웹 서버 캐쉬를 향상시키기 위한 요청 패턴을 기반으로 한 선-인출 버퍼에 대한 연구

정회원 나 호 진*, 김 종 은**, 주 영 상***, 안 효 범**

Pre-fetch Buffer Based On Request Pattern to Improve Web Server Cache

Ho-Jin Na*, Joung-Eun Kim**, Young-Sang Joo***, Hyo-Beom Ahn** *Regular Members*

요 약

월드 와이드 웹을 이용한 정보의 제공은 웹 서버에 과중한 부하를 부여하고 있고, 이러한 이유로 웹 서버에서 빠른 응답시간을 얻기 위한 방법이 요구되어진다. 본 논문에서는 웹 서버의 문서를 요청하는 사용자들의 요청 패턴을 기반으로 한 선-인출 버퍼에 대하여 제안한다. 제안된 방법은 캐쉬와 병행하여 사용하였을 경우에는 선-인출 버퍼를 사용하지 않았을 경우보다 13%의 향상을 보임을 시뮬레이션을 통해 보였다.

ABSTRACT

An increasing amount of information is currently becoming available through World Wide Web Servers. Document requests to popular Web Servers arrive over few billion times per day at peak rate. To reduce the overhead imposed by frequent document requests, we propose a pre-fetch buffer based on the pattern of user's request to World Wide Web Server's documents

If we use cache, simulation result shows inappropriate hit rate of Web Server. we analyze that the hit rate of the proposed architecture increase 13%.

Ⅰ. 서 론

웹(WWW, World Wide Web) 사용자의 급격한 증가로 인하여 인기 있는 웹 서버는 병목현상과 트래픽(traffic)의 오버헤드(overhead)로 인해 사용자들에게 불편을 느끼게 하므로, 웹의 성능을 향상시킬 수 있는 여러 요소들을 분석하여 개선할 필요가 있다. 웹 성능 개선을 위한 프로토콜(protocol) 관련요소로는 정보의 요구와 응답 과정이 얼마나 효율적으로 운영되는지의 분석과 네트워크 트래픽을 줄이기 위한 ICP(Internet Cache Protocol)등을 들 수 있다. 또한 웹의 성능은 네트워크의 대역폭에 의한 정체보다는 서버와 통신망에서의 지연과 과부하가 문제가 되므로 이들을 줄일 수 있는 서버 캐쉬, 프락시 캐쉬, 클라이언트 캐쉬 기법들이 제안되었다[2][5].

이러한 다양한 기법들 중에서 본 논문에서는 웹 서버의 처리 시간을 줄이기 위하여 서버 캐쉬의 히트율을 높일 수 있는 기법을 제시한다. 많은 사용자가 동시에 서비스를 요청할 경우 서버 캐쉬에서 히트하지 못하면 디스크로부터 문서를 읽어 서비스해야 하므로 응답시간이 늦어지며, 서비스 자체가 불가능할 수도 있다. 그러나 기존 캐쉬의 관리 정책만으로는 높은 히트율을 제공하지 못하기 때문에, 본 논문에서는 홈 디렉토리 노드의 파일과 특정 노드의 파일들이 자주 참조되는 웹 성향을 고려하여 웹 페이지 링크 구조를 레벨로 나누어 가중치를 부여하고 문서를 참조할 때마다 증가치를 부여하였다.

* (주)코리온시스템 선임 연구원 ** 단국대학교 전산통계학과 *** 동해대학교 컴퓨터 공학과
논문번호 : T01003-0324, 접수일자 : 2001년 3월 24일

그리고 가중치와 증가치의 합이 한계치를 넘으면 선-인출 버퍼로 페치하는 선-인출(pre-fetch) 기법을 도입하여 히트율을 높이는 방안을 제시하고, 개선된 히트율을 실제 웹서버의 작업 부하를 이용한 시뮬레이션을 통해 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 웹의 성능적 문제점과 이에 대한 기존 해결책을 제시하고 3장에서는 클라이언트 캐쉬, 프락시 캐쉬, 서버 캐쉬를 각각 설명하고, 기존의 제거(replacement) 정책 관리 기법들을 소개한다. 4장에서는 웹 서버에서 문서의 참조의 투명성과 캐쉬의 히트율을 높이기 위한 선-인출 기법의 구조를 제안한다. 5장은 웹 서버에 요청된 내용을 반영한 Access_log 파일을 분석하고 시뮬레이션을 통해 제안된 구조의 성능을 분석하고, 6장의 결론으로 마무리한다.

Ⅱ. 웹 성능의 문제점과 해결책

웹의 성능이란 사용자가 웹을 이용하여 필요한 문서 정보들을 얼마나 신속하게 얻을 수 있는가의 평가이다[6]. 웹 성능 분석 지표는 웹이 얼마나 효율적이고 신속한가를 결정하는 통신망에 대한 관점과 서버의 처리 시간에 절대적 영향을 주는 서버 캐쉬의 히트율에 대한 관점으로 나눈다.

자료 패킷을 송수신하는 통신망에서의 성능은 지연(latency)과 대역폭(bandwidth)으로 나타낼 수 있고, 웹 캐쉬의 성능은 캐쉬의 히트율로 나타낸다. 요청에 대한 빠른 응답 시간을 제공하고 서비스 불가능 상태를 줄이기 위한 서버 캐쉬는 응답 시간에

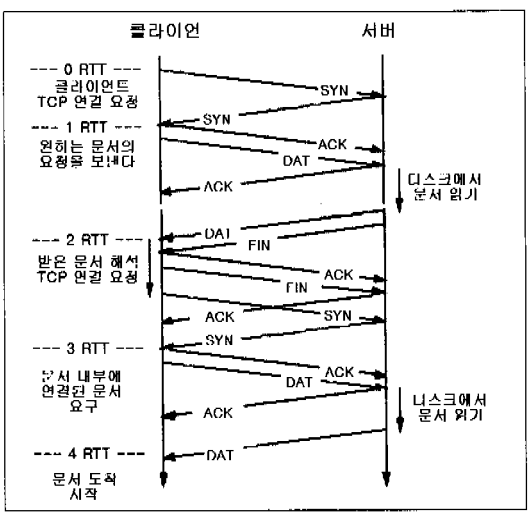


그림 1. HTTP 데이터 송수신을 위한 TCP 세그먼트 교환

직접적으로 영향을 미치기 때문에 문서의 히트율(Document Hit Rate)로 평가되고, 네트워크의 전송을 줄이기 위한 프락시 캐쉬는 네트워크의 대역폭을 반영하기 때문에 바이트 히트율(Byte Hit Rate)로 평가된다[5].

1. HTTP 프로토콜의 문제와 개선 방법들

웹에서 서버와 클라이언트 사이의 하이퍼텍스트 문서의 전송에 사용되는 프로토콜인 HTTP는 TCP/IP 프로토콜 체계상에서 TCP 연결을 통해서 이루어진다. 클라이언트에서 브라우저를 이용하여 서버로부터 문서를 갖고 오기 위해서는 그림1과 같이 HTTP 프로토콜을 이용하여 문서를 요청하고, 서버는 이에 대한 응답으로 요청된 문서를 전송한다. 대부분의 웹 페이지는 여러 그림 파일을 전송 받아오기 위해 매번 앞에서 설명된 TCP연결을 설립해야 하므로 사용자의 지연이 증가하고 추가적인 오버헤드가 발생하므로 비효율적이다. 이런 문제점을 개선하기 위하여 다음과 같은 몇 가지 방법이 제시되었다[10].

첫 번째 방법인 Long-lived Connections은 브라우저가 서버에게 HTML 문서를 처음 요청할 때 설립된 TCP연결을 서버가 문서를 전송하고 난 후에도 해제시키지 않고 연결된 상태로 유지한다. 따라서, 동일한 서버로부터 각각의 데이터 송수신을 위한 별도의 TCP 연결 설립을 하지 않아도 연속적인 데이터 요청을 할 수 있으므로 프로토콜의 성능을 향상시킬 수 있다[11].

두 번째 방법인 파이프라이닝은 GETALL 메소드(Method), GETLIST 메소드, 연속적 GET 메소드 방법을 사용하여 HTTP 프로토콜을 개선하여 웹 성능의 향상을 보인다[11]. GETALL은 클라이언트가 서버에게 요구 메시지를 보낼 때, 요청된 문서에 포함된 같은 서버 내에 있는 파일들을 한꺼번에 보내달라고 요청하는 방법이다. GETLIST는 클라이언트에서 사용하는 캐시 속에는 이미 많은 문서들이 캐싱되어 있을 수 있으므로 먼저 GET 메소드를 이용하여 해당 HTML 파일을 가져오고, 파일 내의 정보를 분석하여 캐쉬 되어 있지 않은 파일들에 대해서만 하나의 응답 메시지로 전송해 달라고 요청하는 방법이다. 연속적 GET은 하나의 long-lived 연결을 맺어 아직 응답 메시지가 도착하지 않았음에도 연속적인 GET 요구 메시지를 서버에게 전달한다. 이때 서버는 요구 메시지를 수신한 순서대로 응답 메시지를 전달해야 한다.

세 번째로 데이터 압축은 전송할 문서 파일을 압축하여 데이터의 양을 줄이는 방법이다. 이것은 클라이언트에서 "Accept-Encoding" 헤더 필드에 압축 방식을 설정하여 요구 메시지에 포함하여 전송하면 서버에서는 데이터 부분을 압축하고 압축 방식을 포함한 응답 메시지를 사용자 데이터 부분에 실어서 보내게 된다. 클라이언트에서는 이 정보를 이용하여 원래의 데이터로 복원한다[6].

2. 문서의 일관성(Coherence)

모든 웹 캐쉬는 최근의 문서를 가지고 있어야 한다. 이처럼 캐쉬가 가장 최근의 문서를 유지하는 것을 일관성 유지라고 한다. 그러나 일관성을 유지하기 위해 서버의 부하 또는 네트워크의 부하를 증가시키면, 웹의 성능 저하를 일으킨다. 따라서 사용자의 대기 시간을 줄이고, 서버나 네트워크의 부하를 최소하면서 일관성 문제를 해결해야 한다[1].

분산 파일 시스템에서는 일관성을 유지하는 정책으로 유효성 검사 메커니즘(Validation Check Mechanism)과 콜백 메커니즘(Callback Mechanism)을 사용한다.

유효성 검사는 캐쉬 되어진 문서의 최종 변경 일자를 타임스템프(time stamp)에 기록하여 이 정보를 이용하여 캐쉬 되어진 이후 변경 여부를 확인한다. 콜백은 클라이언트가 서버로부터 문서를 받은 후, 문서가 변경되었다면 서버는 모든 클라이언트에게 문서 변경을 알린다. 분산 파일 시스템의 일관성을 유지하는 위의 정책을 웹에서 응용하여 사용한다. 그러나 문서를 변경할 때 그 내용을 모든 클라이언트들에게 알려주기 위해서는 고비용이 요구되므로, 콜백 보다는 유효성 검사를 이용한다. 다음의 정책들은 웹 상에서 문서의 일관성 문제를 해결하기 위한 정책들이다.

HTTP GET 메소드는 요구 메시지에 If-Modified-Since 헤더 필드를 포함시켜 GET을 조건부 GET으로 동작할 수 있도록 한다. 이럴 경우 GET의 의미는 지정된 자원이 지정된 일자 이후에 수정된 것일 경우에만 전송하라는 것이다. 이 조건을 이용하여 불필요한 데이터 전송을 막을 수 있으므로 네트워크의 활용성을 높일 수 있다.

만기 시간(expiration time)에 근거한 일관성 유지 방법은 캐쉬 되어진 모든 문서에 할당된 만기 시간까지만 일관성이 있다고 판단하고, 만기되었을 때는 조건부 GET요구를 상위 캐쉬에 보내어 문서의 변경여부를 확인한다. 만약 변경되었다면 최근의 문서를 가져오고 만기 시간을 재 설정한다. 하지만 이 방법은 문서의 만기 시간을 결정하는 것이 어렵다는 문제가 있다.

선-인출(Pre-fetching)방법은 주기적으로 캐쉬 된 문서들을 최근의 문서로 갱신하는 방법이다. 클라이언트들은 주기적으로 최근의 문서를 가져오므로 잠재된 일관성 문제를 예방한다. 이 방법에서는 문서가 갱신되는 주기를 결정하기 어렵다는 문제가 있다.

Ⅲ. 캐쉬의 구조

앞에서 제시된 웹 성능향상에 대한 문제점과 해결책 외에도, 캐싱 기법을 이용하여 웹의 성능 향상을 가져올 수 있다. 웹 캐쉬는 네트워크를 통하여 다른 컴퓨터로부터 문서들을 가져와 디스크나 메모리에 저장한다. 웹 캐쉬에는 사용자에게 빠른 응답 시간을 주기 위한 클라이언트 캐쉬와 네트워크의 부하를 줄이기 위한 프락시 캐쉬 및 웹 서버의 과부하로 인한 대기 시간 증가를 방지하기 위한 서버 캐쉬가 있다.

1. 클라이언트 캐쉬

클라이언트 캐쉬에는 그림 2에서처럼 브라우저가 탐색하는 항목의 복사본이 디스크에 캐싱된다. 사용자가 항목을 선택하면, 브라우저는 자신의 디스크 캐쉬를 검사하여 캐쉬에 있으면 캐쉬로부터 복사본을 가져온다. 네트워크의 지연이 없기 때문에 빠른 응답 시간을 가져올 수 있다. 그러나 디스크 용량의 한계로 웹의 방대한 양의 문서를 모두 디스크에 저장할 수도 없고, 변화되는 정보를 검색하기 때문에 캐싱 되어진 문서가 다시 참조될 확률은 높지 않다. 따라서, 브라우저는 캐싱된 문서를 사용자의 정의에 의해 일정 시간이 지나면 삭제할 수 있다.

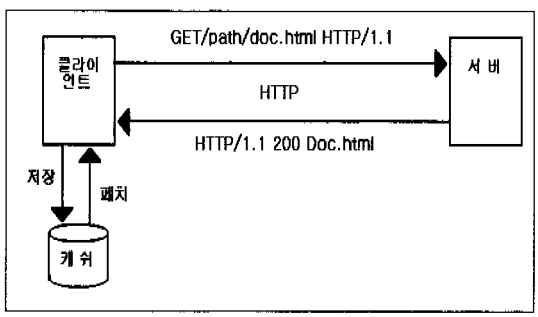


그림 2. 클라이언트 캐쉬[3]

2. 프락시 캐쉬

프락시 캐쉬는 클라이언트가 요구할 때마다 매번 서버에서 문서를 가져오게 되어 발생하는 문제를 해결하기 위해 사용된다. 요구한 문서를 서버에 접속하지 않고, 네트워크 상의 클라이언트와 가까이 있는 프락시 서버가 프락시 캐쉬로부터 요구에 대한 응답을 한다[2]. 만약 가까이 있는 프락시에 문서가 존재하지 않으면 ICP(Internet Cache Protocol)을 이용하여 형제 프락시 또는 부모 프락시에게 문서를 요청하고 이 곳에도 존재하지 않으면 그림 4에서처럼 문서를 서버에서 가져와 클라이언트에게 응답하고, 다음 사용자를 위해 문서를 프락시 서버에 저장한다[4][7].

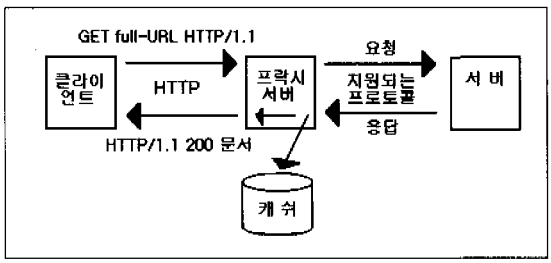


그림 3. 프락시 캐쉬

3. 서버 캐쉬

클라이언트의 요구가 발생하면 서버는 요구한 문서를 열고(open), 문서를 읽어(read) 임시 버퍼에 저장하고, 문서를 닫는다(close). 이와 같은 일을 반복하므로 사용자 응답 시간이 증가하게 된다. 동시에 많은 요청이 서버에 들어와 위와 같은 일을 반복한다면 응답 시간이 늦어질 뿐만 아니라 서버의 부하로 인해 서비스 자체가 불가능하게 된다. 이 때문에 클라이언트 캐쉬나 프락시 캐쉬 보다 서버 캐쉬의 히트율이 중요 시 되고 있다.

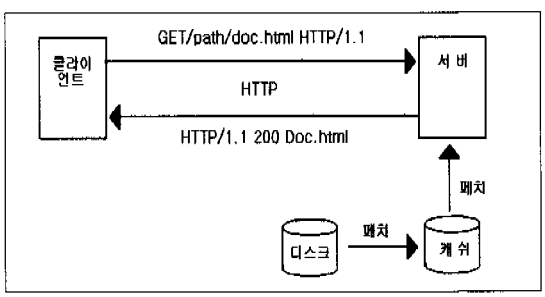


그림 4. 서버 캐쉬

4. 기존 캐쉬의 관리 정책

웹 캐쉬는 자주 접근하는 문서를 저장하여 긴 지연이 요구되는 디스크의 접근을 피하는 것이 목적이다. 웹 서버에서 사용되는 캐쉬의 관리 정책은 웹 특성상 문서의 크기가 다르다는 것을 고려하여 문서 크기, 마지막 접근 시간, 접근 횟수 등을 캐쉬의 제거 정책의 요소로 사용한다.

다음은 웹 서버의 기존 캐쉬 제거 정책들이다[8][9].

1) LRU-SIZE는 문서 크기 순으로 항목을 정렬한 후, 가장 큰 문서를 삭제하여 크기가 같을 경우 LRU 정책을 사용한다.

2) LRU-‖ Log ‖ 는 LRU-SIZE의 문제점을 보안하기 위해 16K와 32K-1 사이의 크기를 가진 문서는 같은 크기로 인식한다. 따라서 크기가 같은 문서가 많이 발생되기 때문에 다른 캐쉬 정책의 요소들과 함께 사용한다.

3) LRU-MIN은 작은 문서를 캐쉬에 저장하기 위하여 큰 문서를 삭제하는 문제를 막기 위해, 새로 들어오는 문서와 캐쉬 되어진 문서들의 크기를 비교하여 새로 들어온 문서보다 같거나 큰 문서를 LRU을 사용하여 삭제한다.

4) LRU-TH는 캐쉬 가능한 문서 크기에 한계 값을 두어 캐쉬 한다. 이 정책은 한계 값에 의존하며, 최적의 한계 값은 캐쉬 크기와 웹 트래픽(traffic)등에 다양하게 의존한다. 그러나 최악의 경우 LRU - MIN와 마찬가지로 CPU에 과부하를 발생시키고, 최적의 한계 값이 상황에 따라 변화한다는 단점을 가지고 있다.

5) LRU-k-TH는 데이터 아이템에 대해 k번째 접근된 횟수를 비교하여, 횟수가 가장 적은 문서를 캐쉬에서 삭제한다. LRU - TH에 비해 더 정교하다. 그러나 캐쉬 관리를 위한 해쉬 테이블 공간이 필요하고, 문서의 k번 접근 정보와 접근 시간을 저장할 수 있는 추가적인 공간이 필요하므로 부적절한 메모리 요구와 높은 CPU 과부하를 발생한다는 단점을 가지고 있다.

6) Perfect-LFU는 문서가 캐쉬에서 삭제되어도, 참조한 counter는 남아있다. 바이트 히트율에서 높은 수치를 보인다. 그러나 cache-LFU에 비해 많은 오버헤드(공간에 대해)를 발생한다는 단점이 있다.

7) GD-Size는 문서의 사이즈와 지역성을 고려한 정책이다. 문서 히트율에서는 높은 수치를 보인다. 그러나 큰문서는 캐쉬 되지 않는다는 단점이 있다.

Ⅳ. 웹서버에서 문서의 참조 특성과 제안구조

본 장에서는 웹 서버에서의 문서의 참조 특성을 분석하여, 그 특성을 활용하여 네트워크 서버에서의 작업 부하를 줄이고, 빠른 응답 시간을 보이기 위해서 사용자 요구 패턴에 기반을 둔 선-인출 (Pre-fetch) 기법을 제안한다.

1. 참조와 파일 크기에 대한 분석

본 절은 제안 구조의 타당성을 검증하기 위해서 문서 참조에 대한 정보와 파일 크기 정보를 가지고 있는 Access_log 파일을 분석한다[5].

그림5는 참조 수 범위에 대한 파일 수를 나타낸다. 전체 참조된 파일 수는 4,007개이고, 전체 참조의 수는 441,873개이다. 전체 파일 중에 10번 이하 참조한 파일들이 전체 파일수의 60%를 차지하지만 전체 요청 중에서는 1%도 안 되는 10,000번의 요청에 불과하다. 반면에 5,000번 이상 참조한 파일은 29개에 불과하지만 전체 요청에 38%를 차지하는 168,746번의 참조가 있었다. 이는 소수의 파일들만이 반복되어 참조되는 것을 의미한다.

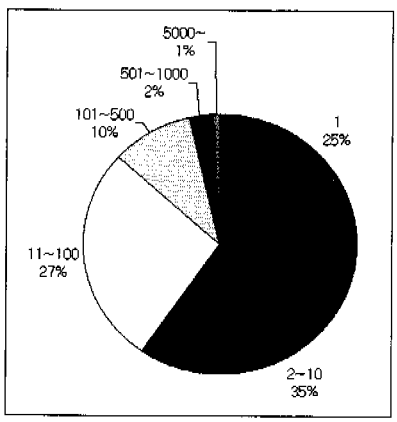


그림 5. 파일 참조 분포

그림6는 파일 크기 범위에 대한 파일 수를 나타낸다. 79% 파일들이 16K을 넘지 않는 크기가 작은 파일들이고, 표1에 의하면 그림5 의 참조 수가 많은 파일들 대부분은 크기가 작은 파일들이다.

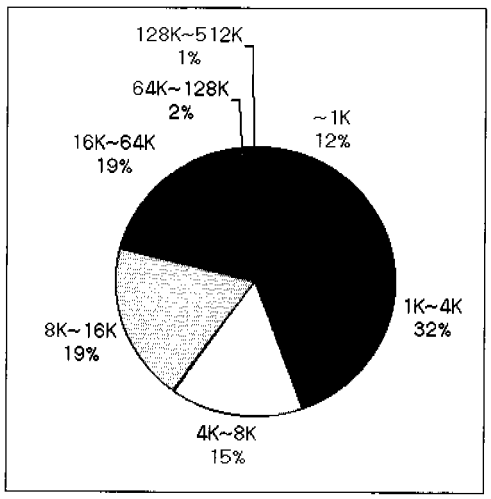


그림 6. 파일 크기 분포

표1은 그림4와 그림5의 결과와 누적 값을 나타낸다. 크기는 10K 보다 작은 파일들로 제한하였다. 표1에서 볼 수 있듯이 10K 보다 작은 파일들 중에 500번 이상의 참조가 있는 파일들의 크기 합이 310K이므로, 310K 정도의 작은 선-인출 버퍼에 저장해 두면 최대 70%정도의 히트율이 예상되고 100번 이상의 참조가 있는 파일들의 크기 합이 1.3M이므로, 1.3M 정도의 선-인출 버퍼에 저장해 두면 최대 85% 정도의 히트율을 예상할 수 있다.

2. 제안 구조

본 절에서는 캐쉬의 히트율을 높여 응답 시간을 줄이고, 접속 불가능 상태가 발생하지 않도록 웹 서버 캐쉬에서 사용자 요청 패턴에 기반을 둔 선-인출 기법을 제안한다.

표 1. 10Kbyte 이하인 파일에 대한 누적 분산

참조_수 범위	파일 수	파일 수 누적	문서 크기(Byte)	문서 크기 누적(Byte)	참조 수 합	참조 수 누적 합
5000-	26	26	78,124	78,124	186,296	186,296
2001-5000	7	33	8,311	86,435	30,960	217,256
1001-2000	12	45	14,897	101,332	37,057	254,313
501-1000	84	129	208,982	310,314	60,792	315,105
101-500	322	451	992,507	1,302,021	61,172	376,277
11-100	797	1248	2,782,536	4,085357	27,950	404,227
1-10	1345	2593	4,492,921	8,578278	4,607	408,834

본 논문에서 제안하는 선-인출 기법은 사용자가 요청한 문서에 대한 과거 참조 패턴에 기반을 둔 기법으로 그림6과 같은 웹의 링크 구조를 기반으로 한다. 각 참조되는 웹 페이지를 노드(node)로 그들의 참조 관계를 링크로 표시한다. 홈 디렉토리의 노드를 루트 노드(root node)로 하고 루트 노드로부터의 거리를 레벨(Level)로 표시하여 레벨이 낮아질수록 작은 가중치를 부여한다. 이와 같이 하는 이유는 사용자 요청 패턴은 루트 노드에 접속하여 링크를 따라 문서를 참조하고, 필요한 정보를 찾는다면 사용자는 그 이하 레벨에 있는 문서는 참조하지 않고 접속을 끊으므로 루트 노드에 가까울수록 파일이 참조될 확률이 높기 때문이다.

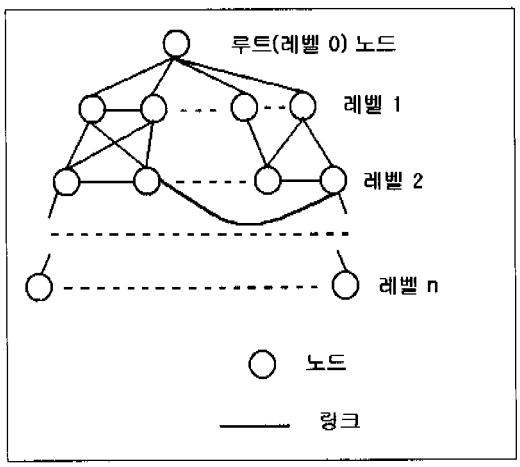


그림 7. 웹서버에서의 디렉토리 구조

선-인출 테이블(pre-fetch Table) 구조

파일명	크기	가중치	증가치	한계치

그림 8. 선-인출 테이블 구조

사용자가 웹에 접근할 때 항상 루트 노드로 먼저 접근하는 것은 아니다. 예를 들면, URL등을 사용하여 임의의 노드로 직접 접근할 수도 있다. 따라서, 참조가 있을 때마다 해당 문서에 부여된 가중치에 증가치를 더한다. 즉 레벨의 가중치가 낮더라도, 접근 횟수가 많다면 선-인출될 수 있도록 한다. 그림8은 선-인출을 위한 정보를 가지고 있는 선-인출 테이블의 구조이다. 크기는 파일의 크기이고, 가중치와 한계치는 초기에 부여되며 한계치가 가중치보다 크다. 증가치는 파일이 참조될 때 증가하는 값이다.

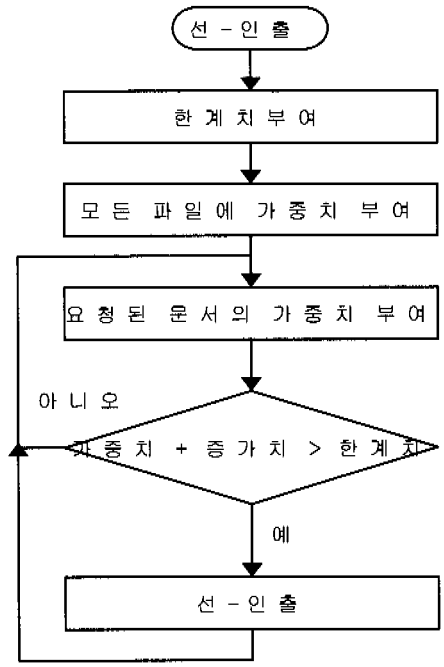


그림 9. 선-인출 알고리즘

그림9은 논문에서 사용된 선-인출 알고리즘을 보인다. 문서 각각에 대해 가중치는 사전에 부여하고, 가중치가 일정한 값이 되었을 때 선-인출하도록 하는 한계치도 사전에 부여한다. 증가된 값이 한계치보다 커지게 되면 해당 파일은 선-인출 버퍼로 인출된다. 선-인출은 사용자의 요청이 있기 전에 미리 수행한다.

클라이언트로부터 요청이 들어오면 서버는 한계치 보다 큰 가중치를 가진 문서들만 존재하는 선-인출 버퍼를 확인하여 만약 존재한다면 선-인출 버퍼에서 응답한다. 문서가 선-인출될 때 선-인출 버퍼가 더 이상 문서를 삽입할 수 없다면 선-인출 버퍼에서 LRU 정책을 사용하여 문서를 삭제하고 한계치 보다 높은 파일을 선-인출 버퍼에 삽입한다. 문서가 선-인출 버퍼에 존재하지 않으면 캐쉬에 존재하는지 확인하고 문서가 존재하면 응답하고, 존재하지 않는다면 디스크의 문서를 캐쉬로 가져와 적재하고 클라이언트에게 응답한다. 캐쉬 역시 문서를 더 이상 삽입할 수 없다면 LRU 정책을 사용하여 문서를 제거한 후 미스한 문서를 삽입한다. 이 때 선인출 버퍼는 캐쉬와의 문서의 중복을 피하기 위해 문서는 삽입하지 않고, 한계치가 넘는 문서에 대해서만 선-인출 버퍼에 삽입한다.

V. 성능 분석

본 장에서 제안된 기법의 검증을 위해 사용될 Access_log 파일의 특성을 분석한다. Access_log 파일은 사용자의 요청이 발생할 경우 요청에 관련된 모든 내용을 기록한 파일이므로 실제 요청을 반영한다.

웹 서버의 성능을 분석하기 위해서는 가상적인 작업 부하보다는 실제 웹 서버에 요청된 내용의 처리를 분석하는 것이 효율적이다. 본 논문의 제안 구조에 대한 성능 분석 결과를 추출하기 위해 Sun Microsystems Inc. SunOS 5.6 Generic Augest 1997의 단국대학교 서버의 1999년 6월 1일 - 6월 21일간 참조된 Access_log 파일에 저장된 정보를 사용한다. Access_log 파일은 133,496K byte 파일 크기에 713,659번의 요청 중에 서버로부터 문서를 가져간 상태코드 200번인 서버에서 파일을 제공한 내용만을 추출한 80,126K byte의 파일 크기의 441,873번의 요청만을 사용하였다. 요청에 사용된 파일은 총 4,007개이다.

1. Access_log 파일 분석

웹 서버는 자신에게 참조 요청된 기록을 Access_log 파일에 별도로 저장하고 있다. Access_log파일은 웹 서버의 요청에 대한 모든 내용을 담고 있다. 표2은 이 파일의 형식을 보인다.

표 2. Access_log 파일내의 정보

URL	시간	GET	버전	상태 코드	크기	파일 경로

URL은 요청된 웹사이트에 대한 유일한 이름으로 절대 URL(Absolute URL) 또는 절대 경로(Absolute PATH)로 지정할 수 있다. 시간은 서버에서 요청된 파일을 전송하기 시작한 시간을 나타낸다. GET은 서버에서 전송한 파일의 절대경로이다. VERSION은 HTTP 프로토콜의 버전을 나타낸다. HTTP는 0.9에서 시작해서 지금은 성능이나 효율이 개선된 1.1을 사용한다. 상태 코드는 응답 상태를 나타내는 3자리 정수로 코드의 첫 자리 수에 따라 1XX는 정보, 2XX는 성공과 같이 다른 의미를 갖는다(예, 200은 OK). SIZE는 전송한 문서의 크기를 바이트 단위로 나타내고, 파일 경로는 파일을 절대 URL로 나타낸다.

본 논문에서는 Access_log 파일의 상태코드, 크기(SIZE), 파일경로의 정보를 추출하여 사용한다.

2. 히트율 분석

표 3. 선-인출 버퍼 크기 변화에 따른 히트율

캐쉬와 선인출 버퍼의 크기 합(K Byte)	선인출 버퍼 크기 비율에 대한 히트율			
	5%	10%	20%	50%
36044.8	99	99	98.8	98.4
18022.4	98.3	98.3	98	96.3
9011.2	95.6	95.8	94.3	91.5
4505.6	90	90.6	89.7	84.3
2253.8	83	83.1	81.8	75.1
1126.4	73.4	73.8	72.2	65.4
563.2	63.8	64.1	61	52.3
281.6	51.2	52.2	49.5	36.5
140.8	33.3	34.2	31.7	22.8

표 4. 캐쉬 + 선인출 버퍼에 의한 히트율

캐쉬 크기(K Byte)	캐쉬 히트 수	선인출 버퍼 크기(K Byte)	선인출 버퍼 히트 수	캐쉬+선인출 버퍼 히트 수	히트율(%)
32768	9424	3276.8	428079	437503	99
16384	8809	1638.4	426739	435548	98.3
8192	11467	819.2	414688	426155	95.8
4096	18115	409.6	386206	404321	90.6
2048	24536	204.8	347363	371899	83.1
1024	30162	102.4	300906	331068	73.8
512	31577	51.2	256476	288053	64.1
256	45350	25.6	187362	232712	52.2
128	51109	12.8	106752	157861	34.2

선-인출 버퍼의 적절한 크기를 정하기 위해, 표3은 서버 캐쉬 크기에 대한 선-인출 버퍼 크기의 비율을 5%, 10%, 20%, 50%로 적용한 결과이다. 10%에서 가장 히트율이 높게 측정됨을 볼 수 있다. 그러므로 본 논문에서는 히트율이 가장 높은 10%를 선-인출 버퍼의 크기로 할당하여 히트율을 측정하였다. 표4는 캐쉬와 선-인출 버퍼의 히트율을 별도로 분석한 결과다.

선-인출 버퍼의 크기가 3M Byte 인 경우에 42만번 정도 히트하였다. 그리고 캐쉬에서는 9천번 정도의 히트를 보인다. 이는 대부분이 선-인출 버퍼에서 히트하고, 선-인출 버퍼에서 히트하지 못할 경우 캐쉬에서 히트함을 볼 수 있다. 이때의 전체 히트율은 99%에 이른다. 선-인출 버퍼와 캐쉬가 작아지면서 상대적으로 캐쉬는 히트수가 늘어나는 것을 볼 수 있다. 이는 선-인출 버퍼가 작아지면서 히트하지 못하여 캐쉬에서 히트함을 보인다.

표 5. 캐쉬만에 의한 히트율

캐쉬 크기(K Byte)	히트 수	히트율(%)
36044.8	437114	99
18022.4	431784	97.7
9011.2	415255	94
4505.6	384741	87
6252.8	343703	77.7
1126.4	296450	67
563.2	240124	54.3
281.6	169303	38.3
140.8	92374	21

표5는 제안 구조와의 비교를 위해 캐쉬만을 사용하는 경우의 히트율이다. 캐쉬 크기가 140K부터 36M까지 증가하면서 히트율 증가를 볼 수 있다. 히트율은 최소 21%에서부터 최대 99%까지 증가함을 볼 수 있다.

3. 히트율 비교

제안된 구조는 사용자들의 요구 패턴에 기반을 두어 요청이 자주 일어나는 문서를 예측하여 선-인출 버퍼에 가져다 둠으로써 높은 히트율을 보인다. 표4와 표5의 히트율을 비교하면, 선-인출 버퍼와 캐쉬를 사용할 경우 캐쉬만을 사용하는 것 보다 최대 13% 정도의 히트율 향상을 보인다. 그림 10은 두 시뮬레이션 결과 비교를 그림으로 보인다.

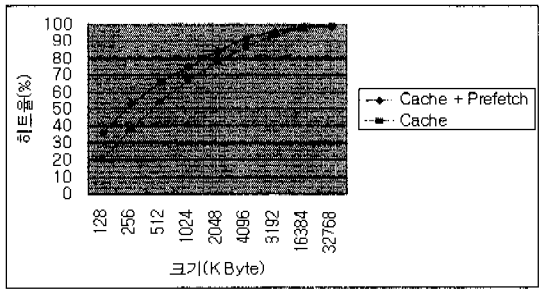


그림 10. 히트율 비교

Ⅵ. 결 론

본 논문에서는 기존 웹 서버 캐쉬의 낮은 히트율로 웹 사용자에게 늦은 응답 시간을 유발하는 문제점을 해결하기 위하여 사용자 요청 패턴에 기반을 둔 선-인출 버퍼를 사용하여 캐쉬의 히트율을 높이는 기법을 제안하였다.

사용자의 요청 패턴은 루트 노드에 가까울수록 많이 참조하고, 루트 노드가 아닌 특정 노드로도 많은 참조가 있음을 Access_log파일 분석으로 제시하였다.

자주 참조되는 파일들은 요청이 있기 전에 선-인출 버퍼에 삽입시키고, 삽입이 불가능할 경우 선-인출 버퍼에서 문서를 삭제하고 선-인출 테이블로부터 문서를 페치하여 선-인출 버퍼에 삽입한다. 선-인출 버퍼는 캐쉬와의 중복을 최소화하기 위해 사용자 요청에 미스한 문서는 삽입하지 않고 캐쉬에만 삽입한다. 선-인출 버퍼에서 미스한 경우 캐쉬에 존재 여부를 확인하여 캐쉬에서도 미스하면 디스크로부터 문서를 페치하여 캐쉬에 저장하고 요청에 응답한다. 본 연구에서는 선-인출 버퍼의 적절한 크기를 정하기 위하여 서버 캐쉬 크기에 대한 선-인출 버퍼 크기의 비율을 변화시켜본 결과 선-인출 버퍼의 크기가 전체 캐쉬 크기의 10%에서 히트율이 가장 높게 측정되었고, 캐쉬만을 사용했을 보다 사용자 요청 패턴에 기반을 둔 선-인출 버퍼를 같이 사용했을 때의 히트율이 최대 13% 정도 높음을 보였다. 연구 결과에 따라 사용자 요청 패턴에 기반을 둔 선-인출 버퍼를 사용하므로 웹 서버 캐쉬의 히트율 향상을 볼 수 있다.

참 고 문 헌

[1] A. Dingle, T. Partl, "Web cache coherence," Computer Networks and ISDN Systems, Vol.

28, No. 7, pp907-920, May 1996.

[2] Luotonen, A., and Altis, K. 1994. World-Wide Web Proxies. Computer Networks and ISDN systems, First International Conference on the World-Wide Web, Avaliable from http://www1.cern.ch/PapersWWW94/luotonen.pshttp://www1.cern.ch/WWW94/PrelimProcs.html

[3] Douglas E. Comer, Computer Networks and Internets, pp. 374-375, Prentice Hall, 1997

[4] D. Wessels and K. Claffy, "Internet cache protocol (ICP), version 2," Network Working Group RFC 2186, September 1997. Available from ftp://ftp.isi.edu/in-notes/rfc2186.txt

[5] Evangelos P. Markatos, "Main memory caching of Web documents," In Proceedings of the Fifth International World-Wide Web Conference, Paris, France, May 1996.

[6] Henrik Frystyk Nielsen, Jim Gettys, Anselm BairdSmith, Eric Prudhommeaux, Hakon Wium Lie, and Chris Lilley. Network performance effects of http/1.1. In Proceedings of ACM SIGCOMM'97, August 1997. Available from http://www.w3.org/Protocols/HTTP/Performance/Pipeline.

[7] J. Wang. A survey of web caching schemes for the internet. ACM Computer Communication Review, Vol 29, pp. 36-46, October 1999.

[8] Igor Tatarinov, "Performance Analysis of Cache Policies for Web Servers," In proc of 9th Intl Conf on Computing and Information, ICCI'98 Winnipeg, June 1998. Available from http://www.cs.ndsu.nodak.edu/~tatarino/pubs/ICCI98-final.ps

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the Implications of Zipf's Law for Web Caching", In 3rd International WWW Caching Workshop, June 1998.
Available from
http://wwwcache.ja.net/events/workshop/26/cao-zipf-implications.ps.

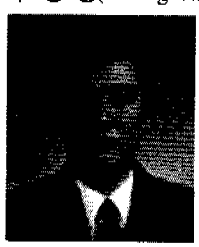
[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. "Hypertext Transfer Protocol -- HTTP/1.1," Internet Engineering Task Force Working Draft, Aug 1996.
Avaliable From
http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[11] Venkata N. Padmanabhan and Jeffrey C. Mogul. "Improving HTTP latency." Computer Networks and ISDN Systems, 28(1/2), pp.25-35, December 1995.

나 호 진(Ho-Jin Na)
1998년 2월 : 단국대학교 응용물리학과 (이학사)
2001년 2월 : 단국대학교 전산통계학과 (이학석사)
2001년 1월~현재 : (주)코리온시스템 선임 연구원
<주관심 분야> 전자상거래, 네트워크 보안

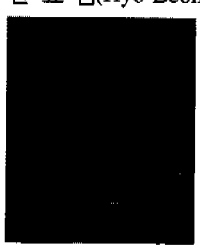
김 종 은(Joung-Eun Kim)
1997년 2월 : 단국대학교 전산통계학과(이학석사)
1997년 8월~현재 : 단국대학교 전산통계학과 박사과정
<주관심 분야> 분산 시뮬레이션, 컴퓨터구조

주 영 상(Young-Sang Joo)



1995년 8월 : 단국대학교 전산통계학과(이학석사)
2000년 2월 : 단국대학교 전산통계학과(이학박사)
2000년 3월~현재 : 컴퓨터공학과 동해대학교 전임강사
<주관심 분야> 파이프라인, 데이터선인출, 컴퓨터구조

안 효 범(Hyo-Beom Ahn)



1994년 2월 : 단국대학교 전산통계학과(이학석사)
1994년 3월~현재 : 단국대학교 전산통계학과 박사과정
1998년 10월~현재 : 천안공업대학 정보통신과 조교수
<주관심 분야> 네트워크 시스템, 웹 캐쉬, 스트리밍 캐쉬, 네트워크 보안