선호도 기반 웹 캐싱 전략
(Web Caching Strategy based on Documents Popularity)

유 해 영 † 박 철 ††
(Hae-Young Yoo) (Chel Park)

요 약 본 논문에서는 파일이 요청된 순간에는 파일의 선호도만을 조사하고, 일정 시간이 흐른 후에 선호도가 높은 파일들을 일괄적으로 캐싱하는 새로운 캐싱 전략을 소개한다. 이 전략에서는 일정 기간 동안 캐시의 상태가 변하지 않기 때문에 캐시 조작 비용이 거의 들지 않는 매우 효과적인 자료 구조의 사용이 가능하다는 장점을 가지고 있다. 뿐만 아니라, 국내외 5개 웹 서버로부터 수집한 log 파일을 대상으로 실험한 결과에 의하면, LRU를 사용하였을 때보다 오히려 캐시 히트율이 증가하였으며, 캐시 내 자료 재사용율도 증가하는 장점을 보여 주고 있다. 본 논문에서 제안하는 선호도 기반 게으른 캐싱전략(Popularity Based Lazy Caching Strategy)은 캐시에 포함되지 못한 파일의 선호도가 크게 증가하는 경우에 성능이 떨어지는 단점을 가지고는 있다. 그러나 이러한 경우는 자주발생하지 않으며, 웹 서버를 적응적으로 구현하여 해결할 수 있다.

키워드 : 웹 캐싱 전략

Abstract In this paper, we propose a new caching strategy for web servers. The proposed algorithm collects only the statistics of the requested file, for example the popularity, when a request arrives. And, at times, only files with higher popularity are cached all together. Because the cache remains unchanged until the cache is made newly, web server can use very efficient data structure for cache to determine whether a file is in the cache or not. This increases greatly the efficiency of cache manipulation. Furthermore, the experiment that is performed with real log files built by web servers shows that the cache hit ratio and the cache transfer ratio are better than those produced by LRU. The proposed algorithm has a drawback such that the cache hit ratio may decrease when the popularity of files that is not in the cache explodes instantaneously. But in our opinion, such explosion happens infrequently, and it is easy to implement the web servers to adapt them to such unusual cases.

Key words : web caching strategy

1. 서 론

웹 서버의 수나 웹 서버가 제공하는 정보의 양, 그리고 웹을 이용하는 사용자의 수가 매우 빠른 속도로 증가하면서 발생하는[1], 파일 검색에서의 지연을 줄이기 위하여 파일 요청후 응답을 받을 때까지의 지연 시간을 결정하는 주요 요소인 통신망에서의 지연 시간을 줄이거나 혹은 웹 서버의 성능을 개선하여 웹 서버가 파일을 송신하는 서버 처리 능력을 개선코자 많은 연구가 진행되어 왔다. 서버의 처리 능력을 향상시키는 방안으로는 컴퓨터 H/W적인 처리 능력을 증가시키는 방법[2], 캐싱이나 부하균등화 작업 등과 같이 S/W로 요구들을 효율적으로 처리하는 방법[3]등이 제시되었다. 서버에서의 캐싱 기법은 서버가 응답하는 시간을 줄임으로써 사용자가 파일을 요청한 후 파일을 받은 때까지의 시간을 줄이기 위해 매우 효과적인 방법으로써 캐싱 장소의 차이나 캐시에서의 교체 전략 등 다양한 관점에서 연구가 진행되었다[4][5][6][7][8][9][10][11].

웹 서버에서의 캐싱 전략들은 각각 어떤 파일을 캐싱할 것인가를 결정하는 기준에서는 그들 나름대로의 독특한 특징과 장단점을 가지고 있으면서 동시에 공통점도 가지고 있다. 가장 흥미 있는 공통점의 하나는 파일이 요청되는 순간에 비록 기준은 다르다고 하더라도 이 파일을 캐싱할 것인가를 결정한 후, 이 결정에 따라 캐시를 즉시 조작한다는 점이다. 즉, 파일이 요청된 순간

· 본 연구는 2001년 단국대학교 대학연구비의 지원으로 연구되었습니다.
† 종신회원 : 단국대학교 정보컴퓨터학부 교수
yoohy@dankook.ac.kr
†† 정 회 원 : 단국대학교 대학원 컴퓨터과학 및 통계학과
pcman@dankook.ac.kr
논문접수 : 2002년 3월 12일
심사완료 : 2002년 7월 26일

에, 이 파일을 캐싱하는 것이 유리한지 혹은 캐싱하지 않고 캐시를 현상태 그대로 유지한 상태에서 서비스를 하는 것이 유리한지를 판단한다는 점이다. 본 연구에서는 이러한 전통적 캐싱 기법들을 캐시에 파일들이 서비스되는 시점에 캐시에 들어가야 하는 지를 판단하고 이에 따른 행동을 취한다는 관점에서 서비스 시점 캐싱 전략, 즉 CST (caching at the service time)이라 분류하고자 한다. CST 캐싱 전략들의 단점은 캐시 조작의 성능을 높이기 위하여 효과적인 자료구조를 사용하고 있더라도 매 요청마다 캐시를 조작하기 위하여 상당한 자원을 낭비하고 있으며, 자료가 동적으로 삽입과 삭제가 자유로워야 한다는 점 때문에 검색 시간을 줄일 수 있는 매우 효과적인 자료 구조, 예를 들면, 이진 자료 검색 등을 사용하지 못하고 있다.

이에, 본 연구에서는 파일이 요구된 시점에서는 캐시에 관련된 판단을 하는 것이 아니라 파일들의 선호도 등 기본적인 캐싱 여부 판단에 유효한 정보만을 축적한 후, 일정기간 동안 조사된 이들 자료를 근거로 하여 파일들을 일괄적으로 캐싱하는 전략을 소개한다. 본 논문에서는 캐싱 여부를 판단하는 기준으로 선호도만을 고려하였다. 제안하는 이 전략은, 어찌보면 게으르게, 일정시간이 흐른 후에야 비로소 캐싱할 파일들을 결정한 후, 일괄적으로 캐싱하고, 그리고 새로운 결정이 내려질 때까지는 캐시를 변경하지 않으므로 선호도 기반 게으른 캐싱 전략(PLC:Popularity based Lazy Caching)이라 부르고자 한다. PLC를 사용하는 경우에, 캐시는 새로운 파일로 채워질 때까지 변하지 않는 read-only 캐시로 변하게 되어, 기존의 웹 서버에서 사용하는 선형 연결 구조보다 훨씬 검색이 빠른 자료구조의 사용이 가능해진다. 뿐만 아니라, 요청을 처리하면서 파일을 처리할 때마다 수행하였던 캐시 조작 시간이 없어지게 되어 웹 서버의 요청 처리 시간이 거의 무시할 수준으로 줄어들게 된다.

PLC의 성능을 평가하기 위하여 국내외 웹 서버가 생성한 log 파일을 대상으로 성능 평가 실험을 수행하였다. 비교 평가 캐시 교체 전략으로는 가장 대표적인 LRU(Least Recently Used)를 선택하였다. PLC는 캐시의 적중률이나 캐시내 자료 재사용율도 LRU를 사용하였을 때보다 증가하는 결과를 보여주고 있다. 다만, 캐시를 새로운 파일들로 교체하는 시간 간격보다 짧은 시간 내에 일부 파일에 대한 선호도가 급격히 변화하는 경우에 캐시의 적중률이나 캐시내 자료의 재 사용율이 떨어질 수가 있다. 그러나 이는 매우 특별한 경우에 해당할 뿐만 아니라 PLC 알고리즘 내에, 파일을 처리하면서 집계되는 선호도를 기반으로 선호도의 급격한 변화에 따른 처리를 구현함으로써 해결할 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 PLC 제안 배경에 대하여 기술하고, 3절에서는 PLC 알고리즘을 기술한다. 4절에서 국내외 유명 웹 서버의 log 파일을 대상으로 수행한 성능 평가 실험 결과를 기술하고, 끝으로 5절에서는 결론과 향후의 연구 과제에 대하여 기술한다.

2. 관련 연구

웹 서버에서의 캐싱은 웹 서버가 디스크에 있는 파일을 메모리에 캐싱하여 웹 서버가 클라이언트의 요청에 응답하는 요청 처리 시간을 줄이는데 목적이 있다.

클라이언트가 파일을 요청하면 웹 서버는 디스크로부터 파일을 메인 메모리의 버퍼로 읽어 들인 후에 클라이언트로 전송하게 된다. 웹에서 클라이언트에 의하여 요청되는 대부분의 문서들은 정적인 문서이기 때문에 웹 서버의 요구 응답 시간의 대부분은 디스크 입출력에 사용되어 진다. 따라서 요청된 파일이 메모리에 존재한다면 매우 빠른 요구 응답 시간을 보일 수 있다.

현재의 모든 운영체제들의 파일 시스템은 이미 디스크 캐시를 포함하고 있다. 그러나 기존의 디스크 캐시들은 웹 트래픽에 최적화 되어 있지는 않다[10]. 첫째, 캐싱의 단위가 다르다. 전통적인 파일 시스템의 캐싱 전략은 각각의 디스크 블록에 적용되지만 웹 서버는 전체 파일 단위로 필요로 하기 때문에 파일 단위의 전략이 필요하다. 둘째, 불필요하게 운영체제가 개입된다. 셋째, 웹 문서는 대부분 읽기 전용으로 사용되어 진다.

이러한 이유로 Markatos는 자주 요청되는 웹 문서를 웹 서버 프로그램의 메모리에 캐싱하는 메인 메모리 캐싱을 제시하였다[10]. 메인 메모리 캐싱은 자주 사용되는 문서를 웹 서버의 주소 공간에 캐싱한다. 웹 서버는 자신의 메모리 공간에 있는 캐시에 파일 시스템의 도움 없이 직접 접근하게 된다.

D. W. Chang, H. R. Ke, R. C. Chang는 adaptive-level 메모리 캐싱을 지원하는 웹 서버를 LRU 교체정책을 사용하여 FreeBSD Release 2.2.5에서 구현하고 유명한 웹 성능 평가 프로그램임 WebStone을 사용하여 구현된 웹 서버의 성능을 평가하였다[11].

Adaptive-level 메모리 캐싱을 현재 운영중인 웹 서버의 작업 부하를 사용하여 실험한 결과는 파일전체를 캐싱하거나 고정된 threshold을 사용하는 캐싱에 비하여 확연한 성능의 차이가 나타나지 않으나 크기가 큰 이미지 파일과 동영상 등의 멀티미디어 컨텐츠의 비중이 높아질수록 adaptive-level 메모리 캐싱은 효과적인 캐싱

전략이 될 수 있음을 보여준다.

3. 선호도 기반 게으른 캐싱 전략의 배경

다음의 표들은 한국 교육학술정보원에서 운영중인 교육 전용 웹 사이트 edunet4u를 구성하고 있는 서버 중 하나가 10일 동안(2000년 6월 7일부터 2000년 6월 16일까지) 생성한 로그파일을 분석한 결과를 보여주고 있다. [표 1]은 10일 동안 처리한 요청의 수 및 파일에 대한 통계를 보여주고 있다. 6월 15일 이외에는 매일 평균 900,000 정도의 요청을 처리하였음을 알 수 있다. 그러나 요청의 수가 2배 이상이었던 15일에도 요청된 내용면에서는 큰 차이를 보이지 않음을 알 수 있다. 예를 들면, 1회 이상 요청된 파일의 수는 약 30,000개로써 큰 차이가 없으며, 10회 이상 전송된 파일들은 불과 6,000여 개임을 보여준다. [표 2]는 10일간 서비스된 파일들이 얼마만큼이나 다시 요청되어서 서비스되었는가를 보여주고 있다. [표 2]에 있는 표를 배열 C라 할 때, 배열 C의 j번째 행, k번째 열인 C[j,k]는 j번째 날에 10회 이상 전송된 파일의 집합과 k 번째 날에 10회 이상 전송된 파일의 집합과의 공통집합의 원소의 개수를 가리킨다.

표 1 처리된 파일의 수

날짜	처리된 요청 갯수	1회 이상 전송된 파일수	10회 이상 전송된 파일수
7	906,000	29,917	6,739
8	928,171	32,805	6,536
9	900,587	32,106	6,387
10	809,550	33,626	5,652
11	882,184	30,928	6,365
12	942,388	32,346	6,336
13	912,588	30,983	6,200
14	967,942	29,820	6,576
15	1,909,410	29,440	9,756
16	909,109	29,555	6,258

표 2 10회 이상 전송된 파일의 재사용수

	8	9	10	11	12	13	14	15	16
7	5601	5442	4852	5333	5289	5328	5471	6145	5233
8		5498	4928	5186	5403	5320	5559	6233	5292
9			4876	5147	5230	5235	5424	6035	5248
10				4926	4885	4823	4933	5297	4757
11					5208	5121	5243	5942	4982
12						5275	5441	6062	5215
13							5455	5988	5198
14								6315	5409
15									5955

예를 들면 C[7,9]의 값이 5,442이라는 것은 7일에 10회 이상 전송된 파일 중 약 80.8%에 해당하는 5,442개의 파일이 9일에도 10회 이상 서비스되었음을 가리키고 있다. [표 2]는 어느 날 자주 요청된 파일들 중 많은 수의 파일들이 다음날에도 빈번하게 요청됨을 보여준다.

파일의 선호도를 사용하여 요청되는 파일들이 어느 정도로 다시 요청되는가를 나타내어 보자. $i-1$ 시점부터 i시점 사이에 기록된 파일 f의 요청 건수를 $A(i,f)$라 표현하고, $i-1$시점부터 i시점까지 처리한 총 요청건수를 $A(i)$라 표현하자. i시점에서 파일 f의 요청 건수의 상대적 비율, 즉 $A(i,f)$ 를 $A(i)$으로 나눈 값인 $p(i,f)$를 파일 f의 i시점에서의 선호도라 정의하자. 그리고 i시점에서 선호도의 순위가 k% 이하인 파일들의 집합을 $H(i,k)$라 표현하고, $i-1$시점부터 i시점까지 전송된 일부 파일들의 집합을 F라 할 때, F에 있는 파일 $f \in F$들의 선호도의 $p(i,f)$의 합을 $S(i,F)$로 표기하자. 예를 들면, $S(7, H(7,2))$는 시점 7에서 선호도가 2%이내인 파일들의 선호도의 합을 가리킨다.

$A(i, f)$=$i-1$시점부터 i시점 사이에 기록된 파일 f의 요청건수

$A(i)$=$i-1$시점부터 i시점 사이에 처리한 총 요청건수

$$= \sum_{f}^{all\ file} A(i, f)$$

$p(i, f)$=i시점에서의 파일 f의 선호도

$$= A(i, f) / A(i)$$

H(i, k)=i시점에서 전체파일 중 선호도가 높은 파일부터 선택하여 선호도의 합이 전체 선호도의 k%가 될 때까지 선택된 파일의 집합

$$S(i, F) = \sum_{f}^{f \in F} p(i, f)$$, F:$i-1$시점부터 i시점사이에 요청된 파일 중 일부 파일들의 집합

[표 3]은 HP01서버로부터 시점을 일별로 구별한 뒤 구한 7일부터 14일까지의 $S(i,H(i,1))$와 $S(i+1,H(i,1))$, 그리고 이들의 비인 $S(i,H(i,1))/S(i+1,H(i,1))$를 보여주고 있다. [표 3]에 기술된 실험 결과가 나타내듯이

표 3 선호도가 큰 파일의 선호도합의 변화정도 (%)

날짜	A=$S(i,H(i,1))$	B=$S(i+1,H(i,1))$	B/A
7	51.6	46.1	89.3
8	47.6	50.1	105.3
9	53.5	52.7	98.5
10	51.8	50.8	98.1
11	53.0	51.3	96.8
12	50.9	50.7	99.6
13	50.3	47.6	94.6
14	49.7	49.1	98.8

매일 선호도의 순위가 1% 이내인 파일들의 선호도의 합이 거의 50%에 이르고 있으며, 그 다음날에도 이들의 선호도의 합이 거의 비슷한 수준인 50%에 이르고 있음을 알 수 있다.

이러한 실험 결과들은 서버가 보유한 파일들의 선호도가 특별한 경우를 제외하면, 캐싱 여부를 고려해야 할 정도로 급격히 변하지는 않는다는 것을 의미한다. 이에 본 논문에서는 이러한 현상, 즉 과거의 일정 기간 동안 축적된 자료들로부터, 이들이 가까운 미래에도 많이 요청되면서 서버의 성능 향상에 기여가 예상되는 파일들을 추출하고 이들을 일괄적으로 캐싱하는 전략을 제안하고자 한다.

4. 선호도 기반 게으른 캐싱 알고리즘

4.1 캐싱 알고리즘의 모델링

본 절에서는 우선 캐싱 알고리즘의 목적을 해석적으로 살펴보고 이에 따라 선호도 기반 캐싱 알고리즘을 제시하고자 한다. 캐싱 알고리즘의 해석적 접근은 어떠한 캐싱 알고리즘을 적용하는 것이 유리한지를 판단할 논리적 기준을 제시하게 될 것이다.

A가 서버에서 적용되고 있는 캐싱 알고리즘을 가리킨다고 하자. 앞의 2 절에서 정의한 바와 같이 i 시점에서 파일 f의 선호도를 $p(i,f)$라 할 때, A알고리즘을 사용하고 있는 웹 서버에 파일 f가 요청되었을 때 캐시에 적중되는 비율을 가리키는 $C(i,f,A)$를 다음과 같이 정의하자.

$C(i,f,A)$=파일 f를 캐시에서 서비스한 경우의 수/$A(i)$

또한, i시점에서의 $C(i,f,A)$들의 합인 $C(i,A)$를 다음과 같이 정의할 때,

$$C(i,A)=\sum_{f}^{all\ file} C(i,f,A)$$

$C(i,A)$는 캐싱 알고리즘에 의한 캐시의 성능을 가리킨다. 따라서, $C(i,A)>C(i,B)$이면 알고리즘 A의 성능이 알고리즘 B의 성능보다 우수하다고 말한다. $C(i,f,A)$는 선호도인 $p(i,f)$를 사용하여 다시 표현하면 다음과 같다.

$C(i,f,A)$=(파일 f를 캐시에서 서비스한 경우의 수 /$A(i,f)$)*($A(i,f)/A(i)$)

$=R(i,f,A) * p(i,f)$

단, $R(i,f,A)$=파일 f를 캐시에서 서비스한 경우의 수 / $A(i,f)$으로써 파일 f가 요청된 횟수에 대한 캐시에서 서비스된 횟수의 비율을 가리킨다. 예를 들면, $R(i,f,A)$ =1은 파일 f가 요청되었을 때 마다 항상 캐시에서 서비스된다는 것을 의미하며, $R(i,f,A)$=0은 파일 f가 요청될 때마다 항상 디스크로부터 읽어서 서비스한다는 것을 의미한다.

[그림 1]은 에듀넷 사이트를 구성하고 있는 서버 중 한 서버에서 측정된 log 파일들을 LRU전략에 의하여 서비스한다고 가정하고 모의 실험을 수행한 결과이다.

선호도, 즉 요청 건수가 가장 많은 400개의 파일들의 $R(i,f,LRU)$, $p(i,f)$, 그리고 $C(i,f,LRU)$들을 선호도가 높은 파일부터 왼쪽에서 오른쪽으로 표시한 것이다. 400개는 하루에 요청되는 파일 중 약 1.09%에 해당하는 파일이며 전체 요청건수의 약 65%에 해당하는 파일들이기도 하다. [그림 1]의 제일 위에 있는 꺾은선은 $R(i,f,LRU)$를 나타내며, 가운데 놓인 꺾은선은 $p(i,f)$를 나타낸다. 그리고 제일 아래의 꺾은선은 $C(i,f,LRU)$를 나타낸다. 따라서 캐싱의 목표인 캐시 적중율은 제일 아래 그래프와 Y축, 그리고 X축이 만드는 영역의 면적이 된다. 일부의 파일은 선호도는 낮지만, 즉 요청 건수는 작지만 항상 캐시에서 서비스되는 비율이 매우 높음을 알 수 있다.

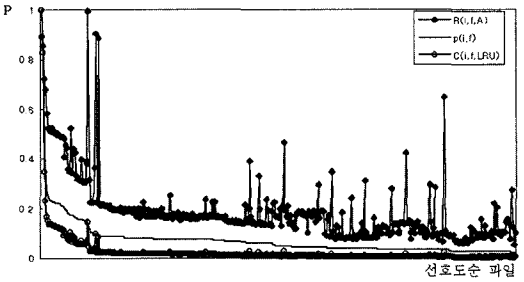


그림 1 HP01에서 측정된 log파일에 대하 선호도

캐시의 적중율을 높이는 것이 캐싱의 목표라면, 캐싱 알고리즘 A의 목적은 다음과 같은 최적화 문제로 표현된다.

$$z=\text{maximize} \sum_{f}^{all\ file} C(i,f,A)$$

$$=\text{maximize} \sum_{f}^{all\ file} R(i,f,A)*p(i,f)$$

위 수식의 정의에서 볼 수 있듯이, 캐시의 성능은 $R(i,f,A) * p(i,f)$에 의해 정의된다. 그리고 대부분의 캐싱 방법은 $R(i,f,A) * p(i,f)$의 값을 최대로 하기 위한 휴리스틱 알고리즘을 사용하고 있다. 예를 들면, 전통적인 *LFU (Least Frequently Used)* 전략은 캐시내에 있는 파일 중에서 액세스 횟수가 가장 작은 파일을 제거하는 것이므로 $p(i,f)$가 큰 파일을 캐싱하고 $p(i,f)$가 작은 파일을 캐시에서 제거하는 알고리즘으로 모델링이 가능하다. 그러나 LFU 알고리즘은 캐싱된 파일들의 $p(i,f)$의 값을 크게 할 수는 있으나 $R(i,f,A)$의 값까지,

더 나아가 그들의 곱이 크다는 것을 보장하지는 않는다. 또한, LRU에서는 최근 요청된 파일을 가능한 캐싱한다. 즉, LRU는 최근 요청된 파일이 다시 요청된다는 locality의 성질에 의거하여 파일들의 $R(i,f,A)$를 크게하는 것으로 모델링이 가능하다. 이 경우는, 요청된 파일이 다시 요청되므로 $R(i,f,A)$을 크게 만들 수는 있으나 $R(i,f,A) * p(i,f)$도 크게 한다는 보장이 없다. 이처럼 기존의 알고리즘들은 캐싱 알고리즘의 목적 함수의 값을 최대로 하기 보다는 목적 함수의 값을 최대로 하기 위하여 목적함수를 구성하고 있는 일부의 함수들의 값만을 최대로 하고 있는 것이다. 효과적인 캐싱 알고리즘이라면 R(i,f,A)와 p(i,f)의 곱이 최대가 되도록 하여야 한다. 본 논문에서는 선호도가 일정기간 크게 변하지 않는다면 p(i,f) 값만을 고려함으로써 캐시를 일괄적으로 변경하는 PLC 알고리즘을 제시하려 한다.

4.2 PLC 알고리즘

본 논문에서 제안하는 PLC 알고리즘은 [그림 2]와 같다.

```
Execute the following whenever a request arrives
    let f be the file in the request
    update caching effectiveness of f
    if f is in the cache then
        transfer the content reading from cache
    else
        read f from disk and transfer it to client
    check that the current time is the time to update
    cache
    if it is time to update cache then
        Get L that is the list of files with higher
        measure
        Destroy and build new cache with L.
        Reset measure for all files.
```

그림 2 PLC 알고리즘

PLC에서는 일정기간 서비스된 파일들의 선호도를 조사하여 선호도가 큰 파일들을 일괄적으로 캐싱하며, 파일이 요청되었을 때마다 캐시를 조정하지는 않는다. PLC에서는 일정기간 동안은 캐시의 내용이 변하지 않으므로 적중한 파일은 항상 적중하게 되고, 적중되지 않은 파일은 항상 적중되지 않는다. 따라서 캐시 내에 있는 파일에 대하여는 $R(i,f, PLC)=1$이며, 캐시내에 없는 파일에 대하여는 $R(i,f, PLC)=0$이다. 따라서 캐싱을 실행하는 시점에서 캐시의 크기가 결정되어 있을 때, 캐싱된 파일들의 $\sum_{f}^{all\ file} p(i,f)$을 최대로 하기 위하여 파일을 결정하는 문제와 동일하다. 즉, 캐시를 C라 할 때, 캐시의 크기를 size(C)라 하고, 파일 f의 크기를 size(f)라 할 때, $f \in C$인 파일들에 대하여 목적함수는 다음과 같이 표현된다.

$$z=\text{maximize}\{p \sum_{f}^{all\ file} (i,f) | f \in C\},$$

$$\text{단, } \sum_{f}^{all\ file} \text{size}(f \in C) \le \text{size}(C)$$

이를 매개 변수 x_f를 이용하여 다시 쓰면,

$$z=\text{maximize}\{ \sum_{f}^{all\ file} x_f\ p(i,f)\},$$

$$\text{단, } \sum_{f}^{all\ file} x_f\ \text{size}(f) \le \text{size}(C)$$

이 된다. 즉 서비스된 모든 파일들에 대하여 $\sum_{f}^{all\ file} x_f\ p(i,f)$이 최대가 되도록 x_f를 구하는 것이다. x_f는 파일 f를 캐싱할지 여부를 선택하는 플래그로 0 또는 1의 값을 가진다. $x_f=1$이면 파일을 캐시하는 것을 의미하고, $x_f=0$이면 파일을 캐시하지 않는 것을 의미한다. 파일 f이 선호도 $p(i,f)$를 a_f라 하고, 파일 f가 캐시에서 차지하는 비율 size(f)/size(C)를 b_f라 하면, 위 식은 다음과 같은 표기가 가능하다.

$$z=\text{maximize}\{ \sum_{f}^{all\ file} a_f\ x_f\}$$

$$\text{단, } \sum_{f}^{all\ file} b_f\ x_f=1,$$

$$\sum_{f}^{all\ file} a_f=1$$

$$x_f=0 \text{ or } 1$$

$$0 \le a_f \le a_{f-1} \le 1,\ b_f \le 1,\ i=1,2,3,\cdots.$$

이다. 문제의 편의를 위하여 캐시의 크기보다 큰 파일은 검토대상에서 제외하여도 좋으므로 $b_f \le 1$이라고 가정한다. 변수 x_f의 값은 파일이 캐싱이 되는가 아닌가에 따라 1 혹은 0의 값을 가지게 되므로 pure binary integer programming 에 속한다. 이 문제는 다음과 같은 linear programming relaxation이라 불리우는 linear Knapsack problem으로 변환하여 해결하면 간단해 진다[12].

위 문제는 r_f를 a_f / b_f으로 정의하고, 변수 x_f들을 r_f가 큰 값에서 작은 값 순으로 정돈한 후, 다시 기술하면 다음과 같다.

$$\text{maximize}\{ \sum_{f}^{all\ file} r_f\ x_f\ \}$$

$$\text{단, } \sum r_f\ x_f = 1,$$

$$0 \le x_f \le 1$$

이 문제의 최적해 x는 $x=(1,1,1,\cdots,1,(1-\sum_{f}^{f\in q})/r_p,0,0,\cdots.)$이며, r_p를 critical efficiency라 한다. 이를 기반으로 한 캐시에 캐싱될 파일을 찾는 방법은 다음과 같다.

Let C be the size of cache;
Compute r_f = $p(i,f)$ / size(f);
Let F={f_1, f_2, ⋯, } be list of files such that $r_i \le r_{i+1}$
Remaining=size(C);
L= ϕ ; k=1
While (size of the file fk ≤ Remaining) {
  Add f to L;
  Remaining=Remaining−size of (f_k)
  k=k+1;
}

4.3. PLC의 특징

기존의 연구들은 매번 요청이 발생할 때마다 교체정책을 수행하게 된다. 각각의 알고리즘은 나름대로의 시간 복잡도를 가진다. 가장 단순한 LRU 교체기법의 경우 O(1) 시간 복잡도를 가지며 LRU보다 높은 성능을 보이는 교체정책의 경우 이보다 큰 시간 복잡도를 가지고 매번 요청마다 서버에 부하를 준다. 실험적으로 높은 성능을 가지는 교체정책을 실제상황에 응용하려는 경우 이러한 높은 시간 복잡도는 새로운, 서버의 부하요인이 된다. PLC는 서버가 바쁘지 않은 시간에 캐시교체를 수행함으로써 서버의 부하를 덜어준다.

PLC의 성능은 캐시에 있는 파일들의 선호도의 합인 $\sum_{f}^{all\ file} p(i,f)$의 값에 의하여 결정된다. 그리고 임의의 알고리즘 A 의성능은 서버가 서비스한 모든 파일에 대하여 $R(i,f,A) * p(i,f)$의 합인 $z= \sum_{f}^{all\ file} R(i,f,A) * p(i,f)$의 값에 의하여 결정된다. 이 수식에 사용된 $R(i,f,A)$와 $p(i,f)$ 중에서 파일들의 선호도인 $p(i,f)$는 클라이언트의 선호성향에 따라 결정됨으로 캐싱 알고리즘과 서로 독립이다. 따라서 남은 항인 $R(i,f,A)$가 캐싱 알고리즘에 의존적인 부분으로써 PLC와 타 알고리즘 사이의 성능을 좌우하는 요소가 된다.

5. 실험 및 평가

5.1 실험환경

본 논문에서는 제안한 선호도 기반 캐싱 전략의 성능을 평가하기 위해 KERIS에서 운영중인 국내의 대표적인 교육 정보 포털 사이트인 에듀넷 (www.edunet4u.net)에서 운영중인 3개 웹 서버의 로그 파일과 1995년에 사용된 NASA와 CLARKNET의 웹 서버의 로그 파일을 사용하였다. 분석에 사용된 에듀넷의 각 서버를 편의상 각각 HP01, HP05, EDU01로 부른다. NASA 는 미국 플로리다에 위치한 NASA의 웹 서버의 로그이고 CLARKNET은 워싱턴 DC와 볼티모어 지역의 인터넷 제공자인 Clarknet사의 웹서버의 로그 파일이다. 이들 HP01, HP05, EDU01, NASA, CLARKNET의 로그 데이터에 대한 수집기관과 컨텐츠, 총접근 수, 총전송량은 [표 4]에 게재 되어있다. [표 5]는 실험에 사용된 로그 파일의 규모를 보여준다.

표 4 실험에 사용된 로그 데이터

표 4-1 에듀넷 로그 데이터

	HP01	HP05	EDU01
수집기간	2000/07/22 ~ 200/08/004(2주간)		
컨텐츠	평생교육	사이버학습교재	사용자 인증 및 포탈
총접근수	12,233,842	18,717,312	119,573,369
총전송량	187,087MB	276,029MB	576,365,605,969MB

표 4-2 기타 로그 데이터

	NASA	CLARKNET
수집기간	1995/07/01 ~ 07/28	1995/08/28 ~ 09/03
컨텐츠	NASA 웹 서버	Clarknet 웹 서버
총접근수	1,888,467	1,653,239
총전송량	38,612Mbytes	14,445MBytes

* 총 접근수 : 수집 기간 중에 웹 서버에 접근한 총 횟수
* 총 전송량 : 수집 기간 중에 웹 서버에 의하여 클라이언트에 전송된 데이터의 총량

표 5 실험에 사용된 로그의 일평균 통계

서 버	수집기간	접속자수	요 청 수	전 송 량
HP01	1	14,036	873,846	13,363,361,949
HP05	14일	7,878	1,336,951	19,716,394,674
EDU01	14일	55,586	8,540,955	41,168,971,855
NASA	28일	4,683	67,445	1,379,005,634
CLARKNET	7일	16,387	236,177	2,063,591,876

5.2 서비스된 파일의 재사용성 분석

[그림 3]는 선호도가 높은 파일들의 선호도의 합의 변화 모습을 보여주는 그래프이다. 그래프에서 X축은 $S(07/22,\ H(07/22,\ x))$를 의미한다. Y축은 $S(\text{다른날},\ H(07/22,\ x))$를 의미한다. 에듀넷 웹서버의 경우, 실험은 첫째 날에서 선호도 순으로 상위 100개 파일의 선호도부터, 100개씩 증가하면서 상위 2000개 파일까지의 선호도가 다른 날에 가지는 선호도를 조사하였다. 이렇게 조사된 180개의 $S(07/22,\ H(07/22,\ x))$와 $S(\text{다른날},\ H(07/22,\ x))$의 쌍을 산점도로 그린 그래프가 [그림 3]이다. NASA와 CLARKNET 웹서버도 같은 방법으로 실험하였다.

그래프에 나타난 점들은 모두 Y=X의 직선의 주변에 위치하고 있어 어떤 날의 선호도의 누적이 다른 날에도

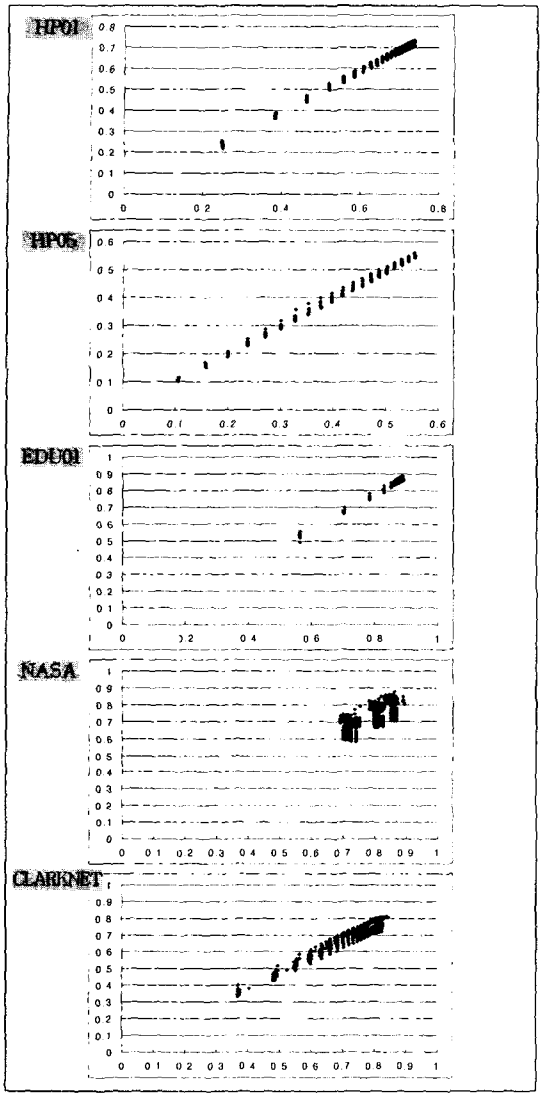
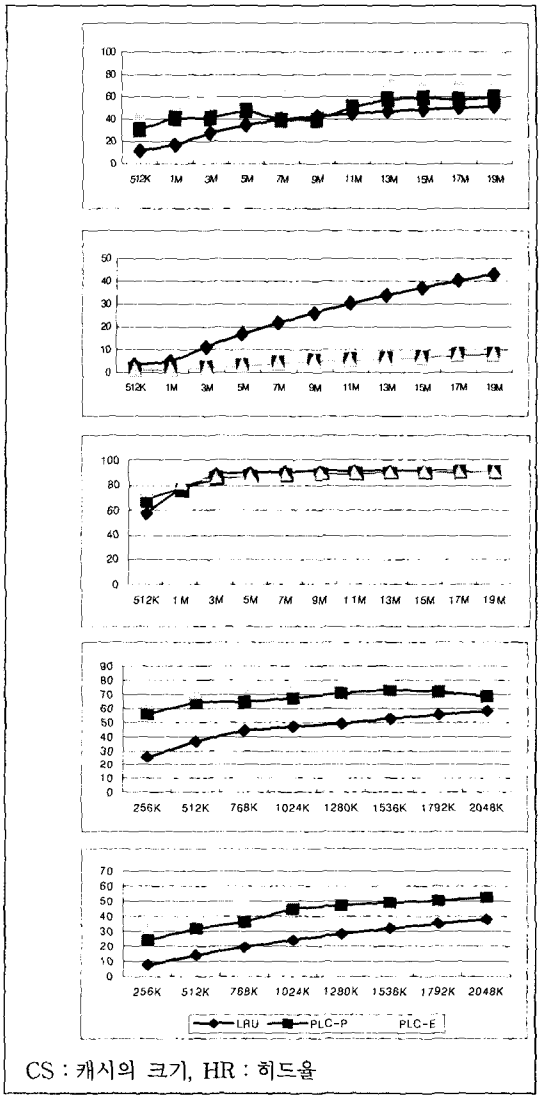


그림 3 재사용성 분석



CS : 캐시의 크기, HR : 히드율

그림 4 CS에 따른 LRU와 PLC 성능 평가표

유지되고 있음을 보여준다.

5.3 PLC의 성능 분석

3장에서 제안한 PLC에서 캐시에 들어갈 파일을 선택하는 기준으로써 효율성($r_f=a_f/b_f$)대신에 파일의 선호도(a_f) 만을 고려하는 방법도 가능하다. 비교 평가하기 위하여 파일의 선호도(a_f) 만을 고려하는 PLC를 PLC-P로 부르고 효율성($r_f=a_f/b_f$)을 고려하는 PLC를 PLC-E로 부르기로 한다. 실험은 각각의 서버별로, 전통적인 LRU를 사용한 캐시와 3장에서 제안한 PLC-P, PLC-E를 사용한 캐시에 대하여 캐시의 크기를 변화시키면서 캐시의 히트율을 비교하여 각 캐시의 성능을 평가한다.

첫번째 실험은 서버에 대하여 캐시의 크기(CS)를 변화시키면서 수행되었다. Threshold는 적용하지 않았다. 에듀넷 서버인 HP01, HP05, EDU01은 캐시의 크기를 512K에서부터 19M까지 11개 조건에 대하여 실험을 하였고 기타, NASA와 CLARKNET서버의 경우는 256K에서부터 2M가 될 때까지 실험을 수행하였다.

[그림 4]와 [그림 5]는 각 서버에서 캐싱 전략으로

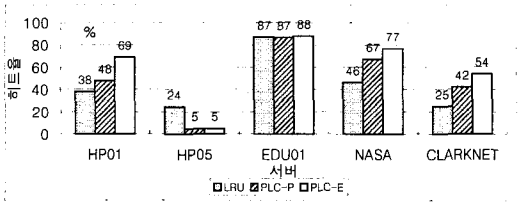


그림 5 각 서버의 평균 캐시 히트율

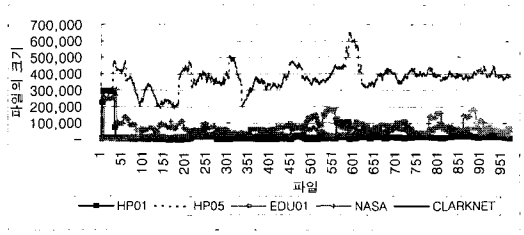


그림 6 각 서버의 상위 1000개 파일의 크기 분포

LRU를 사용하였을 때와 PLC-P, PLC-C를 사용하였을 때의 결과를 보여준다. 실험 결과를 보면 HP05를 제외한 다른 서버의 경우 캐시 히트율이 LRU << PLC-P << PLC-E의 순으로 나타나고 있다. HP05서버의 실험 결과는 LRU에 비하여 PLC가 매우 낮은 성능을 보이고 있다. 다른 서버들의 경우는 모두 LRU에 비하여 PLC가 높은 성능을 보이고 있다는 점으로 보아 HP05서버는 다른 서버와 다른 특이한 현상을 나타낸다고 볼 수 있다.

두번째 실험은 특이한 서버인 분류한 HP05가 다른 서버들과는 달리 PLC의 성능이 LRU에 비하여 현저히 떨어지는 이유를 보여준다. LRU의 경우 캐시의 성능은 각 파일의 locality에 의하여 결정되어 진다. LRU는 매번 자원이 요청될 때마다 캐시를 갱신하여 최근에 사용된 파일이 캐시에 존재하도록 한다. 이에 반하여 PLC의 경우는 하루동안의 선호도에 의하여 하루동안 캐싱될 파일을 결정한다. PLC의 경우 선호도가 높은 파일의 크기가 매우 큰 경우 캐시의 크기가 한정되어 있어 캐싱될 수 있는 파일의 수가 매우 적어지게 된다.

[그림 6]에서는 각 서버에서 서비스하는 파일들의 크기를 보여주고 있다. 각 서버에서 서비스된 CGI가 아닌 파일들 중에서 선호도가 높은 상위 1000개의 파일을 선택하여 파일의 크기를 추출하였다. 추출된 데이터를 선호도가 높은 파일부터 왼쪽으로부터 나열한 후 구간을 30으로 하는 이동평균을 구하여 그린 그래프가 [그림 6]이다. HP05는 대단히 큰 파일을 서비스하고 있다. 따라서 캐시의 크기가 작을 경우에 캐시에 캐싱되는 파일의 수가 매우 적게되며, 자연히 히트하는 파일의 수도 줄어들게 된다. 그러나 HP01의 경우에는 파일의 크기가 상대적으로 작다. 따라서 캐시에 캐싱되는 파일의 수가 많아지게 되었고, 이는 파일의 히트율을 높이는 효과를 주게 되었다.

6. 결 론

본 연구에서는 지금까지 웹 서버에서의 캐싱 연구에서와는 달리 웹 서버가 서비스하는 파일들은 매우 독특한 특성을 가지고 있음을 주시하고, 이를 근거로 요청된 각각의 파일을 사용자에게 전송하면서 이를 캐싱할 것인가를 결정하는 기존의 LRU와는 달리 일정기간 동안 단순히 서비스되는 파일들에 대한 통계만을 축적한 후, 이를 근거로 일괄적으로 캐싱하는 것을 가능하게 하는 새로운 캐싱전략인 PLC를 제안하였다. 유명 웹 사이트의 log 파일을 대상으로 실험한 결과, 히트율 측면에서 PLC 전략은 매 파일을 전송할 때에 이를 캐싱하지 않으면서도 기존의 LRU전략보다 높은 효과를 주며, 캐싱으로 인한 웹 서버의 부하를 줄이는 효과를 주고 있음을 확인하였다. 본 연구에서는 히트율 만을 캐싱 알고리즘의 기준으로 보았으나, 이외의 다양한 기준, 예를 들면, 전송률 등 다양한 기준에 대한 연구도 진행 중에 있다.

참 고 문 헌

[1] Internet Statistics:Web Growth, Internet Growth , http://www/mit.edu/people/mkgray/net/

[2] Bestavros. A, Matta., "Load profiling for efficient route selection in multi-class.," Network protocols, 1997.

[3] Bestavros. A, Cheatham. T, Jr. Stefanescu. D, "Parallel bin packing using first fit and k-delayed best-fit.," parallel and Distributed Processing, 1990.

[4] Mike Reddy & Graham P. Fletcher, "Intelligent web caching using document life histories: A comparison with existing cache management techniques," J228, School of Computing University of Glamorgan, Pontypridd, Mid Glamorgan. CF37 1DL.

[5] M. R. Korupolu and M. Dahlin, "Coordinated placement and replacement for large-scale distributed caches," Proceedings of the IEEE Workshop on Internet Applications, July 1999

[6] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, "Removal policies in network caches for World Wide Web documents," Proceedings of Sigcomm, pp. 293-304, 1996

[7] Igor Tatarinov, Alex Rousskov, valery Soloviev, "Static Caching in Web Servers," ncstrl.ndsu_cs/NDSU-CSOR-TR-97-04. 1997.
[8] 안효범, 조경산, "웹 서버의 참조 특성 분석과 성능 개선", 정보처리학회논문지A, V.8-A, N.3, pp.201-208, 2001.
[9] 염미령, "이중 큐 구조를 갖는 웹 서버", 정보처리학회논문지A, V.8-a, N.3, pp293-298, pp. 293-298, 2001.
[10] E.P.Markatos, "Main Memory Caching of web documents," Proceedings of the 5th International World Wide Web Conference, May 6-10, Paris, 1996.
[11] D.W.Chang, H.R.KE, R.C. Chang, "Adaptive-level memory caches on World Wide Web servers," Elsevier Computer Networks, 32, 2000, pp. 261-275.
[12] Stanistaw Walukiewiez, "Integer Programming," Polish Scientific Publishers \ Warszawa, 1991

유 해 영
1979년 단국대학교 수학과 졸업(이학사). 1982년 단국대학교 대학원 수학과 수료(이학석사). 1994년 아주대학교 대학원 컴퓨터공학과 수료(공학박사). 1983년 ~ 현재 단국대학교 정보컴퓨터학부 교수. 관심분야는 소프트웨어 공학, 시스템 프로그램 등

박 철
1994년 단국대학교 전산통계학과 졸업(이학사). 1998년 단국대학교 대학원 전산통계학과 졸업(이학석사). 2001년 단국대학교 대학원 전산통계학과 박사과정 수료. 관심분야는 웹 어플리케이션, 소프트웨어 공학, 분산 시스템 등