 불가능하다. 왜냐하면, 인터넷에서는 응답 속도는 객체의 크기뿐만 아니라 지리적 혹은 네트워크의 트래픽 상황에 따라 크게 영향을 받기 때문이다.

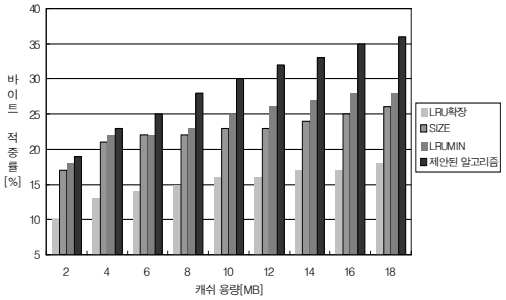
$$\text{바이트 적중률} = \frac{\sum_{i=1}^{n} b_i nhit_i}{\sum_{i=1}^{n} b_i nreq_i} \quad (1)$$

여기서, b_i는 i번째 객체의 크기이고, nh_it_i는 i번째 객체를 캐쉬로부터 응답한 회수, 그리고 $nreq_i$는 i번째 객체가 요청된 회수이다.

본 논문에서는 캐쉬의 크기를 달리하여 바이트 적중률을 비교하였다. (그림 6)과 (그림 7)은 각각 서버 1과 서버 2의 성능 평가의 결과를 나타낸 것이다. 우리가 제안한 알고리즘은 기존의 기법들에 비해 높은 바이트 적중률을 갖는 것을 알 수 있다. 객체의 크기에 비중을 둔 LRUMIN과 SIZE 기법들은 거의 비슷한 성능을 보였으며 LRUMIN이 SIZE 기법에 비하여 평균 약 2%정도 바이트 적중률이 높았다.



(그림 6) 바이트 적중률 비교(서버 1)



(그림 7) 바이트 적중률 비교(서버 2)

서버 1의 경우 약 80%의 바이트 적중률을 얻기 위해서는 LRU 확장을 적용하면 68MB 크기의 캐쉬가, SIZE에서는 49MB, LRUMIN에서는 39MB, 그리고 제안된 알고리즘에서는 20MB의 캐쉬 공간이 필요하다. 즉, 제안된 알고리즘을 사용하여 약 80%의 바이트 적중률을 얻기 위해서는 LRU 확장 보다 캐쉬 크기의 27%만 이용하면 동일한 결과를 얻을 수 있다. 따라서 73%의 캐쉬 공간을 절약할 수 있다.

이 그래프에서 LRU 확장이 가장 낮은 성능을 보임을 알 수 있다. 이는 기존의 다른 파일 시스템과 같은 대체 기법들에서는 캐싱되는 객체들은 페이지라는 고정된 크기를 가지고 있으나, 인터넷 캐싱에서는 인터넷 객체를 분할하여 기존의 고정 크기의 대체 기법을 사용하면 캐싱을 하는 아무런 유용성을 발휘할 수 없기 때문에 단지 앞으로의 사용자의 성향을 최적화한 LRU 확장 기법을 인터넷에서 그대로 사용하기에는 적합하지 않음을 알 수 있다.

5. 결론

본 논문에서는 인터넷 캐싱의 여러 가지 대체 알고리즘들에 대해 다루었다. 캐쉬를 논리적으로 3부분으로 나누어 처리하는 새로운 기법을 제안하였다. 또한, 성능 평가로 본 논문에서 제안한 인터넷 캐싱 대체 알고리즘이 다른 기존의 캐싱 알고리즘들에 비해 좋은 성능을 보인다는 것을 보였다. 인터넷 객체의 가변적인 크기 때문에, LRU 확장 기법은 다른 기법에 비하여 전반적으

로 성능이 떨어졌다. 제안된 알고리즘은 실제 사용중인 인터넷 서버를 기반으로 실제 성능 평가 결과에 의해 다른 기법에 비해 높은 바이트 적중률을 보임을 알 수 있었다.

향후 연구 과제로는 캐쉥 시스템의 성능을 향상하고, 인터넷 객체에서 순차 접근의 경향을 활용하기 위해 본 논문에서 다루지 않은 선반입 기법의 부가 비용과 그 효과에 대한 정형화 및 분석, 이에 근거를 둔 실용적인 선반입 기법의 개발은 특히 사용자 지연시간 감소에 큰 도움을 줄 것으로 예상된다. 또한, 여러 개로 분할된 캐쉬를 관리함으로 인하여 검색, 갱신 등에 소요되는 추가적인 복잡성에 대해 기존 알고리즘과 비교해 볼 필요가 있겠다.

참 고 문 헌

[1] C. Williamson, "On Filter Effects in Web Caching Hierarchies," ACM Trans. on Internet Technology, Vol. 2, No. 1, pp. 47-77, 2002.

[2] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox, "Caching Proxies: Limitations and Potentials," Proc. of the 4th Int'l World Wide Web Conf., 2002.

[3] L. Cherkasoba and G. Ciardo, "Role of Aging, Frequency, and Size in Web Cache Replacement Policies," Proc. of the 6th Int'l Symp. on Computers and Communications(ISCC'01), pp. 3-15, 2001.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the Implications of Zipf's Law for Web Caching," In 3rd Int'l WWW Caching Workshop, 2005.

[5] J. C. Bolot and P. Hoschka, "Performance engineering of the World-Wide Web : Application to dimensioning and cache design," Proc. of the 5th Int'l WWW Conf., 2003.

[6] K. Cheng and Y. Kambayashi, "Enhanced Proxy Caching with Content management," Knowledge and Information Systems, 2002.

[7] C. Maltzahn, K. J. Richardson, and D. Grunwald, "Performance issues of enterprise level Web proxies," Proc. of the ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems, pp. 13-23, 2004.

[8] D. A. Patterson and J. L. Hennessy, Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers, 1996.

[9] 김현섭, "웹 Caching을 위한 교체 알고리즘", 동국대학교 석사학위논문, 2006.

[10] 김희수, 황병연, "이동 컴퓨팅 환경에서 이동 호스트의 다치성 증대를 위한 선택적 캐쉬 일관성 유지 기법," 제10권, 제4호, pp. 655-660, 2005.

문 진 용

1998년 : 건국대학교 대학원 (공학석사)

2001년 : 수원대학교 대학원 (이학박사)

2001년~현 재 : 극동정보대학 방송영상미디어과 부교수

관심분야 : 데이터베이스, 인터넷, 멀티미디어 등