클러스터 웹 서버에서 성능 향상을 위한 노드간 선인출 기법

(Back-end Prefetching Scheme for Improving the Performance of Cluster-based Web Servers)

박 선 영 † 박 도 현 †† 이 준 원 ††† 조 정 완 †††

(Seon-Yeong Park) (Dohyun Park) (Joonwon Lee) (Jung-Wan Cho)

요 약 급속히 증가하고 있는 인터넷 트래픽의 절반 이상이 웹 서비스에 관련된 것으로 인터넷에서 웹이 차지하는 비중은 점점 커지고 있다. 증가하는 웹 서비스 요구에 대처하기 위해서 확장성과 가격 대 성능비가 우수한 클러스터 웹 서버가 최근 많이 연구되고 있다. 클러스터 웹 서버는 여러 대의 서버 노드로 구성되어 있는데 각 서버 노드에 들어오는 사용자 요구에 대한 응답 데이타가 지역 메모리에 없는 경우, 디스크 접근이나 다른 서버 노드로부터의 데이타 전송이 필요하다.

본 논문에서는 클러스터 기반 웹 서버에서 서비스 지연을 감소시키기 위한 서버 노드간 자료 선인출 기법을 제안하고 이를 위한 설계 방법을 소개한다. 또한, 선인출에 필요한 알고리즘을 제안하고 모의 실험을 통해 제안하는 알고리즘의 성능을 측정하였다. 후변(back-end) 웹 서버에서 수집된 로그를 바탕으로 서비스 지연 시간을 측정한 결과, 노드간 선인출 방법을 사용한 것이 사용하지 않은 경우에 비해 약 10~25% 감소하였다. 이 때, 각 서버 노드의 메모리 크기는 웹 서버 로그로부터 측정된 전체 요구 데이타 크기의 약 10% 정도이다. 제안하는 선인출 알고리즘 중 접근 확률(access probability)과 사용자 요구 사이의 지연 시간을 고려하는 선인출 알고리즘인 TAP2(Time and Access Probability-based Prefetch) 방법이 가장 좋은 성능을 보였다.

키워드 : 웹 서버, 클러스터 시스템, 선인출 알고리즘, 접근 확률

Abstract With the explosive growth of WWW traffic, there is an increasing demand for the high performance Web servers to provide a stable Web service to users. The cluster-based Web server is a solution to cope with the heavy access from users, easily scaling the server according to the loads. In the cluster-based Web server, a back-end node may not be able to serve some HTTP requests directly because it does not have the requested contents in its main memory. In this case, the back-end node has to retrieve the requested contents from its local disk or other back-end nodes in the cluster.

To reduce service latency, we introduce a new prefetch scheme. The back-end nodes predict the next HTTP requests and prefetch the contents of predicted requests before the next requests arrive. We develop three prefetch algorithms based on some useful information gathered from many clients' HTTP requests. Through trace-driven simulation, the service latency of the prefetch scheme is reduced by 10 ~ 25% as compared with no prefetch scheme. Among the proposed prefetch algorithms, Time and Access Probability-based Prefetch (TAP2) algorithm, which uses the access probability and the inter-reference time of Web objects, shows the best performance.

Key words : Web Server, Cluster System, Prefetch Algorithm, Access Probability

† 비 회 원 : 한국전자통신연구원 컴퓨터시스템연구부 연구원
sypark@etri.re.kr
†† 비 회 원 : 한국과학기술원 전산학과
dhpark@calab.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
joon@calab.kaist.ac.kr
jwcho@calab.kaist.ac.kr
논문접수 : 2001년 3월 19일
심사완료 : 2002년 2월 18일

1. 서 론

최근 많은 웹 사이트들은 급증하는 사용자 요구를 처리하기 위하여 여러 대의 PC나 워크스테이션으로 구성된 클러스터 웹 서버를 도입하고 있다(*e.g.*, Sydney Olympic[1], France98 World Cup[2], Netscape[3], 등). 클러스터 웹 서버는 여러 대의 웹 서버를 고성능

네트웍으로 연결하여 하나의 강력한 웹 서버로 활용하는 것이다. 최근에 연구되고 있는 클러스터 웹 서버들은 일반적으로 디스패처(dispatcher)와 여러 대의 웹 서버 노드로 구성되어 있어서 디스패처가 클라이언트로부터 전송되는 요구를 받아 뒤단의 웹 서버 노드들에게 전달한다. 클러스터 웹 서버는 하나의 가상 주소(virtual IP address)를 갖고 있는데, 이 가상 주소가 웹 클라이언트들에게 알려져 있어서 모든 요구는 이 가상 주소를 통해 디스패처에게 먼저 보내진다.

한편, 전자 상거래를 위한 웹 서비스가 증가함에 따라, 세션 무결성(session integrity)을 유지하는 것이 웹 서버의 중요한 기능 중 하나가 되고 있다. 세션 무결성은 특정 세션 기간 동안 클라이언트가 서버에게 요청한 일들이 일관되게 처리되도록 하는 것이다. 이를 위해서 웹 서버는 각 클라이언트를 위한 상태 정보를 관리해야 한다. 예를 들어, 전자 쇼핑몰에서 고객이 물건을 구입했을 때, 구입한 물품 목록이 쇼핑몰을 떠나기 전까지 일관되게 유지되도록 세션 정보를 관리해야 한다. 그러나 클러스터 웹 서버에서는 하나의 세션에 속한 클라이언트 요구가 클러스터 내의 서로 다른 웹 서버 노드에게 전달될 수 있기 때문에 세션 무결성을 유지하는 것이 쉽지 않다. 클러스터 웹 서버에서 세션을 관리하는 방법 중 하나는 한 세션에 속한 모든 요구가 하나의 웹 서버 노드에게 전달되도록 하는 것이다[4]. 즉, 클라이언트가 새로운 세션 설정을 요구하면 디스패처는 가장 적절한 웹 서버 노드를 선택하고 그 세션 기간 동안 도착한 클라이언트 요구들이 선택된 한 대의 웹 서버 노드에게 전달될 수 있도록 하는 것이다. 이와 같은 방법을 사용하게 되면 웹 서버 노드들이 클러스터 기반의 환경을 고려하지 않고 세션 관리를 할 수 있다는 장점이 있다. 그러나 클라이언트가 요구하는 웹 객체(HTML 문서 또는 이미지, 등) 파일이 다른 서버에 있을 경우, 그 서버에게 이를 요청하고 응답을 받아 클라이언트에게 전송해야 한다. 이러한 방법은 요구와 응답이 클러스터 내의 다른 웹 서버 노드를 거쳐와야 하기 때문에 이로 인한 오버헤드가 발생하게 된다.

본 논문에서는 클러스터 내의 웹 서버 노드간 데이타 선인출 방법을 제안함으로써 빠른 응답 시간을 갖는 클러스터 웹서버 시스템을 설계하고 선인출에 필요한 알고리즘들을 제안한다.

웹 서버의 선인출과 관련된 기존의 연구[5][6][7][8]들은 주로 클라이언트와 웹 서버 간의 선인출이 고려된 반면 본 연구는 클러스터 웹 서버 내의 서버 노드 간 선인출 방법을 연구함으로써 클러스터 웹 서버의 응답 속도를 향상시키기 위한 것이다.

본 논문의 구성은 다음과 같다. 제2절에서는 선인출 모듈을 포함하는 클러스터 웹 서버를 설계하고 제3절에서는 웹 객체에 대한 접근 확률(Access Probability)과 클라이언트 사이의 시간 간격(Inter-Reference Time) 그리고 웹 객체의 크기 등을 고려하는 선인출 알고리즘 세 가지를 제안한다. 제 4절에서는 시뮬레이션을 통해 제안하는 선인출 알고리즘들의 성능을 비교 평가하고 제5절에서 관련 연구를 소개한다. 마지막으로 제6절에서는 앞으로의 연구 방향을 제시하고 결론을 맺는다.

2. 선인출 모듈을 추가한 클러스터 웹 서버의 설계

본 절에서는 선인출 모듈을 포함하는 클러스터 웹 서버의 구조를 보이고 선인출 모듈과 다른 모듈들이 어떠한 방식으로 상호 작용하는지에 관해 설명한다. 이러한 웹 서버를 설계하는데 있어서 중요하게 고려된 점은 구현을 용이하게 하기 위해서 시스템에 가장 적은 수정이 가해지도록 하는 것이다. 그림1은 선인출 모듈을 가진 클러스터 웹 서버로 커널의 수정 없이 사용자 수준에서 구현할 수 있도록 설계되었다.

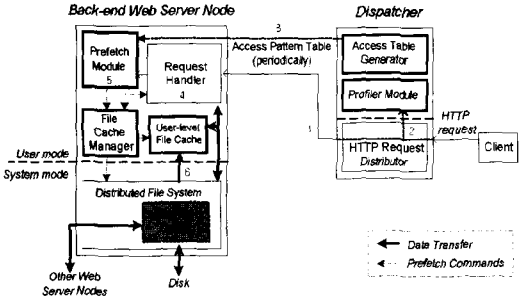


그림 1 선인출 모듈이 추가된 클러스터 웹 서버의 설계

이러한 시스템을 설계하는데 있어서 몇 가지 가정이 있는데, 첫째로 각 웹 서버 노드들은 NFS[9] 혹은 AFS[10]와 같은 네트웍 파일 시스템을 사용하여 파일을 공유한다. 또한, 디스패처는 세션 단위로 클라이언트 요구를 분산시키는 기능을 가지고 있어서 한 클라이언트 세션에 속한 모든 요구가 특정 웹 서버 노드에게만 전달된다. 마지막으로, 디스패처는 클라이언트가 세션을 시작할 때, 뒤단에 있는 하나의 웹 서버 노드와 일대일 매핑을 시켜서 하나의 클라이언트에 속한 요구가 특정 웹 서버 노드에게 보내지도록 하여 세션 일관성을 유지시킨다.

그림 1에 표시되어 있는 숫자는 웹 객체가 클라이언트로부터 요구되어 처리되는 과정과 그 다음에 요청될 수 있는 웹 객체를 선인출하는 과정을 나타낸 것이다.

1. 디스패처 내에 있는 요구 분산기(HTTP Request Distributor)가 임의의 클라이언트로부터 처음으로 HTTP 요구를 받으면 적절한 뒤단의 웹 서버 노드를 선택하고 그 클라이언트를 웹 서버 노드와 매핑 시킨다.

2. Profiler 모듈은 HTTP 요구를 받아 분석하고 선인출 하는데 필요한 정보들을 얻어낸다. 예를 들어, 웹 객체에 대한 요구 횟수, 연속되는 두 요구 사이의 지연 시간, 요구된 웹 객체의 크기, 등을 측정한다.

3. Profiler 모듈에서 수집된 정보들은 접근 테이블 생성기(Access Table Generator)로 보내져서 접근 패턴 테이블(Access Pattern Table)을 만든다. 접근 테이블 생성기는 주기적으로 접근 패턴 테이블을 뒤단의 웹 서버 노드에게 전달한다.

4. 웹 서버 노드에 있는 요구 처리자(Request Handler)는 HTTP 요구 분산기로부터 클라이언트 요구를 전달받고 요구된 파일이 사용자 수준 파일 캐쉬에 있는지를 파일 캐쉬 관리자(File Cache Manager)를 통해 살펴본다. 만약 선인출된 파일이 사용자 수준 파일 캐쉬에 있다면 그것을 읽어 클라이언트에게 전송하고 그렇지 않은 경우, 파일 시스템 호출을 통해 일반적인 방법으로 파일을 전송한다.

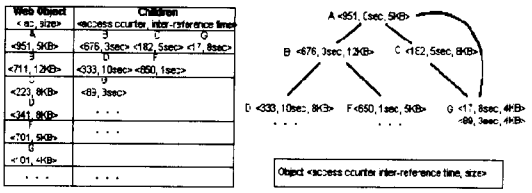
5. HTTP 요구의 내용은 선인출 모듈(Prefetch Module)에게 보내지고 다음에 요청될 클라이언트 요구를 예측하는데 사용된다. 선인출 모듈은 접근 테이블 생성자로부터 받은 접근 패턴 테이블로부터 다음에 요구될 웹 객체를 예측한다.

6. 어떤 웹 객체를 선인출할 것인지가 결정되면 파일 캐쉬 관리자는 사용자 수준 파일 캐쉬에 선택된 웹 객체 파일들을 읽어 놓는다.

그림 1의 각 모듈에 대한 구체적인 기능은 다음과 같다.

• 접근 테이블 생성기(Access Table Generator): Profiler 모듈로부터 수집된 HTTP 요구에 관한 정보를 요약하여 각 웹 객체에 대한 요청 빈도, 연속된 요구 사이의 평균 시간 간격 등을 알아낸다. 접근 패턴 테이블은 그림 2에서 보는 바와 같이 논리적으로 웹 객체 트리 형태로 보여진다.

그림 2 (b)에서 각 노드는 웹 객체를 나타내고 각각의 속성은 접근 횟수(Access Counter), 참조 사이의 시간 간격(Inter-Reference Time) 그리고 웹 객체의 크기로 괄호 안에 표시된다. 만약 어떤 객체가 객체 A 다음



(a) (b)

그림 2 접근 패턴 테이블 (a) 물리적 테이블 (b) 논리적 트리

괄호 안에 표시된다. 만약 어떤 객체가 객체 A 다음으로 요청되었다면 이 객체는 A의 자식 객체가 된다. HTML 문서와 같이 객체 내에 다른 객체의 링크를 포함하면 자식 객체를 갖게 되고 이미지 파일과 같이 링크를 포함하지 않는 객체는 자식 객체를 갖지 않는다. 접근 횟수는 부모 객체가 요청된 다음에 자식 객체가 요청된 횟수이다. 예를 들어, 그림 2에서 객체 B의 접근 횟수 C_B의 값 676은 객체 A 다음에 객체 B가 요청된 횟수이다. 접근 확률은 접근 횟수로부터 계산될 수 있는데, 이것은 선인출할 웹 객체를 예측하는 중요한 자료가 된다. 본 논문에서 제안하는 선인출 알고리즘들은 Jiang과 Kleinrock의 연구[6]에 기반하여 접근 확률을 구한다. 객체 B의 접근 확률은 $\min(1, C_B/C_A)$과 같이 구해지고 그림 2에서 웹 객체 B와 C의 접근 확률은 각각 0.71, 0.19가 된다. 참조 사이의 시간 간격은 부모 객체가 참조된 이후에 자식 객체가 참조될 때까지의 평균 지연 시간이고 웹 객체의 크기는 각 객체의 바이트 크기를 나타낸다. 접근 패턴 테이블은 뒤단의 웹 서버 노드에 있는 선인출 모듈에게 주기적으로 보내진다.

• 선인출 모듈(Prefetch Module): 요구 처리기는 클라이언트의 요청을 처리하고 그것을 선인출 모듈에게 넘겨준다. 선인출 모듈은 다음에 클라이언트가 요청할 가능성이 높은 웹 객체를 예측하고 선인출할 웹 객체의 목록을 파일 캐쉬 관리자에게 알려준다. 선인출 모듈이 사용하는 알고리즘은 제3절에서 자세히 설명한다.

• 파일 캐쉬 관리자(File Cache Manager): 파일 캐쉬 관리자는 선인출 모듈로부터 웹 객체의 목록을 받아 그 파일들을 사용자 수준 캐쉬에 읽어 놓는 역할을 한다. 요구 관리자가 파일 읽기를 요청하면 현재 사용자 수준 캐쉬에 요청된 파일이 있는지 살펴보고 만약 있다면 파일이 저장되어 있는 위치와 크기 정보를 요구 관리자에게 넘겨준다.

사용자 수준 캐쉬는 크기에 제약이 있어서 파일을 대체시키는 빈도가 높지만 커널 수준의 캐쉬를 거치기 때

문에 비교적 큰 커널 캐쉬를 이용하게 되는 효과를 얻을 수 있다.

• 요구 처리기(Request Handler): 요구 처리기는 클라이언트로부터 요청된 파일을 읽어오는 부분을 제외하면 일반적인 웹 서버 응용 프로그램과 같다. 요구 처리기는 HTTP 응답 메시지를 만들기 위해서 캐쉬 관리자를 호출하여 사용자 수준 파일 캐쉬에 요청된 파일이 선인출 되어 있는지 살펴본다. 만약 있다면 사용자 수준 파일 캐쉬에 있는 파일을 직접 전송하고 그렇지 않다면 파일 시스템 호출을 통해 전송한다. 이 과정을 구현하기 위해서는 웹 서버 응용 프로그램에 약간의 수정만이 요구된다.

3. 클러스터 웹 서버에서의 선인출 알고리즘

선인출을 하기 위해서는 다음에 요청될 웹 객체를 예측하기 위한 알고리즘이 필요하다. 본 절에서는 제안하는 세 가지 선인출 알고리즘을 소개한다.

3.1 접근 확률 기반 선인출 알고리즘

접근 확률 정보만으로 웹 객체 파일을 선인출할 것인지를 결정할 수 있다. 이러한 선인출 알고리즘을 접근 확률에 기반한 선인출 알고리즘(Access Probability-based Prefetch Algorithm, AP^2)이라 하고 접근 확률을 구하는 식은 제 2절에서 소개하였다. 이 알고리즘에서는 다음에 요구될 가능성이 있는 웹 객체들 중에서 가장 접근 확률이 큰 것을 우선적으로 선인출한다.

3.2 크기 가중치 접근 확률 기반 선인출 알고리즘

선인출의 목표는 웹 객체 파일을 읽어 오는 시간을 절약하여 응답 시간을 빠르게 하려는 것이다. 따라서, 웹 객체를 선인출 할 때, 시간적인 이득이 얼마나 클 것인지를 고려해야 한다. 만약에 객체 J_{IN}가 클라이언트로부터 요청되고 뒤단의 웹 서버 노드에 이것이 이미 선인출 되었다면 다른 노드 혹은 로컬 디스크에서 그 객체를 읽어오는 시간 즉, $t(J_{IN})$만큼이 절약된다. 그러나 선인출에 대한 예측이 항상 정확한 것은 아니기 때문에 웹 객체에 대한 접근 확률이 고려되어야 한다. 따라서 선인출의 시간적 이득은 다음과 같이 구할 수 있다.

$$B_{IN} = t(J_{IN}) \, X \, AP_{IN} \qquad (3.1)$$

여기에서 APIN은 객체 JIN의 접근 확률이다. 이 식을 이용한 선인출 알고리즘을 크기 가중치 접근 확률에 기반한 선인출 알고리즘(Size-weighted Access Probability-based Prefetch, SAP2)이라 하고 이 알고리즘에서는 가장 선인출 이득이 큰 웹 객체가 가장 먼저 선인출된다.

3.3 참조 시간 간격 및 접근 확률 기반 선인출 알고리즘

실제적인 선인출 이득이 최대가 되게 하기 위해서는 선인출 이득뿐만 아니라 비용이 고려되어야 한다. 즉, 선인출 비용보다 이득이 큰 객체를 선인출해야 한다.

선인출 비용은 크게 두 가지를 생각할 수 있다. 첫 번째는 선인출되는 객체가 많아져서 다음 클라이언트 요구에 대한 서비스가 늦어지기 때문에 발생하는 시간적 비용이다. 그러나 이 비용은 CPU나 네트웍 등의 자원이 사용되지 않을 때 선인출을 한다는 조건을 추가함으로써 0에 가깝게 할 수 있다. 두 번째는 메모리에서 쫓겨난 웹 객체 파일을 다시 읽어 들이는데 걸리는 시간 비용이다. 선인출을 할 때, 메모리에 여유 공간이 부족하면 LRU(Least-Recently-Used)와 같은 대체 알고리즘에 의해 메모리에 있던 다른 웹 객체 파일이 쫓겨난다. 이 때, 쫓겨난 객체 파일에 대한 요구가 발생한다면 이것을 다시 메모리에 읽어들이는 시간만큼이 필요하게 된다. 그림 3은 이러한 상황을 설명한다.

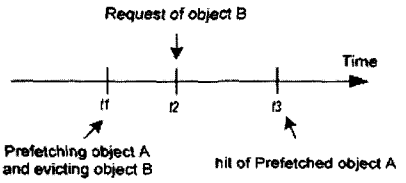


그림 3 선인출 시나리오

t_1 시간에 웹 객체 A가 선인출되고 웹 객체 B는 대체 알고리즘에 의해서 메모리 밖으로 쫓겨난다. 그러나 t_2 시간에 B에 대한 클라이언트의 요청을 받게 되면 또 다시 이것을 읽어오는 시간 비용이 들게 된다. 만약 선인출을 하지 않았다면 B를 읽어오는 시간 비용은 들지 않았을 것이다. 반대로 A가 t_2 이전에 요구되었다면 이후에 B에 대한 요구가 있어도 이것을 선인출로 인한 시간적 비용이라고 볼 수 없다. 왜냐하면, 선인출이 발생하지 않았더라도 객체 A에 대한 요구가 B에 대한 요구보다 먼저 있었기 때문에 B는 t_2 시점에 메모리에 존재할 수 없기 때문이다.

선인출의 두 번째 비용을 계산할 때, 선인출된 객체가 요구되기 전에 대체된 객체가 요구될 확률을 구해야 한다. 이러한 확률을 참조 시간 확률(Reference Time Probability)이라 한다. 이것은 스왑 아웃 시간 간격(Swap-out Time Interval)의 분포로부터 구할 수 있는데 이것은 메모리에 있던 객체가 대체된 이후에 다시 클라이언트로부터 요구되어 메모리에 읽어들여지기까지

걸린 시간을 뜻한다. 그림 3은 스왑 아웃 시간 간격의 분포를 실제 웹 서버 로그를 가지고 측정한 결과이다.

스왑 아웃 시간 간격을 측정하기 위해서 ClarkNet 웹 서버의 로그가 사용되었다[12]. x 축은 스왑 아웃 시간 간격을 나타내고 y 축은 주어진 시간 내에 메모리에서 쫓겨났다가 다시 읽어 들여지는 웹 객체의 비율을 나타낸다. 만약 선인출 하려는 객체가 과거의 기록으로 보아 약 t초 후에 요구될 예정이라면 선인출될 객체보다 대체될 객체가 먼저 요구될 확률은 t초 이하의 그래프 면적이 된다.

수식 (3.2)은 두 가지 선인출 비용에 관한 수식을 정리한 것이다.

$$C_{OUT}=t(J_{OUT})\,X\,RTP_{IN}+Delay_{next\_service} \quad (3.2)$$

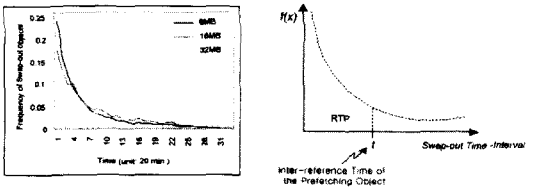
RTP_{IN}는 웹 객체 J_{IN}의 참조 시간 간격에 대한 J_{OUT}의 참조 시간 확률을 나타낸다. 웹 객체의 참조 시간 간격은 과거의 클라이언트 요구로부터 얻을 수 있다. 선인출 이득과 비용을 모두 고려한 식을 정리하면 다음과 같다.

$$PG_{IN}=t(J_{IN})\,x\,AP_{IN}-\{t(J_{OUT})\,x\,RTP_{IN}+Delay_{next\_service}\} \quad (3.3)$$

선인출의 실질적인 이득을 최대로 하기 위해서는 이득이 비용보다 큰 객체를 선인출해야 한다. 따라서 최대 이득은 다음과 같이 계산될 수 있다.

$$G_{MAX}=max\,\Sigma PG_{IN} \quad (3.4)$$

이러한 선인출 알고리즘을 참조 시간 간격 및 접근 확률에 기반한 선인출 알고리즘(Time and Access Probability-based Prefetch Algorithm, TAP^2)이라 한다.



(a) (b)

그림 4 스왑아웃 시간 간격의 분포 (a) 실제 로그를 이용한 결과 (b) 간략화된 그래프

4. 성능 측정 결과

본 절에서는 모의 실험을 통해서 선인출 방법의 효율성을 평가하고 제3절에서 제안된 세 가지 알고리즘을 비교 평가한다.

4.1 모의 실험 모델

모의 실험을 위한 클러스터 웹 서버는 한 대의 디스패처와 네 대의 웹 서버 노드로 구성되어 있다. 그림 5은 모의 실험 모델을 나타낸 것이다.

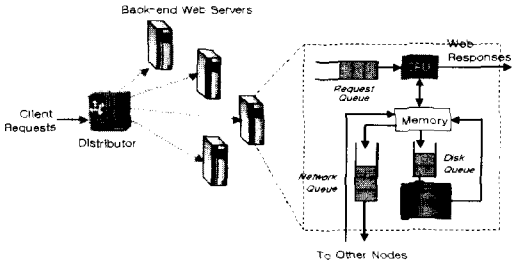


그림 5 모의 실험 모델

각 웹 서버 노드는 CPU와 메모리 그리고 하나 또는 그 이상의 저장 디스크로 구성된다. 각 HTTP 요구마다 설정과 해제 그리고 파일 전송에 걸리는 CPU 시간이 필요한데, 만약 요구된 객체가 메모리에 저장되어 있지 않다면 디스크 혹은 다른 노드의 메모리에서 읽어오는 시간도 필요하다.

그림 5의 큐들은 아직 처리가 끝나지 않은 클라이언트의 HTTP 요구를 갖고 있다. 요구 큐(Request Queue)에는 새롭게 웹 서버 노드에 도착한 요구와 다른 노드에게 응답 파일을 요청하여 이를 가다리는 요구들이 있다. 후자는 읽기 요구가 디스크나 네트워 읽기 동작 때문에 정체되어 있는 경우를 의미한다. 디스크 큐(Disk Queue)는 디스크에서 응답 파일을 읽을 때까지 기다리고 있는 요구들을 포함하며, 네트워 큐(Network Queue)는 네트워 읽기 동작 때문에 정체되어 있는 요구들을 포함한다. 디스패처는 전체 클러스터의 성능에 영향을 주지 않는다고 가정하였고 선인출은 CPU가 사용되지 않는 동안에만 수행하도록 하여 선인출 비용을 최소화하였다.

시뮬레이터에서 클라이언트 요구를 처리하기 위한 시간은 Pentium II 300-MHz 머신에서 측정된 값[11]을 이용하였다. 연결 설정과 해제에 드는 시간은 각각 145μsec이고 512bytes를 전송하는데 걸리는 시간은 410μsec이다. 디스크 접근률(disk access rate)은 28msec이고 4Kbyte 당 데이타 전송 시간은 410μsec이다. 한편, 클러스터 내의 노드 사이에 네트워 지연 시간은 4Kbyte 당 800μsec이다. 이 값은 Linux 기반에 TCP/IP 프로토콜을 사용하여 100Mbps Fast Ethernet으로 연결된 클러스터 시스템에서 측정된 값이다.

표 1 웹 서버 로그의 특성

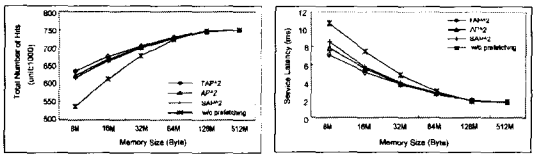
	ClarkNet	NASA	WorldCup98
Date	August 28-30, 1995	July 1-3, 1995	July 7, 1998
Avg. Requests per Sec.	2.92	0.81	18.35
Experiment Time	About 3.5 days	About 3.5 days	About 13 hours
Time Interval of Profile Distribution	About 7 hours(7.17)	About 7 hours(7.47)	About 1 hour(1.24)
Total No. of Requests	866815	245009	862086
Total No. of Actual Requests (Web status 200)	770000	220000	770000
Total No. of Image Files	605343	143559	659126
Avg. Object Size (KB)	9.8	27.1	5.2
Avg. HTML File Size (KB)	7.4	11.1	13.2
Avg. Image File Size (KB)	10.5	35.6	3.9
Total Working-set Size (MB)	400	250	400

4.2 모의 실험 입력

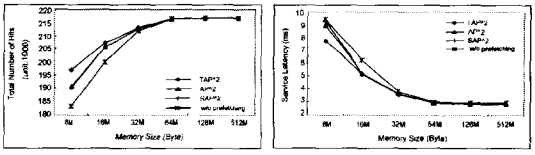
모의 실험의 입력으로 Internet Traffic Archive [12]에서 제공하는 NASA, ClarkNet, 98' France World-cup의 웹 서버의 로그를 사용하였다. 표 1은 모의 실험에 사용된 입력 로그들의 특성을 보여준다. 접근 패턴 테이블을 생성할 때는 응답 상태 코드가 성공(200번)인 것만을 사용하였고 동적 요구(dynamic request)에 대해서는 계산에 필요한 시간은 포함하지 않고 오직 응답 메시지를 전송하는 시간만이 필요하다고 가정하였다.

4.3 모의 실험 결과

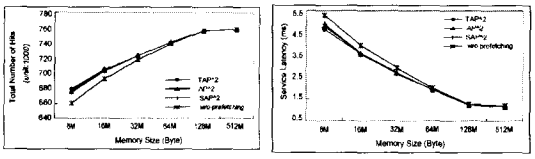
그림 6는 메모리에서 적중된 객체의 총 개수와 평균 서비스 지연 시간을 측정한 것이다. 그래프의 x축은 각 웹 서버 노드의 메모리 크기를 나타낸다. 모든 웹 서버 로그에서 선인출 알고리즘을 사용한 것이 사용하지 않은 것에 비해 더 많은 메모리 적중 개수를 보였다. TAP^{2}는 제안된 선인출 알고리즘 중 가장 높은 메모리 적중 개수를 얻었고 AP^{2}와 SAP^{2}가 각각 그 뒤를 따른다. 메모리 적중의 개수가 높은 순으로 서비스 지연 시간 역시 감소하였다. 즉, 요구가 들어왔을 때, 다른 노드의 메모리나 디스크에서 응답 파일을 읽어 전송하지 않고 미리 선인출하여 자신의 메모리에서 직접 응답 파일을 전송하여 빠른 응답 시간을 얻었다. 평균 서비스 지연 시간은 선인출 하는 것이 하지 않는 것에 비해서 8Mbyte 메모리를 사용했을 때, 약 10~25% 정도 감소하였다. 8Mbyte 메모리 크기는 입력으로 사용된 웹 서버 로그가 시뮬레이션 기간 동안 필요로 한 메모리 크기의 약 10%에 해당하는 것이다. 선인출 비용과 이득을 고려한 TAP^2 알고리즘은 제안된 다른 알고리즘에 비해 가장 서비스 지연이 적었는데, 8Mbyte 메모리에서 평균



(a) ClarkNet Access Log



(b) NASA Access Log



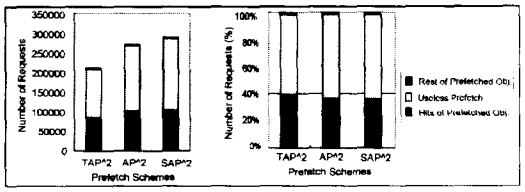
(c) WorldCup98 Access Log

그림 6 메모리 적중의 총 개수와 서비스 지연 시간

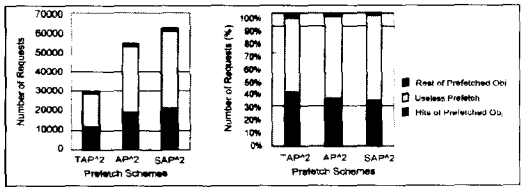
20.8%정도의 지연 시간을 감소시켰다. 98' France World-cup 로그는 선인출 효과가 크게 나타나지 않고 있는데, 이것은 초당 클라이언트 요구의 개수가 많아서 선인출에 필요한 CPU 시간이 부족하기 때문이다. 제 3 절에서 설명한 것처럼 선인출은 CPU가 사용되지 않는 동안 수행된다.

그림 7은 세 가지 선인출 알고리즘을 비교하기 위하여 선인출된 객체의 적중률을 측정한 결과이다. 각 막대

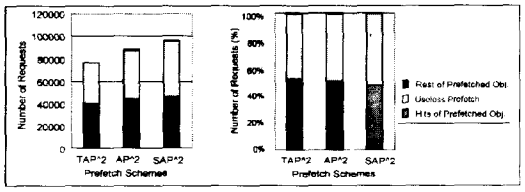
그래프의 높이는 총 선인출의 횟수를 나타내고 아래 막대 그래프부터 각각 선인출된 객체가 메모리에서 적중된 횟수, 적중되지 않고 대체된 횟수, 그리고 모의 실험이 끝난 시점에 메모리에 선인출 상태로 남아 있는 객체의 개수를 나타낸다. 제안된 선인출 알고리즘은 자식 객체 중 두 개 혹은 세 개의 객체를 선인출하고 그 중 한 개의 객체가 실제로 메모리에서 적중되었다. TAP^2 알고리즘은 적중률이 40%~50%로 가장 높았고, 그 다음은 AP^2, SAP^2 순이다. World-cpu 로그의 경우, 표 4.1에서 보는 바와 같이 평균 객체의 크기가 다른 것과 비교해서 상대적으로 작기 때문에 적은 메모리에서 실험한 결과가 비교되었다.



(a) ClarkNet Access Log(32MB Memory)



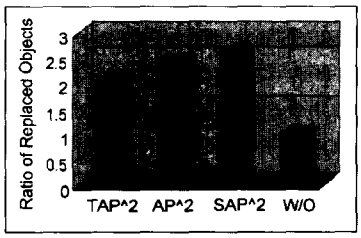
(b) NASA Access Log(32MB Memory)



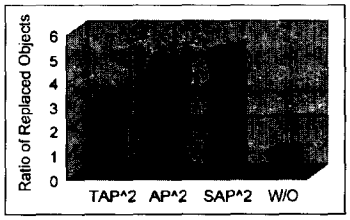
(c) WorldCup98 Access Log(16MB Memory)

그림 7 선인출 알고리즘의 적중률

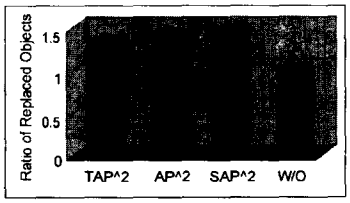
그림 8는 제안된 알고리즘에서 메모리 대체가 일어난 횟수를 그래프로 나타낸 것이다. 선인출 방법을 사용하지 않았을 때, 메모리에서 대체된 웹 객체의 개수를 1로 정규화하여 나타내었다. 선인출된 객체의 개수가 많고 선인출된 객체의 평균 크기가 클수록 대체가 많이 일어난다. 이 그래프를 통해, AP^2와 SAP^2 알고리즘이 TAP^2 알고리즘에 비해 불필요한 선인출을 많이 한다는 것을 알 수 있다.



(a) ClarkNet Access Log(32MB)



(b) NASA Access Log(32MB)



(c) WorldCup98 Access Log(16MB)

그림 8 대체된 객체 개수의 상대적 비교

5. 관련 연구

본 논문에서 제안된 선인출 방법은 클러스터 내의 웹 서버 노드 사이에서 발생된다. 그러나 웹 객체를 선인출하는 기본 아이디어는 클라이언트에서 서버의 웹 객체를 선인출하는 방법들을 연구한 기존의 연구들과 비슷하다.

Jiang과 Kleinrock[6]은 접근 확률에 기반한 선인출 방법을 제안한 바 있다. HTML 문서와 이미지들은 웹 사용자의 접근 히스토리나 네트웍 상태에 따라 다르게 선인출된다. 접근 확률은 웹 페이지 내의 각 링크가 요청된 횟수로 계산된다. 선인출 임계점은 클라이언트의 부하, 용량, 그리고 네트웍의 상태에 따라 계산되고 이 임계점보다 높은 접근 확률을 가진 웹 객체가 선인출된다.

Padmanabhan과 Mogul[8]도 접근 확률에 기반한 선인출 방법을 제안하였다. 이 연구에서는 서버가 사용자의 접근 로그를 바탕으로 웹 객체에 대한 관계 그래프(dependency graph)를 생성하고 이것을 통해 접근 확

률을 계산한다. 만약 객체 A가 요청되고 다음 n개의 요구 이내에 B가 요청되면 B는 A의 자식 객체가 된다. 사용자가 A에 대한 요청을 하면 웹 클라이언트는 계산된 접근 확률에 따라 자식 객체 중 선인출 할 것을 결정한다.

앞에서 소개한 두 가지 선인출 기법은 웹 서버와 클라이언트가 비교적 먼 거리를 두고 분산되어 있는 환경에서 클라이언트가 선인출하는 방법이다. 이러한 선인출 방법을 클러스터 내의 노드 간 선인출에 직접 적용하기는 어렵다. 왜냐하면, 본 논문에서 제안한 선인출 방법은 클러스터 내의 웹 서버 노드 사이에서 수행되기 때문에 빈번한 대체가 발생한다면 서버의 성능에 영향을 줄 수 있기 때문이다. 클라이언트에서의 선인출은 단지 앞으로 요청될 파일을 읽어 오는데 메모리를 사용하면 되지만 서버에서는 여러 클라이언트들로부터의 요구를 처리하기 때문에 자주 요청되는 객체가 메모리에 있도록 해야 총 서비스 지연 시간을 줄일 수 있다. 따라서 가까운 미래에 요청될 가능성이 큰 객체를 선별하여 선인출해야 한다. 또한, 선인출된 웹 객체가 메모리에서 사용되지 않고 대체되지 않도록 참조 사이의 시간 간격도 고려되어야 한다.

6. 결 론

본 논문에서는 클러스터 웹 서버 내에서의 선인출 방법을 제안하였다. 선인출에 필요한 알고리즘으로 접근 확률에 기반한 선인출 알고리즘(Access Probability-based Prefetch Algorithm, AP^2), 크기 가중치 접근 확률에 기반한 선인출 알고리즘(Size-Weighted Access Probability-based Prefetch Algorithm, SAP^2) 그리고 참조 시간 간격 및 접근 확률 기반 선인출 알고리즘(Time and Access Probability-based Prefetch Algorithm, TAP^2)을 고안하고 각 알고리즘의 성능을 모의 실험을 통해 알아보았다.

모의 실험 결과, 선인출 방법을 사용한 클러스터 웹 서버가 선인출을 사용하지 않은 것에 비해 서비스 지연 시간을 10~25% 감소시켰으며 특히, 본 논문에서 제안한 TAP^2 알고리즘이 가장 좋은 성능을 보였다. TAP^2 알고리즘은 선인출 비용과 이득을 고려하여 선인출할 객체를 선택하며 필요한 메모리 크기의 약 10%(8Mbyte)를 가지고 있을 때, 20.1%정도의 지연 시간 감소를 얻었다.

본 논문에서 소개된 선인출 방법은 객체 트리에서 자식 객체들에 대한 선인출만을 고려하지만 앞으로 한 단계 이상의 다단계 선인출 방법에 대한 연구도 수행될 것이다.

참 고 문 헌

[1] IBM Corporation, The Sydney 2000 Olympic Games, http://www.olympic.ibm.com/olympics/, 2000.

[2] Arlitt, M. and Jin T., "Workload Characterization of the 1998 World Cup Web Site," TR. Hewlett-Packard Lab, 1999

[3] Mosedale, D., Foos, W. and Mccool, R., "Lessons Learned Administering Netscape's Internet Site," IEEE Internet Computing, Vol.1, No. 2, pp. 28-35, 1997

[4] ArrowPoint Communications, "Cisco Web Network Services for E-Commerce: Implementing Secure and Scalable E-Commerce Network Services," White Paper, http://www.cisco.com/warp/public/cc/pd/si/11000/prodlit/csecm_wi.htm

[5] Duchamp, D., "Prefetching Hyperlinks," In Proc. Second USENIX Symp. on Internet Technologies and Systems, USENIX, pp. 127-138, 1999

[6] Jiang, Z. and Kleinrock, L., "An Adaptive Network Prefetch Scheme," IEEE Journal on Selected Areas in Communications Vol.16 No.3, pp. 358-368, 1998

[7] Kroeger,T.M., Long, D.D.E., Mogul, J., "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," In Proceedings of the USENIX Symposium on Internet Technologies and Systems, pp. 13-22, 1997

[8] Padmanabhan, V. N. and Mogul, J. C., "Using predictive prefetching to improve World Wide Web latency," ACM Computer Communication Review, Vol. 26, No. 3, pp. 22-36, 1996

[9] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B, "Design and Implementation of the Sun Network Filesystem," Proceedings of the USENIX Summer Technical Conference, 1985

[10] Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and Wext, M, "Scale and Performance in a Distributed File System," ACM Transactions on Computer Systems, Vol. 6, No. 1, pp. 51-81, 1988

[11] Pai, V., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W. and Nahum, E., "Locality-aware request distribution in cluster-based network servers," ACM SIGPLAN Notices, Vol. 33, No.11, pp. 205-216, 1998

[12] The Internet Traffic Archive, http://ita.ee.lbl.gov/index.html

박 선 영

1999년 충남대학교 컴퓨터공학과 학사, 2001년 한국과학기술원 전산학과 석사, 2001년 ~ 현재 한국전자통신연구원 컴퓨터시스템연구부 연구원. 관심분야는 Clustering, Web Server System, Storage Networking

박 도 현

1995년 한국과학기술원 전산학과 학사, 1997년 한국과학기술원 전산학과 석사, 1997년 ~ 현재 한국과학기술원 전산학과 박사과정 재학 중. 관심분야 Clustering, Operating System, Parallel Processing

이 준 원

1983년 서울대학교 계산통계학과 졸업 (학사). 1990년 Georgia Tech. 전산학과 (석사). 1991년 Georgia Tech. 전산학과 (박사). 1983년 ~ 1986년 (주)유공 근무. 1991년 ~ 1992년 IBM 근무. 1992년 ~ 현재 한국과학기술원 전산학과 부교수. 관심분야는 운영체제, 병렬처리 등

조 정 완

1964년 서울대학교 공과대학 전자공학과 졸업. 1968년 와이오밍 주립대학 전기공학과 석사학위 취득. 1973년 노스웨스턴 대학 전산학과 박사학위 취득. 1968년 ~1973년 미국 IBM 연구원. 1982년 ~ 1984년 삼성전자 고문. 1985년 ~ 1988년 금성소프트웨어 대표, 고문. 1973년 ~ 현재 한국과학기술원 전산학과 교수. 관심분야는 computer architecture, parallel processing, knowledge systems, man-machine interfaces