CDN 서비스를 위한 웹 캐싱 기법

이화여자대학교 반효경

1. 서 론

캐싱 기법은 저장 장치 계층 간의 속도 차이를 완충시켜주기 위해 컴퓨터 구조, 운영 체제, 데이타베이스 등의 분야에서 각각 캐쉬 메모리, 페이징 기법, 버퍼링 기법 등으로 널리 연구되어 왔다. 하지만, 최근 웹(WWW)의 보편화와 CDN 서비스의 활성화로 인해 단일 시스템 내에서 속도차가 있는 저장 장치 간에 이루어지는 캐싱 기법 뿐 아니라 원격지의 객체를 캐싱하는 기법의 중요성이 커지고 있다. 특히, 웹의 인기가 높아감에 따라 네트워크의 병목 현상과 그로 인한 사용자 지연 시간 증가 등이 점점 더 심각해져가는 추세이며, 이를 완화시키기 위해 최근 다양한 각도에서의 웹 캐싱 기법에 관한 연구가 활발히 이루어지고 있다.

웹 캐싱이란 웹 사용자에 의해 빈번히 요청되는 컨텐츠를 사용자와 지리적으로 가까운 웹 캐쉬에 보관하여 빠른 서비스를 가능하게 하는 기법을 말한다. 이와 같은 웹 캐싱 기법은 웹 서버 또는 웹 클라이언트 차원에서의 캐싱 뿐 아니라 웹 캐싱만을 전담하는 프락시 캐쉬에 의해 광범위하게 이루어지고 있다. 프락시 캐쉬는 통상적으로 일개 그룹의 웹 클라이언트에 대한 서비스 지연 시간을 줄이고자 하는 목적으로 사용되며 궁극적으로는 네트워크의 대역폭 절약과 함께 웹 서버의 부하를 줄이는 역할도 하게 된다. 이와 반대로 웹 서버 쪽에는 역방향 프락시 캐쉬(reverse proxy cache)가 사용될 수 있으며 이는 일개 그룹에 속한 웹 서버의 객체들을 캐싱하여 서버의 부하를 직접적으로 줄이는 역할을 하게 되며, 이는 궁극적으로 웹 사용자의 지연 시간을 줄이는 역할을 하게 된다.

캐싱 시스템의 성능에 중요한 역할을 차지하는 부분 중 하나가 캐쉬 교체 알고리즘(replacement algorithm)이다. 캐쉬 교체 알고리즘은 한정된 캐쉬 공간을 가지고 사용자의 지속적인 요청을 처리하기 위해 어떠한 객체를 캐쉬에 보관하고 있고 어떠한 객체를 삭제할지를 온라인으로 결정한다. 이러한 교체 알고리즘은 버퍼 캐싱과 페이징 등 전통적인 캐싱 시스템에서 LRU(Least Recently Used) 알고리즘 등으로 많이 연구되어 왔다. 하지만, 웹 캐쉬는 기존의 캐싱 시스템과 다른 여러 가지 독자적인 특성이 있어 이러한 특성을 반영하는 새로운 교체 알고리즘에 대한 활발한 연구가 이루어지고 있다. 본 고의 2장에서는 이러한 교체 알고리즘에 대해 최근 연구를 중심으로 살펴보도록 하겠다.

한편, 캐쉬에 보관된 웹 객체는 근원지 서버에서 변경될 수 있으므로 캐싱 시스템은 통상적으로 일관성 유지 기법(consistency policy)을 필요로 한다. 일관성 유지 기법은 사용자가 요청한 웹 객체가 캐싱되어 있는 경우 이 객체가 근원지 서버에 있는 객체와 동일한지를 확인하여 사용자에게 최신의 정보를 전달하기 위해 필요하다. 하지만, 전통적인 컴퓨터 시스템이 캐쉬 일관성을 유지하지 못할 경우 시스템 전체에 치명적인 문제점을 발생시킬 수 있는 것과 달리 웹 캐쉬에서는 일관성 유지 여부가 큰 문제점을 야기하지는 않는 경우가 대부분이며 일관성의 불일치에 대한 의미가 전통적인 시스템에서와 차이가 있다. 따라서, 웹 캐쉬에 적절한 일관성 유지 기법이 필요하며 이에 대해서 본 고의 3장에서 살펴보도록 하겠다.

웹 캐싱의 효과를 극대화시키기 위해 최근에는 웹 캐쉬 간의 공유 및 협력 기법에 관한 이슈가 중요시되고 있다. 본 고의 4장에서는 웹 캐쉬들 간의 공유 및 협력 기법에 대해 간단히 살펴보도록 하겠다. 이어서 5장에서는 웹 캐쉬에서의 사전 인출(prefet-

ching) 기법에 대해 살펴 보고, 6장에서는 동적 웹 컨텐츠(dynamic contents) 캐싱 기법에 대해 간단히 알아 보겠다.

2. 웹 캐쉬의 교체 알고리즘

캐싱 기법의 성능은 캐쉬 교체 알고리즘에 의해 크게 좌우된다. 캐쉬 교체 알고리즘은 미래의 참조를 미리 알지 못하는 상태에서 한정된 캐쉬 공간에 보관하고 있을 객체와 삭제할 객체를 동적으로 결정하는 온라인 알고리즘이다. 이러한 알고리즘은 가상 메모리의 페이지 교체 알고리즘과 버퍼 캐싱 등에서 광범위하게 연구되어 왔다. 하지만, 전통적인 캐싱 환경에서와 달리 웹 캐싱에서는 캐싱 대상이 되는 객체들의 사이즈와 인출 비용이 균일하지 않기 때문에 효율적인 교체 알고리즘의 설계에 어려움이 따른다. 예를 들어 페이징 기법에서는 동일한 크기의 페이지를 캐싱의 대상으로 하며 캐쉬 부재(cache miss)시 페이지를 디스크에서 주기억장치로 읽어오기 때문에 그 비용이 균일하다. 반면, 웹 캐싱에서는 객체를 가지고 있는 근원지 서버의 위치 및 특성에 따라 객체를 캐쉬로 읽어오는 비용이 다르며 하나의 URL에 대응하는 파일 단위로 캐싱이 이루어지기 때문에 캐싱 단위의 사이즈가 균일하지 않다. 따라서, 효율적인 캐쉬 교체 알고리즘은 이와 같은 객체의 이질성을 고려할 수 있어야 한다.

2.1 교체 알고리즘의 성능 척도

객체의 사이즈와 인출 비용이 균일한 전통적인 캐싱 환경에서는 사용자가 요청한 객체가 캐쉬에 존재하는 경우 그 효용이 모두 동일하므로 캐쉬 교체 알고리즘의 목표는 참조 가능성이 높은 객체를 캐쉬에 보관하여 식 (1)의 캐쉬 적중률(HR, hit rate)을 높이는 것으로 충분했다. 캐쉬 적중률이란 사용자의 총 요청 중 캐쉬에서 적중되어 서비스된 요청의 비율을 나타낸다. 하지만, 웹 캐싱에서는 객체들의 사이즈와 인출 비용이 균일하지 않기 때문에 객체들의 참조 가능성과 이질성을 함께 고려해서 객체들의 가치를 평가하는 것이 바람직하다. 식 (4)는 이와 같은 관점에서 캐싱의 목표를 비용 절감률(CSR, cost- savings ratio)로 정형화시킨 것이다. 만약 객체의 사이즈와 인출 비용이 모두 균일하다면 비용 절감률은 캐쉬 적중률과 동일한 의미를 가지게 된다.

비용 절감률의 식에서 객체의 비용 c_i는 다양한 환경에서의 캐싱 목표에 따라 다르게 정의할 수 있다. c_i를 객체의 사이즈로 정의하면 비용 절감률은 식 (2)의 바이트 적중률(BHR, byte hit rate)과 동일한 의미가 되고, 객체의 인출 지연 시간으로 정의하면 이는 식 (3)의 지연 감소율(DSR, delay savings ratio)과 동일한 의미가 된다. 바이트 적중률은 사용자에 의해 요청된 총 바이트 (데이터의 양) 중에서 캐쉬에 이미 존재하여 근원지 서버에서 받아올 필요가 없는 바이트 수의 비율을 나타내어 주는 척도이다. 지연 감소율은 캐쉬가 없을 경우의 총 사용자 지연 시간 중에서 캐쉬가 있음으로 해서 줄어드는 지연 시간의 비율을 나타내어 준다.

(1) 캐쉬 적중률 $= \sum h_i \,/\, \sum r_i$

(2) 바이트 적중률 $= \sum(s_i \cdot h_i) \,/\, \sum(s_i \cdot r_i)$

(3) 지연 감소율 $= \sum(d_i \cdot h_i) \,/\, \sum(d_i \cdot r_i)$

(4) **비용 절감률** $= \boldsymbol{\sum(c_i \cdot h_i) \,/\, \sum(c_i \cdot r_i)}$

- h_i : 객체 i의 캐쉬 적중 회수
- r_i : 객체 i의 총 참조 회수
- s : 객체 i의 크기
- d_i : 객체 i의 근원지 서버로부터의 인출 지연 시간
- c_i : 객체 i의 인출 비용

2.2 참조 가능성의 예측

효율적인 캐쉬 교체 알고리즘은 캐쉬 내의 객체들에 대한 재참조 가능성을 잘 예측하여 가까운 미래에 참조될 가능성이 높은 객체를 캐쉬에 보관해야 한다. 웹 객체의 재참조 성향에 주도적인 영향을 미치는 두 가지 특성은 시간 지역성(temporal locality)과 참조 인기도(reference popularity)이다. 시간 지역성이란 최근에 참조된 객체가 다시 참조될 가능성이 높은 특성을 말하며, 참조 인기도란 참조 회수가 많은 객체일수록 또 다시 참조될 가능성이 높은 특성을 말한다. 교체 알고리즘은 일반적으로 이들 두 가지 성질을 해당 객체의 과거 참조 기록을 통해 반영한다. 시간 지역성의 관점에서 대부분의 알고리즘들은 객체의 직전 참조 시각을 활용한다. 반면, LNC-R 알고리즘은 과거 k번째 참조 시각을 이용한다[3]. 이는 버퍼 캐싱 기법인 LRU-K 알고리즘을 이질형 객체를 위한 캐싱 기법에 맞게 활용한 것으로 볼 수 있다[9]. 참조 인기도 측면에서는 일부 알고리즘들의 경우 객체의 참조 회수를 이용하고 또 다른 부류의 알고리즘

들은 여기에 에이징(aging) 메커니즘을 추가하여 캐쉬 오염(cache pollution)을 방지하고 있다. 최근의 알고리즘들은 시간 지역성과 참조 인기도를 모두 고려하여 객체의 참조 가능성을 예측한다. LRV 알고리즘과 MIX 알고리즘은 직전 참조 시각과 객체의 참조 회수를 동시에 고려하여 참조 가능성을 예측한다[4, 7]. LNC-R 알고리즘이 사용하는 과거 k번째 참조 시각도 시간 지역성과 참조 인기도를 결합시키는 유용한 방법으로 볼 수 있다. 최근의 일부 알고리즘에서는 시간 지역성에 기반한 GD-SIZE 알고리즘에 참조 인기도를 결합시키는 방법을 강구하고 있다[15]. LUV 알고리즘에서는 객체의 참조 가능성을 예측하기 위해 과거의 모든 참조 기록을 이용한다[16]. LUV 알고리즘은 버퍼 캐싱에서 연구된 LRFU 알고리즘을 웹 캐쉬의 특성에 맞게 일반화시킨 방법이다[10].

2.3 객체의 이질성에 대한 고려

페이징 시스템과 같은 전통적인 캐싱 환경에서는 사용자가 요청한 객체가 캐쉬에 존재하는 경우 그 효용이 모두 동일하므로 캐쉬 교체 알고리즘은 참조 가능성이 높은 객체를 캐쉬에 보관하는 것으로 충분하다. 하지만, 웹 캐싱에서처럼 캐싱의 단위 객체들이 이질적인 환경에서는 참조 가능성 이외에 객체의 사이즈와 인출 비용을 고려한 합리적인 가치 평가를 해야 한다. 즉, 객체를 캐쉬에 보관하고 있을지를 결정하는 가치 평가 기준은 객체의 참조 가능성에 의한 가치와 캐쉬에서 적중될 경우 실제로 절약할 수 있는 비용을 동시에 고려해야 한다는 점이다. 한편, 이와 같은 비용에 대한 고려는 최적화하고자 하는 성능 척도와 밀접한 관련이 있다. 캐쉬 적중률을 높이기 위해서 교체 알고리즘은 사이즈가 작은 객체에 높은 가치를 주는 것이 좋다. 이는 한정된 캐쉬 공간에 많은 객체를 보관하여 캐쉬 적중률을 높일 수 있기 때문이다. SIZE와 LRU-MIN이 이와 같은 알고리즘에 속한다[1]. 하지만, 최근에 연구된 많은 알고리즘들은 캐쉬 적중률이 아닌 비용 절감률을 높이고자 한다. 이와 같은 알고리즘들은 크게 두 부류로 나누어 볼 수 있다. 그 첫째 부류는 웹 객체의 참조 가능성에 대한 예측치와 그 객체의 단위 크기당 비용을 곱하여 객체의 전체적인 가치를 평가하는 방법이다. 이와 같은 방법은 비용 절감률에 대한 기여도 측면에서 정규화된 가치 평가가 가능하다는 장점이 있다. LNC-R [3], SW-LFU [6], SLRU [8], LRV [4], LUV [16] 등의 알고리즘이 이 부류에 속한다. 두 번째 부류는 GD-SIZE 계열의 알고리즘이다. GD-SIZE 계열의 알고리즘 역시 객체의 사이즈와 비용을 고려하지만 첫 번째 부류의 알고리즘과 달리 객체의 참조 가능성과 이질성을 정규화된 방법으로 고려하지 않는다. GD-SIZE에서는 시간이 흐름에 따라 참조되지 않은 객체의 가치를 감소시키는 에이징 루틴이 객체의 인출 비용에 관계 없이 모든 객체들에 대해 동일한 값을 적용시킨다. 이는 첫 번째 부류의 알고리즘에서의 에이징 루틴이 객체의 비용에 비례하는 감소치를 적용하는 것과 차이가 있다.

2.4 알고리즘의 시간 및 공간 복잡도

캐쉬 교체 알고리즘이 실제 시스템에 유효하게 사용되기 위해서는 시간과 공간 복잡도 측면에서 현실성이 있어야 한다. 이는 캐싱 알고리즘이 온라인 알고리즘으로서 빈 공간이 필요할 때마다 즉시 삭제할 객체를 선별해야 하기 때문에 시간적인 제약 조건이 따르게 되며, 알고리즘의 운영에 필요한 정보의 저장에 지나치게 많은 공간이 사용되는 것은 현실성이 없기 때문이다. 프락시 캐쉬의 경우 통상적으로 캐쉬 내에 수십만 개에서 수백만 개의 객체가 존재하며 삭제 대상을 찾기 위해 매번 이들을 다 조사하는, 시간 복잡도 $O(n)$의 방법은 캐싱 시스템에 너무나 부담이 크다. 일반적으로 캐쉬 내에 있는 객체의 수가 n 개라고 할 때, 시간 복잡도 측면에서는 캐쉬 운영에 드는 시간이 $O(\log_2 n)$을 넘지 않는 것이 바람직하며, 공간 복잡도 측면에서는 $O(n)$-즉, 캐쉬 내의 객체 하나당 부가적으로 저장되는 정보가 상수인 $O(1)$-을 넘지 않는 것이 바람직하다. 전통적인 캐싱 환경에서의 교체 알고리즘들은 이러한 조건을 쉽게 만족하지만, 이를 객체의 사이즈와 인출 비용이 이질적인 웹 환경에 맞게 확장할 경우 복잡도가 높아져 캐쉬 운영을 위한 온라인 알고리즘으로 사용하기 어려운 경우가 많다.

시간 복잡도의 입장에서 각 알고리즘을 살펴보면 제일 먼저 LRU 알고리즘의 경우 가장 최근에 참조된 시각을 기준으로 객체들의 가치를 일렬로 세워놓고 새롭게 참조된 객체만 가치가 가장 높은 위치로 옮겨 주면 되므로 리스트 자료구조를 통해 $O(1)$의

시간 복잡도에 구현이 가능하다. LRU를 제외한 나머지 알고리즘들은 새롭게 참조된 객체라 하더라도 가장 가치가 높아지는 것은 아니므로 새롭게 참조된 객체의 가치에 맞는 위치를 찾아 주는 연산이 필요하다. 이와 같은 연산을 쉽게 하기 위해 대부분의 알고리즘들은 힙(heap) 자료구조를 이용하여 $O(\log_2 n)$의 시간 복잡도에 각종 캐쉬 연산을 구현하게 된다. 그러나, 최근 참조 시각을 이용하는 알고리즘에서는 객체의 가치가 시간이 지남에 따라 다르게 평가되기 때문에 참조되지 않은 객체들 간에도 가치의 대소 관계가 변할 수 있게 된다. 이와 같은 경우 힙을 이용한 구현이 불가능하기 때문에 매 순간 객체들의 가치 평가를 새롭게 해 주어야 하여 $O(n)$의 시간 복잡도가 필요하게 된다. LNC-R [3], MIX [7], LRV [4], SLRU [8], LRU-MIN [1] 등이 이러한 특성을 가지고 있다. 따라서, 이들 알고리즘은 근사적인 구현 방법을 사용하여 알고리즘의 시간 복잡도를 낮추게 된다. LNC-R 알고리즘의 경우 힙을 사용하고 주기적인 업데이트를 통해 근사적 구현을 하며, SLRU, LRU-MIN, LRV 등은 LRU 리스트를 객체의 가치에 따라 여러 개를 두는 방법을 통해 근사적인 구현을 시도하고 있다. SLRU와 LRU-MIN을 근사적으로 구현한 알고리즘은 각각 PSS와 LOG2-SIZE라는 별도의 이름으로 불린다[1, 8]. 하지만, LRV 알고리즘

표 1 웹 캐쉬의 교체 알고리즘

알고리즘	참조 가능성의 예측		객체의 이질성에 대한 고려	알고리즘의 목표 성능 척도	알고리즘의 복잡도 (n은 캐쉬 내 객체의 수)		장단점	
	최근 참조 성향 (시간 지역성)	참조 빈도 (인기도)			시간 복잡도	공간 복잡도	장점	단점
LRU	직전 참조 시각	사용 안함	고려 안함	BHR로 고정됨	$O(1)$	$O(n)$	• 구현이 간단함	• 이질성을 고려 안함 • 과거 참조 정보 중 특정 요소만 활용
LFU	사용 안함	참조 회수	고려 안함	BHR로 고정됨	$O(\log_2 n)$	$O(n)$		
SIZE [1]	사용 안함	사용 안함	사이즈만 고려	HR로 고정됨	$O(\log_2 n)$	$O(n)$	• 캐쉬 내에 많은 객체들을 보관	• 참조 가능성을 고려하지 않음
LRU-MIN	직전 참조 시각	사용 안함	사이즈만 고려	HR로 고정됨	$O(n)$	$O(n)$	• 캐쉬 내에 많은 객체들을 보관	• 시간 복잡도 • 이질성의 편협한 고려
HYBRID [2]	사용 안함	참조 회수	사이즈 및 인출 지연 시간 고려	DSR로 고정됨	$O(\log_2 n)$	$O(n)$ + 서버 정보	• 인출 지연 시간의 효과적인 추정	• 서버 정보 보관의 오버헤드 • 이질성의 편협한 고려
LNC-R [3]	과거 k번째 참조 시각	k번째 참조 시각에 기초	정규화된 고려	임의의 성능 척도로 확장 가능	$O(n)$	$O(k\,n)$	• 정규화된 가치 평가	• 시간 복잡도
LRV [4]	직전 참조 시각	참조 회수	고려 안함	BHR로 고정됨	$O(1)$	$O(n)$ + 파라미터 값	• 트레이스의 특성 고려	• 트레이스 분석이 선행되어야 함
			정규화된 고려	임의의 성능 척도	$O(n)$			
GD-SIZE [5]	직전 참조 시각	사용 안함	사이즈 및 인출 비용 고려	임의의 성능 척도로 확장 가능	$O(\log_2 n)$	$O(n)$	• 파라미터가 없음 • 임의의 성능 척도	• 참조 빈도를 고려하지 않음
MIX [7]	직전 참조 시각	참조 회수	사이즈 및 인출 지연 시간 고려	HR, BHR, DSR의 결합 형태	$O(n)$	$O(n)$	• 주요 요소들을 모두 고려	• 시간 복잡도
SW-LFU [6]	사용 안함	참조 회수	정규화된 고려	임의의 성능 척도로 확장 가능	$O(\log_2 n)$	$O(n)$	• 정규화된 가치 평가	• 최근 참조 성향을 고려하지 않음
SLRU [8]	직전 참조 시각	사용 안함	정규화된 고려	임의의 성능 척도로 확장 가능	$O(n)$	$O(n)$	• 정규화된 가치 평가	• 참조 빈도를 고려하지 않음 • 시간 복잡도
GDSF [15]	직전 참조 시각	참조 회수	사이즈 및 인출 비용 고려	임의의 성능 척도로 확장 가능	$O(\log_2 n)$	$O(n)$	• 임의의 성능 척도	• 정규화된 평가를 못함
LUV [16]	과거의 모든 참조 시각	참조 회수	정규화된 고려	임의의 성능 척도로 확장 가능	$O(\log_2 n)$	$O(n)$	• 정규화된 가치 평가 • 모든 과거 참조 기록 활용	• 파라미터 튜닝

의 경우 객체의 사이즈와 비용을 고려할 경우 이와 같은 근사적인 구현 방법도 불가능하게 된다. 따라서, 최근의 LRV 버전에서는 사이즈와 비용을 고려하지 않는 알고리즘만을 소개하고 있다[4]. GD-SIZE와 LUV 알고리즘은 최근 참조 시각을 이용하는 알고리즘이지만 참조되지 않은 객체들 간의 대소 관계가 변하지 않는 좋은 성질을 가지고 있어 힙을 이용해서 효율적인 구현이 가능하다는 장점이 있다[5, 16].

공간 복잡도 측면에서는 대부분의 알고리즘이 객체 당 수 바이트의 부가 정보만을 사용하는 제약 조건을 만족한다. 그러나, 몇몇 알고리즘들은 부가적인 정보를 사용한다. 예를 들면 Hybrid의 경우 서버별 정보를 유지하는데 부가적인 공간이 필요하며[2], LRV 알고리즘은 각종 파라미터 값들을 유지하는 공간이 필요하다[4]. LNC-R의 경우 객체당 과거 k번의 참조 시각을 유지하지만 현실적으로 k값을 2-3 정도로 사용하기 때문에 부담은 거의 없다고 할 수 있다[3]. LUV 알고리즘의 경우 과거의 모든 참조 기록을 이용해서 객체의 참조 가능성을 예측하지만, 과거의 참조 기록을 저장할 필요가 없고 하나의 값만을 유지하고 있으면 되는 좋은 성질을 가지고 있다[16].

3. 웹 캐쉬의 일관성 유지 기법

캐싱된 웹 객체가 근원지 서버에서 변경될 수 있으므로 웹 캐쉬는 사용자에게 유효한 정보를 전달하기 위해 일관성 유지 기법을 필요로 한다. 1장에서 이야기한대로 전통적인 컴퓨터 시스템에서와 달리 웹 캐쉬에서는 일관성의 불일치가 큰 문제점을 야기하지 않는 경우가 대부분이다. 따라서, 일반적인 웹 캐싱 시스템에서는 보통 적응적 TTL 기법(adaptive-TTL)과 같은 약한 일관성 유지 기법(weak consistency)을 사용한다. 약한 일관성 기법이란 사용자의 요청이 있을 때마다 캐싱된 객체가 변경되었는지를 근원지 서버에 확인하는 것이 아니라 변경되었을 가능성이 높은 경우에만 확인하는 기법을 말한다. 이와 상대되는 강한 일관성 유지 기법(strong consistency)은 반드시 최신 정보가 사용자에게 전달되는 것을 보장하는 기법으로서 웹 캐싱에서는 강한 일관성 유지 기법을 사용할 경우 이로 인한 웹 서버와 네트워크의 부담이 커서 득보다는 실이 많기 때문에 일반적으로 약한 일관성 기법을 주로 사용한다. 웹 캐쉬의 일관성 유지를 위한 전형적인 기법들은 다음과 같다.

· Polling-Every-time : 이 기법은 캐쉬 내에 이미 존재하는 객체에 대한 요청이 있을 때마다 근원지 서버에 객체의 변경 여부를 확인하는 방법이다.

· Invalidation : 이 기법은 근원지 서버가 자신의 객체를 캐싱하고 있는 모든 프락시 서버를 기록해 두었다가 해당 객체가 변경된 경우 해당 프락시 서버에 변경 사실을 알려주는 방법이다.

· Adaptive TTL (Time-To-Live) : 이 기법은 캐쉬 내에 이미 존재하는 객체에 대한 요청이 있을 때, 해당 객체에 대한 최종 변경 시각과 최종 확인 시각을 고려하여 변경되었을 가능성이 높다고 판단되는 경우에만 근원지 서버에 변경 여부를 확인하는 방법이다. 변경 가능성은 LMF(Last Modified Factor)에 의해 판단하며 LMF가 임계값(threshold) 이상인 경우에만 변경 가능성이 높다고 보아 근원지 서버에 변경 여부를 확인한다.

$$LMF = (C - V) / (V - M)$$

이 때, M은 해당 객체의 최종 변경 시각(Last Modified Time), V는 최종 확인 시각(Last Validated Time), C는 현재 시각(Current Time)을 나타낸다.

Polling-Every-Time과 Invalidation 기법은 사용자에게 변경되기 이전의 정보를 제공할 가능성이 없는 강한 일관성 유지 기법이지만, Polling-Every-Time의 경우 변경 여부를 매번 근원지 서버에 확인하는 과정에서 사용자 지연 시간 증가와 네트워크 유통량 증가, 웹 서버의 과부하 문제 등을 야기시키며, Invalidation 기법의 경우 근원지 서버가 객체를 캐싱하고 있는 모든 프락시 서버를 기억하고 있어야 한다는 부담이 있다. 따라서, Squid를 비롯한 대부분의 상용 프락시 서버들은 약한 일관성 유지 기법인 Adaptive TTL 방법을 주로 사용한다.

4. 웹 캐쉬의 공유 및 협력 기법

웹 캐싱의 효과를 극대화시키기 위해 최근에는 웹 캐쉬 간의 공유 및 협력 기법이 중요시 되고 있다. 웹 캐쉬 간의 공유는 일반적으로 인터넷 캐쉬 프로토콜(ICP, Internet Cache Protocol)에 의해 이루어진다[11]. ICP는 동료 프락시 캐쉬들 사이에서 웹 객체의 검색 및 전송을 지원하기 위한 프로토콜이다. 클라이언트가 프락시 서버에 웹 객체를 요구했는데 프락시

서버가 그 객체를 캐싱하고 있지 않은 경우 ICP에서는 모든 동료 프락시들에게 ICP 질의를 멀티캐스트(multicast)하여 누가 요청된 웹 객체를 가지고 있는지 확인한다. 만약 동료들 중 하나가 요청된 객체를 가지고 있다는 답신을 보내오면, ICP 질의를 보냈던 프락시는 객체를 가지고 있는 동료 프락시에게 HTTP 요청을 보내어 해당 객체를 받아 온다. 그 후, 받아온 객체를 클라이언트에게 전달하게 된다. HTTP는 웹 객체의 전송을 위한 프로토콜인 반면, ICP는 공유 웹 캐쉬들 간에 객체의 위치 확인을 위한 프로토콜로서 HTTP에 비해 매우 부담이 작은 프로토콜이다.

웹 캐쉬들 간의 공유를 위한 또다른 시도로는 캐쉬 배열간 경로 지정 프로토콜(CARP, Cache Array Routing Protocol)이 있다[12]. CARP는 공유 웹 캐쉬들에 동일한 웹 객체들이 중복 저장되는 것을 막기 위해 URL 공간을 분할하여 각각의 캐쉬는 해당 URL이 자신에게 해싱된 객체들만을 캐싱하게 한다. 디렉토리 기반 프로토콜(directory based protocol)에서는 공유 웹 캐쉬에 저장된 객체들의 위치 정보를 디렉토리에 유지하여 ICP 프로토콜의 멀티캐스트 부담을 없애고자 한다. 하지만, 디렉토리의 유지 및 관리에 또다른 부담이 생기게 된다. 디렉토리는 공유 프락시 자체 또는 별도의 디렉토리 서버에 구현하게 되며 디렉 토리 서버의 부하 부담을 줄이기 위해 여러 개의 디렉토리 서버를 두기도 한다[13].

5. 웹 캐쉬의 사전 인출 기법

웹 서비스의 응답 지연 시간을 줄이기 위한 방법의 일환으로 사용자에 의해 아직 요청되지 않은 객체를 미리 받아오는 사전 인출 기법(prefetching)에 관한 중요성이 증가하고 있다[14]. 웹 캐쉬에서의 사전 인출 기법은 예측 사전 인출 기법(predictive prefetching)과 대화식 사전 인출 기법(interactive prefetching)으로 나누어 볼 수 있다. 예측 사전 인출 기법은 웹 페이지들 간의 관계 그래프(depdendency graph) 등을 구성하여 하나의 웹 페이지가 참조되었을 때 새로운 웹 페이지가 참조될 확률을 과거의 참조 기록을 통해 예측하고 이 확률을 기반으로 사전 인출을 수행하는 방법을 말한다. 대화식 사전 인출 기법은 클라이언트가 HTML 문서에 대한 요청을 했을 때 웹 캐쉬는 캐싱하고 있던 HTML 문서를 미리 파싱(parsing)하여 그 문서에 포함(embed)되거나 연결(link)된 웹 객체를 미리 받아와서 클라이언트의 후속 요청에 곧바로 전달하는 기법을 말한다. 웹 캐쉬에서는 이와 같은 사전 인출 기법 외에도 일관성 유지 기법과 관련하여 유효성의 사전 확인 기법(prevalidation)에 관한 연구도 이루어지고 있다. 사전 인출 기법이 웹 객체 자체를 사용자가 요청하기 전에 미리 받아 오는 방법인 반면, 유효성의 사전 확인 기법은 캐싱된 객체의 유효성을 미리 확인해 두었다가 사용자가 해당 객체의 요청 시 웹 서버에 변경 여부를 확인하지 않고 곧바로 보내주는 방법을 말한다.

6. 동적 웹 객체의 캐싱 기법

지금까지 이야기한 웹 캐싱 기법들은 실시간으로 변하지 않는 컨텐츠, 즉 HTML, JPG, GIF 등의 정적 웹 컨텐츠(static Web contents)에 대한 캐싱에 초점을 맞추었다. 그러나, 최근의 연구 결과에 의하면 실시간성을 요구하는 컨텐츠, 즉 ASP, CGI 등 동적 웹 컨텐츠(dynamic Web contents)를 처리하는 부분이 전체 웹 서비스 지연 시간 중 상당한 부분을 차지하며, 이 부분이 웹 서버의 과부하를 일으키는 주요 요인으로 분석되고 있다.

동적 컨텐츠에 대한 웹 캐싱은 정적 컨텐츠의 캐싱에 비해 데이터의 관리에 어려움이 따른다. 이는 웹 서버가 동적 컨텐츠에 대한 요청을 받을 경우 단순히 저장 장치에 있는 파일을 그대로 보내주는 정적 컨텐츠 요청의 처리와 달리, 요청 받은 내용에 대해 특정 처리를 거친 후 그 결과물을 보내주어야 하기 때문이다. 즉, 동적 컨텐츠에 대한 요청의 결과물은 동일한 URL에 대한 웹 요청인 경우에도 서로 다른 결과를 초래할 수 있기 때문에 캐쉬에 저장하는 것 자체가 의미가 없을 수 있다. 그러나, 동적 컨텐츠에 대한 요청의 결과를 조사한 연구에 의하면 비록 동적 컨텐츠에 대한 요청의 결과물이 정확히 일치하지는 않는다 해도 상당히 유사한 부분을 포함하고 있는 것으로 알려져 있다. 이는 동적 컨텐츠의 요청에 대한 결과를 부분적으로 캐싱하여 추후 그 결과를 활용할 수 있다는 것을 의미한다. 하지만, URL에 대응하는 파일 전체를 캐싱하는 기존의 웹 캐싱 체계에서는 이와 같은 방법을 활용하는데 어려움이 있다. 따라서, 현재까지의 동적 컨텐츠 캐싱은 주로 웹 서버 내부에서 빠른 처리를 위해 웹 서버 전위(front-end)에 설

치하는 역방향 프락시 캐쉬(reverse proxy cache) 또는 웹 서버 가속기(Web server accelerator) 중 일부에서 활용되고 있는 실정이다. 하지만, 향후에는 HTML 등의 규약에 동적 컨텐츠 중 캐슁 가능한 곳을 부분적으로 표시하는 태그 등을 활용하여 일반적인 웹 캐쉬에서도 이와 같은 기법이 활용될 수 있을 것으로 기대된다.

7. 요약 및 전망

본 고에서는 웹 캐쉬의 교체 알고리즘과 일관성 유지 기법, 계층적 웹 캐쉬의 공유 및 협력 기법, 동적 웹 컨텐츠의 캐슁 기법과 웹 객체의 사전 인출 기법 등에 대해 간단히 살펴 보았다. 웹 사용량의 폭발적인 증가에 따라 웹 캐슁 시스템 시장은 잉크토미(Inktomi), 시스코(Cisco), 캐쉬플로우(CacheFlow) 등 선발 미국 기업들을 중심으로 최근 몇 년간 전 세계적으로 크게 증가해왔다. 하지만, 더욱 최근에는 이들 기업이 독주하는 체제로부터 탈피하여 국내 기업들 역시 뒤지지 않는 기술력으로 시장을 점유해 가고 있는 추세이다. 아라기술 등 국내 기술진의 웹 캐쉬 제품이 국제 웹 캐쉬 성능 평가 대회에서 1위를 차지하는 등 국내의 기술력이 이미 상당한 수준에 이르고 있다. 또한, 국내 시장의 규모도 작지 않아 국내의 교육기관, 관공서 등에서 인터넷 통신망 구축 시 웹 캐쉬는 필수 사항으로 탑재되고 있으며, 동시 처리 요청이 많은 커뮤니티 사이트, 대기업 사이트 등의 웹 서버에서는 역방향 프락시 서버를 통한 웹 서버 가속기 구축이 늘어나고 있는 추세이다. 또한, 웹 캐슁은 컨텐츠 전송 네트워크(Contents Delivery Network) 서비스에 의해 더욱 폭넓게 활용될 것으로 기대된다.

참고문헌

[1] S. Williams et al, "Removal Policies in Network Caches for World-Wide Web Objects," Proc. 1966 ACM Sigcomm Conf., 1996, pp.293-305.

[2] R.P. Wooster and M. Abrams, "Proxy Caching That Estimates Page Load Delays," Computer Networks and ISDN Systems, vol. 29, no. 8-13, 1997, pp. 977-986.

[3] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Design: Algorithms, Implementation, and Performance," IEEE Trans. Know ledge and Data Eng., vol. 11, no. 4, 1999, pp. 549-562.

[4] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache," IEEE/ACM Trans. Networking, vol. 8, no. 2, 2000, pp.158-170.

[5] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," Proc. USENIX Symp. Internet Technology and Systems (USITS 97), 1997, pp. 193-206.

[6] T.P. Kelly et al., "Biased Replacement Policies for Web Caches: Differential Quality-of-Service and Aggregate User Value," Proc. 4th Int' l Web Caching Workshop (WCW 99), 1999.

[7] N. Niclausse, Z. Liu, and P. Nain, "A New and Efficient Caching Policy for the World Wide Web," Proc. Workshop on Internet Server Performance (WISP 98), 1998, pp. 119-128.

[8] C. Aggarwal, J. Wolf, and P. Yu, "Caching on the World Wide Web," IEEE Trans. Knowledge and Data Eng., vol. 11, no. 1, 1999, pp. 94-107.

[9] E.J. O' Neil, P.E. O' Neil, and G. Weikum, "The LRU-k Page Replacement Algorithm for Database Disk Buffering," Proc. 1993 ACM SIGMOD Int' l Conf. Management of Data, ACM, 1993, pp. 297-306.

[10] D. Lee et al., "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," IEEE Trans. Computers, vol. 50, no. 12, Dec. 2001, pp.1352-1361.

[11] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP), version 2," 1997, http://icp.ircache.net/rfc2186.txt.

[12] V. Valloppillil and K.W. Ross, "Cache Array Routing Protocol v1.0," 1998, http://icp.ircache.net/carp.txt.

[13] S. Gadde and M. Rabinovich, "A Taste of Crispy Squid," Proc. 1998 Workshop Internet Server Performance (WISP 98), 1998.

[14] T. Kroeger, D. Long, and J. Mogul, "Exploring the Bounds of Web Latency Reduction from

Caching and Prefetching," Proc. USENIX Symp. Internet Technology and Systems (USITS 97), 1997, pp.13-22.

[15] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating Content Management Techniques for Web Proxy Caches," Proc. 2nd Workshop on Internet Server Performance, 1999.

[16] H. Bahn, S. Noh, S. L. Min, and K. Koh, "Efficient Replacement of Nonuniform Objects in Web Caches," IEEE Computer, Vol.35, No.6, pp.65-73, June 2002.

반 효 경

1997 서울대학교 계산통계학과 학사
1999 서울대학교 전산과학과 석사
2002 서울대학교 컴퓨터공학부 박사
2002~현재 이화여자대학교 컴퓨터학과 전임강사
관심분야:운영체제, 캐싱 알고리즘, 웹, 멀티미디어 시스템, 유전 알고리즘
E-mail:hyokyung@oslab.snu.ac.kr