효율적인 협동적 웹캐싱 기법

신 용 현†

요 약

인터넷 이용의 확산으로 네트웍 사용량은 급격하게 증가하였다. 최근에는 인터넷 트래픽의 많은 부분을 웹 트래픽이 차지하게 된다. 본 논문에서는 둘 이상의 프락시서버가 서로 유기적으로 협동하여 캐싱을 하는, 새로운 공유 웹 캐싱 (Shared Web Caching)기법을 제안한다. 제안된 기법은 DCOORD (Directory-based COORDinated Caching) 라고 이름 짓는다. DCOORD는 여러 개의 캐쉬서버가 각자 자기에게 캐싱되어 있는 오브젝트들을 유기적으로 조정하여 전체의 캐싱 비용을 줄인다.

실제 사용된 트레이스에 기반한 시뮬레이션으로 DCOORD를 다른 두 개의 대표적인 공유 웹 캐싱 기법인 ICP와 CARP와 비교한다. 실험에서는 기존에 많이 사용되는 메트릭인 히트율 대신, 오브젝트를 가져오기 위한 비용을 고려한 메트릭인 CSR을 주로 사용한다. 캐쉬 수의 증가 등의 실험을 통하여 DCOORD가 기존의 ICP나 CARP 보다 일관되게 많은 비용 절감 효과가 있음을 보여준다. 또한 DCOORD는 캐쉬 상호간의 메시지 수도 많지 않고, 캐쉬간의 트래픽도 상대적으로 적다. 본 논문에서 제안한 DCOORD는 협동적 웹캐싱에서 성능 향상 뿐 아니라 비용 절감에 효과적이다

키워드 : 오브젝트, 웹서버, 프락시서버, 웹캐싱, 협동적 웹캐싱

An Efficient Cooperative Web Caching Scheme

Yong-Hyeon Shin†

ABSTRACT

Nowadays, Internet is used worldwide and network traffic is increasing dramatically. Much of Internet traffic is due to the web applications. And I propose a new cooperative web caching scheme, called DCOORD which tries to minimize the overall cost of Web caching. DCOORD reduces the communication cost by coordinating the objects which are cached at each cache server.

In this paper, I compare the performance of DCOORD with two well-known cooperative Web caching schemes, ICP and CARP, using trace driven simulation. In order to reflect the cost factor in the network communication, I used the CSR(Cost-Saving Ratio) as our performance metric, instead of the traditional hit ratio. The performance evaluations show that DCOORD is more cost effective than ICP and CARP.

Key Words : Object, Web Server, Proxy Server, ,Web Caching, Cooperative Web Caching

1. 서 론

1.1 연구 배경

1990년대 초 CERN에서 WWW(World Wide Web)를 발표한 이후 최근에 와서는 인터넷에서의 트래픽의 상당한 부분이 웹에 관련된 트래픽이다. 인터넷 트래픽은 중복성이 있다. 같은 오브젝트에 대한 요구가 같은 클라이언트 또는 다른 클라이언트들 간에 반복적으로 발생한다. 이런 동일한 오브젝트들이 네트웍 상을 돌아다니면 사실상 불필요한 회선의 낭비와 트래픽을 발생한다.

따라서 클라이언트와 웹서버 사이에서 오브젝트들을 캐싱하여 멀리 떨어져 있는 웹서버에 직접 접근하지 않아도 클라이언트의 요구를 처리할 수 있도록 한다. 이러한 역할을 하는 것이 캐쉬서버(cache server), 프락시서버(proxy server)이다. 프락시서버는 그 프락시서버를 공유하는 내부 네트웍 안의 클라이언트들이 동일한 오브젝트에 대한 요구를 원래의 웹서버(origin Web server)로 보내지 않고 자체적으로 처리해서 요구한 클라이언트에 보내준다. 프락시서버의 사용은 클라이언트의 지연시간 감소, 네트웍 트래픽 감소, 웹서버 부하 감소와 같이 웹의 성능향상에 아주 중요한 역할을 한다[1,2].

하나의 프락시서버를 두는 웹 캐싱 시스템에서는 그 프락시서버가 문제가 생기면 전체 서비스가 중단된다. 또한 증가하는 클라이언트들이나 그들이 보내는 요구들을 적절히 처리하기 어려워진다. 즉 확장성이 부족하다. 그리하여 여러 개의 프락시서버를 사용하는 것이 확장성에서 뿐 아니라 성

† 정 회 원 : 서울산업대학교 컴퓨터공학과 전임강사
논문접수 : 2006년 7월 31일, 심사완료 : 2006년 9월 6일

능도 좋은 것으로 알려져 있다. 4개의 프락시서버로 운영하는 것이 하나의 프락시서버를 운용하는 것보다 14%~36% 히트율이 더 향상됨을 보인바 있다[3]. 이렇게 여러 개의 프락시서버를 사용하여 성능을 높이는 것은 서로가 협동(cooperative)하여 유기적인 관계를 가짐으로써 가능하게 된다. 이러한 공유 웹 캐싱의 방법에는 가장 단순하면서도 널리 사용되는 ICP, URL의 해싱한 값에 따라 오브젝트가 들어갈 캐쉬를 정하는 CARP외에도 WCCP[4], Cache Digest[5], Summary Cache[3] 등과 같은 방법들이 제시되어 왔다.

본 논문에서는 기존의 공유 웹 캐싱 방법들의 장단점을 분석하고, 그 중 대표적인 몇 가지 공유 웹 캐싱 방법보다 더 좋은 성능을 낼 수 있는 기법을 제안한다.

1.2 연구의 목적 및 범위

본 논문에서는 여러 개의 캐쉬서버를 사용하는 기존의 공유 웹 캐싱의 문제점을 분석하고 디렉토리를 이용한 새로운 기법을 제안한다. 제안된 방법은 요구된 오브젝트가 이미 공유 캐쉬를 이루는 캐쉬들 중 하나에 캐싱 되어 있을 때 먼저 각각의 캐쉬에 분산되어 저장되어 있는 디렉토리를 찾아서 해당 오브젝트가 어느 캐쉬에 있는지를 알아낸다. 그 후 그 캐쉬로 가서 해당 오브젝트를 가져와서 요구한 클라이언트에 전송해 주게 된다. 이 때 각각의 캐쉬 공간을 일정한 비율로 나누어 그 캐쉬의 로컬 클라이언트들이 최근에 많이 참조한 오브젝트들을 워킹셋 영역에 두어 오랫동안 캐쉬에 남아 있도록 한다. 미리 정의된 일정한 비율을 초과하는 오브젝트들은 공유영역에 두어, 빠른 시간 내에 퇴출되어 더 중요한 오브젝트가 다시 들어올 수 있도록 공간을 확보하게 해 준다.

결국 지역성(locality)를 이용하여 앞으로 계속 참조될 확률이 높은 오브젝트들을 더 많이 캐쉬에 두고, 그렇지 못한 오브젝트들은 빨리 퇴출되어 중복을 줄여서 더 많은 수의 오브젝트들이 캐쉬에 들어올 수 있도록 해준다. 이러한 것을 기존의 공유 캐싱 프로토콜과 비용을 고려하여 성능을 측정하고 비교한다. 또한 비용뿐 아니라 공유 캐쉬이기 때문에 필수적으로 발생하게 되는 캐쉬들 간을 오가는 메시지 수와 트래픽의 양도 비교 분석한다.

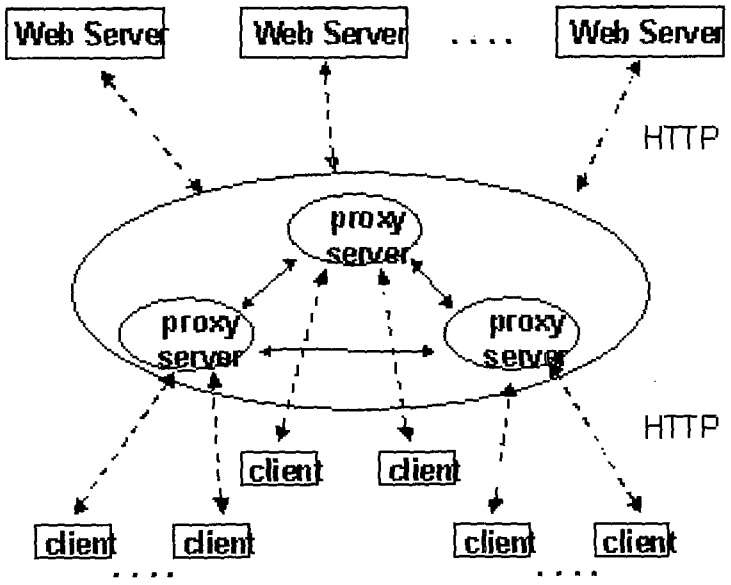
2. 협동적 웹 캐싱

2.1 개요

하나의 프락시서버로만 캐싱을 하는 것은 클라이언트 수가 많아짐에 따라 점점 많은 트래픽이 몰리게 되고, 그 하나의 프락시서버가 병목의 원인이 될 수 있고 시스템의 적절한 확장성을 기대하기 어려워진다. 하나의 프락시서버라도 정지된다든지 하는 문제가 발생 하면 거기에 연결되어 있는 수많은 클라이언트들이 서비스를 받지 못하게 된다.

그래서 여러 개의 서버를 두어 캐싱을 하는 방법들이 연구되어 왔고 실제로 사용되고 있다. 이렇게 둘 이상의 캐쉬들이 서로 유기적으로 협력하면서 캐싱을 하는 것을 공유 캐싱(shared caching) 혹은 협동적 캐싱(cooperative caching)이라고 한다. [3]에서는 4개의 프락시로 공유 캐쉬 서버를 구성했을 때 하나의 캐쉬로 서비스 하는 것 보다 14%~36% 까지 히트율이 더 좋아진다는 것을 보였다.

(그림 1)은 이러한 공유 프락시서버의 일반적인 모습을 도시하였다. 각각의 클라이언트들은 자신이 속한 프락시서버에 먼저 오브젝트가 있는지 요구를 보내게 되고 만약 없으면 다른 프락시서버들에서도 찾게 된다. 공유하는 모든 프락시서버에서 결국 해당 오브젝트를 발견하지 못한다면 해당 오브젝트를 가지고 있는 원래의 웹서버로 요구를 보내게 된다.



(그림 1) 협동적 프락시서버

2.2 ICP

ICP(Internet Cache Protocol)[7]는 1994년 Harvest 프로젝트[8]에 의해 만들어진 공유 캐쉬 프로토콜로 실제로도 많이 사용되고 있다. ICP는 어플리케이션 레이어 프로토콜로 UDP에 기반을 둔 프로토콜이다. ICP는 원하는 오브젝트가 어디에 있는지를 찾기 위해서 다른 캐쉬들에게 질의 메시지를 보내고 응답 받는 것을 기본 골격으로 하는 프로토콜이다. ICP는 초기에 만들어진 프로토콜로, HTTP 프로토콜이 아닌 보다 가벼운 프로토콜을 사용한다는 데에도 의미가 있다.

2.2.1 절차

클라이언트가 자신이 직접 연결되어 있는 로컬 캐쉬로 요구(HTTP request)를 보낸다. 이 로컬 캐쉬에 해당 오브젝트가 있으면(local hit) 클라이언트에 응답을 보낸다(HTTP response). 만약 로컬 캐쉬에 해당 오브젝트가 없으면 로컬 캐쉬는 공유 캐쉬를 구성하는 다른 모든 이웃 캐쉬(neighbor cache)들에게 해당 오브젝트를 가지고 있는지를 질의하기 위해 ICP_QUERY 메시지를 멀티캐스트(multicast) 한다. ICP_QUERY 메시지를 보낼 때는 타임아웃(timeout) 시간을 설정한다. 메시지를 보낸 로컬 캐쉬는 이 타임아웃 동안만 응답 메시지를 기다린다. 이웃하는 캐쉬들은 해당 오브젝트가 있으면 ICP_HIT을, 없는 경우에는 ICP_MISS를 보낸다. 모든 이웃 캐쉬들로 부터 ICP_MISS를 받거나, 설정된 타임아웃 시간이 지나면 공유 캐쉬 상에 해당 오브젝트가 없다고 간주하고 원래의 웹서버에 HTTP request를 보낸다. 이웃 캐쉬 중 하나라도 ICP_HIT의 응답을 보내면 로컬 캐쉬는 그 이웃 캐쉬에 HTTP request를 전달한다. 오브젝트가 있는 이웃 캐쉬에서 해당 오브젝트를 가져온 다음 최초 요구를 했던 클라이언트에 HTTP response를 보낸다.

2.2.2 특징

ICP는 초기에 나온 간단한 형태의 질의 메시지를 포함하는 프로토콜임에도 불구하고 여러 가지 문제점이 있다.

첫째, 메시지 수가 많아진다 둘째, 원하는 오브젝트를 찾기까지 상당한 시간이 걸린다. 오브젝트가 모든 캐쉬에 없는 경우, 오브젝트를 찾기 위해 모든 이웃 캐쉬에 (N-1)개의 메시지를 보내야 하고, 모든 캐쉬에서 ICP_MISS를 받거나 미리 설정된 타임아웃이 종료될 때까지 응답이 지연된다. 셋째, 또한 ICP의 경우 오브젝트가 여러 캐쉬들에 중복하여 저장 될 수 있다. 찾고자 하는 오브젝트가 로컬 캐쉬에 없을 경우 이웃 캐쉬에 있는 오브젝트를 로컬 캐쉬로 복사해 오기 때문이다. 경우에 따라서는 아주 많은 캐쉬에 중복하여 복사되어 저장 공간의 낭비를 가져올 수 있다.

2.3 CARP

CARP(Cache Array Routing Protocol)는 앞 절의 ICP 문제점의 많은 부분을 보완해 준다. 특히 ICP에서 발생할 수 있는 오브젝트의 중복을 제거한다. CARP는 오브젝트 URL의 해쉬 값에 따라 어떤 캐쉬에 저장이 될 것인지가 결정된다. 그러므로 오브젝트가 어느 캐쉬에 저장되어 있는지 찾을 때에도 ICP와 같이 다른 모든 이웃 캐쉬에 질의를 보내지 않고 해쉬 값에 따라 해당하는 이웃 캐쉬에 바로 HTTP request를 보내면 된다.

2.3.1 절차

클라이언트가 로컬 캐쉬서버에 오브젝트의 요구를 보내고, 로컬 캐쉬는 해당 오브젝트의 URL로 해쉬 키(hash key)값을 계산하여 해당하는 캐쉬서버로 HTTP request를 보낸다. 요구를 받은 캐쉬는 캐쉬 히트이면 HTTP response를 최초 요구한 클라이언트에 보내주고, 캐쉬 미스이면 원래의 웹서버로 다시 요구를 보낸다. 이 때 가져온 오브젝트는 해쉬한 키 값을 가지는 캐쉬서버에 저장된다. 로컬 캐쉬로 전송은 되지만 최초의 클라이언트에 전송을 하기 위한 것일 뿐 저장은 되지 않는다.

2.3.2 특징

CARP는 기존의 HTTP 프로토콜을 변경하지 않고 그대로 사용할 수 있다.

어떤 오브젝트가 어느 캐쉬에 저장이 되어야 하고, 어느 캐쉬에서 찾아야 하는지가 해쉬 함수에 따라 정해지므로 명확하게 하나의 캐쉬만을 찾으면 된다. 공유 캐쉬를 구성하는 캐쉬서버의 수가 증가해도 ICP와 달리 오브젝트를 찾는 시간에 별 영향이 없다. 해쉬 값에 따라 해당 캐쉬가 바로 지정되기 때문이다. 많은 질의 메시지가 오고 가야 되는 ICP와 달리 확장성이 좋다.

캐쉬들의 전체 공간이 일정하다고 할 때 ICP보다 훨씬 더 많은 오브젝트를 저장할 수 있어서 저장 공간을 효율적으로 이용할 수 있다. ICP는 하나의 오브젝트가 많이 중복되어 저장될 가능성이 많으나 CARP는 단 하나의 캐쉬에만 저장되기 때문에 공간을 절약한다.

공유웹 캐싱에는 그 이외에도 적응적(Adaptive) 웹캐싱[9, 10], Crisp[11], Cache Digest[5] 등이 있다.

3. 디렉토리 기반 협동적 캐싱

3.1 개요

앞 장에서 언급한 것처럼 공유 웹 캐싱 기법은 웹서버들의 부하를 상당히 많이 줄일 수 있을 뿐 아니라 웹 사용자들의 측면에서 느끼는 웹의 지연시간을 많이 줄일 수 있는 이점이 있다.

그러나 ICP를 이용한 공유 기법은 두 가지 결정적인 약점이 있다. 첫번째 문제점은 클라이언트가 속한 각각의 프락시서버에서 로컬 미스(local miss)가 발생할 경우 그 때마다 공유 캐쉬를 구성하고 있는 다른 모든 프락시서버들에게 해당 오브젝트를 가지고 있는지를 ICP_QUERY 메시지를 멀티캐스트 하여 질의하는 것이다. 즉 공유 캐쉬를 구성하는 프락시서버들의 수가 많아지면 멀티캐스트 하는 메시지

수가 기하급수적으로 증가하게 된다.

ICP의 두 번째 문제점은 공유 캐쉬를 구성하는 모든 프락시서버들이 그들이 가지고 있는 오브젝트들을 전혀 조정하지 않는다는 것이다. 즉 다른 프락시서버들이 어떤 오브젝트를 가지고 있는지에 상관없이 자신의 로컬 캐쉬에 있는 오브젝트들에만 관심이 있다는 것이다. 이러한 경우 캐쉬 전체의 공간은 미리 일정한 크기로 구성되어 있는데 동일한 오브젝트들이 여러 개의 프락시 캐쉬들에 중복되어 캐쉬되는 경우가 발생한다. 최악의 경우에는 공유 캐쉬를 구성하는 모든 프락시서버들이 모두 동일한 오브젝트들만을 가지고 있게 된다. 이는 아주 심한 공간의 낭비가 아닐 수 없을 것이다.

이에 반해 CARP는 모든 오브젝트들이 해쉬을 통해 단 하나의 프락시서버에만 저장이 된다. 그러나 CARP의 경우 ICP와 달리 공간은 효율적으로 사용하지만, 원격 히트(remote hit)라는 문제가 발생하게 된다. 이것은 로컬 클라이언트가 요구하는 오브젝트가 원격 프락시서버에 저장되어 있는 경우이다. 로컬 클라이언트가 원격 프락시서버에 저장되어 있는 오브젝트를 빈번하게 요구하게 되면 그 때마다 로컬 미스(local miss)가 일어나게 되고 원격에서 해당 오브젝트를 가져오게 된다. 이는 불필요한 비용을 발생하게 하는 것이다.

이 논문에서는 위의 대표적인 두 개의 공유캐쉬 시스템의 단점을 보완하기 위한 새로운 기법을 제안한다. 이 새로운 기법을 DCOORD (Directory-based COORDinated Caching)라고 명명한다.

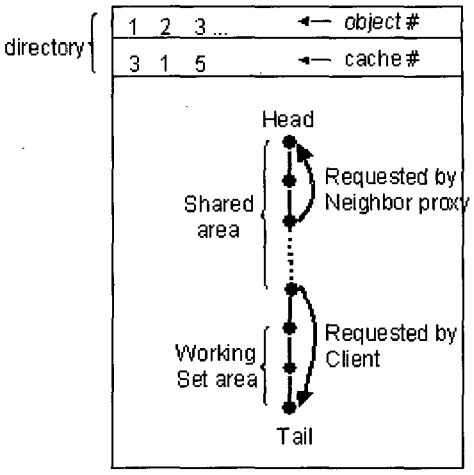
웹에서는 대부분의 참조시간 사이가 상대적으로 짧은 시간 안에 이루어진 걸로 알려졌다[6]. 즉, 오브젝트들에 대한 참조 대부분이 얼마 안가서 다시 재참조(re-reference)된다는 것이다. DCOORD에서는 이러한 부분을 워킹셋으로 두고 특별히 관리하게 된다.

3.2 내부 데이터구조

(그림 2)는 하나의 프락시 캐쉬 내부의 자료구조를 보여준다.

DCOORD에서는 이미 캐쉬된 오브젝트들의 위치 정보를 디렉토리에 넣어서 관리하게 된다. 따라서 ICP와 같이 오브젝트의 위치를 찾기 위해 메시지를 멀티캐스트할 필요가 없다. 오브젝트들의 위치 정보를 가지고 있는 디렉토리는 공유 캐쉬를 구성하고 있는 여러 프락시서버들에 분산되어 저장되게 된다. 각 프락시서버에 있는 디렉토리에는 URL을 키(key)로 자신에게 해쉬되는 오브젝트들에 대해서만 배타적인(exclusive) 정보를 갖게 된다. 즉 어떤 오브젝트가 어떤 프락시서버에 저장되어 있느냐 하는 정보는 그 오브젝트의 URL로 해쉬한 결과값을 갖는 프락시서버의 디렉토리에만있다는 것이다.

오브젝트들의 위치 정보를 가지는 디렉토리를 제외한 나머지 부분에는 그 캐쉬에 캐쉬되어 있는 오브젝트들을 일반



(그림 2) DCOORD 내부의 자료구조

적인 LRU 리스트 구조로 저장한다. 여기서 꼬리(tail)부분은 가장 최근에 참조한 오브젝트가 위치하게 되고, 머리(head) 부분은 새로운 오브젝트가 들어올 경우 가장 먼저 쫓겨나가게 될 오브젝트가 위치하게 된다. DCOORD에서는 이러한 LRU 리스트를 두 부분으로 나누어서 관리하게 된다. 로컬 프락시서버의 사용자들이 자주 요구하는 오브젝트들을 특정한 영역에 넣어 관리하는데 이 영역을 워킹셋 영역(Working Set area)이라 한다. 그 이외의 나머지 오브젝트들이 차지하는 영역을 공유 영역(Shared area)이라고 한다.

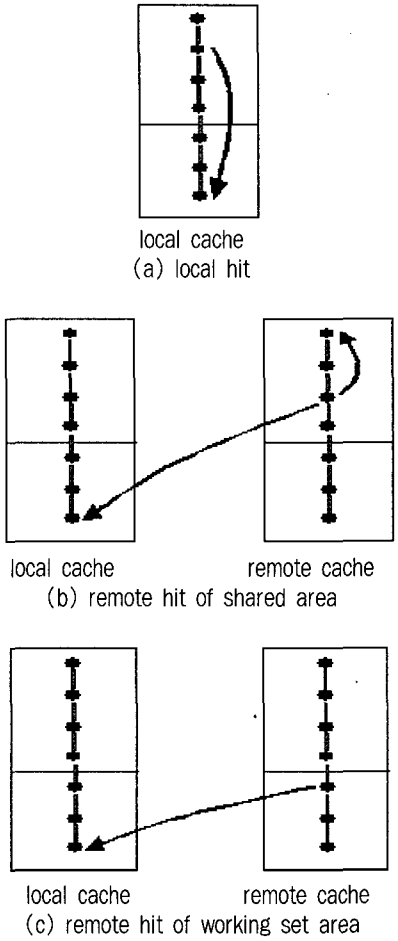
로컬 프락시서버에 연결되어 있는 클라이언트들이 오브젝트를 요구했을 때 다음과 같은 3가지 경우가 발생하게 된다.

- 로컬 히트(local hit) : 찾고자 하는 오브젝트가 로컬 캐시에 있을 경우
- 원격 히트(remote hit) : 찾고자 하는 오브젝트가 로컬 캐시에 없고, 원격의 이웃하는 프락시서버들 중의 한 곳에서 찾았을 때
- 캐쉬 미스(cache miss) : 원하는 오브젝트를 공유하는 프락시서버들 어느 곳에서도 찾을 수 없을 때

로컬 히트일(그림 3)(a) 경우 그 오브젝트는 시간 지역성(temporal locality)에 의해 앞으로 계속 참조될 가능성이 아주 높으므로 될 수 있는 한 오랫동안 캐시에 머물러 있을 수 있도록 LRU 리스트의 꼬리부분으로 이동한다.

원격 히트일 경우 오브젝트의 이동은 참조된 오브젝트가 워킹셋 영역에 있느냐 공유 영역에 있느냐에 따라서 달라지게 된다. 공유영역 (그림 3)(b)에 있으면 이 오브젝트를 빨리 내보낼 수 있도록 리스트의 머리로 이동한다. 이것은 이 오브젝트가 자신을 요구한 캐시로 복사되어 갈 것이기 때문에 가까운 미래에 쫓겨나간다 해도 전역(global) 캐쉬 상에는 존재하기 때문에 큰 손실은 아니기 때문이다. 또한 빨리 쫓겨나가게 하여 전체 캐쉬에서는 좀더 많은 오브젝트를 가질 수 있는 이점이 있다.

원격 히트이고 이 오브젝트가 워킹셋 영역에 있으면(그림



(그림 3) 캐쉬 히트일 때 오브젝트의 위치 변화

3)(c) 로컬 클라이언트들이 계속해서 참조할 가능성이 아주 많다. 그러므로 원격으로 복사해 가더라도 이 캐쉬에서도 오랫동안 남아 있을 수 있도록 워킹셋 영역에 그대로 둔다. 이는 자주 참조되는 오브젝트의 경우 중복 캐싱을 허용하여 전체적인 히트율을 높일 수 있다.

위의 세 가지 경우 각각의 비용을 로컬 히트 0, 캐쉬 미스 1, 그리고 원격 히트를 0과 1사이의 값으로 정한다. 즉 로컬 히트는 클라이언트가 속한 가장 가까이 있는 프락시서버가 요구된 오브젝트를 가지고 있으므로 비용은 최소한이 될 것이고, 미스일 경우 어느 프락시서버도 해당 오브젝트를 가지고 있지 않으므로 그 오브젝트의 URL이 속한 멀리 있는 웹서버로 HTTP 요구를 보내서 응답을 받아와야 할 것이다. 이 때의 비용이 가장 크니까 비용을 1로 정한다. 원격 히트에서의 비용은 위 두 가지 경우의 사이가 될 것이다.

3.3 알고리즘

(그림 4)는 DCOORD의 대략적인 메시지들의 절차를 도시하였다.

(1) 로컬 클라이언트가 특정 오브젝트를 요구한다.

(2) 해당 클라이언트에 접속되어 있는 프락시서버가 이 클라이언트가 요구한 오브젝트가 어느 캐쉬에 저장되어 있는지를 찾기 위해 URL을 해싱하여 그 캐쉬에게 오브젝트의 위치정보를 질의한다.

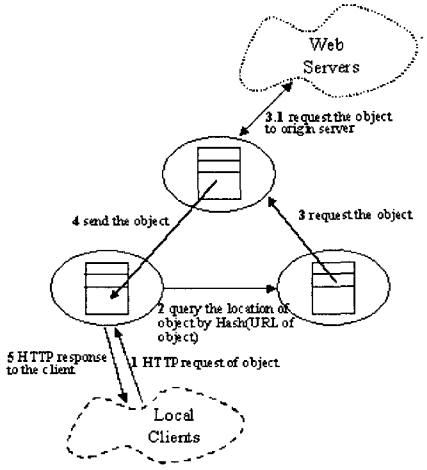
(3) 해당 오브젝트의 위치정보를 가지고 있는 캐쉬는 디렉토리에서 이 오브젝트를 가지고 있는 캐쉬를 찾아 그 캐쉬에 다시 오브젝트를 요구한다.

(3.1) 오브젝트를 가지고 있어야 되는 캐쉬에 해당 오브젝트가 없다면 URL이 속한 웹서버에 요구를 해서 결과를 받아 저장한다.

(4) 해당 오브젝트를 가진 캐쉬는 그 오브젝트를 최초의 로컬 프락시서버로 전송한다.

(5) 로컬 프락시서버는 해당 오브젝트를 최초에 요구한 클라이언트에 전송해 준다.

(그림 4)는 DCOORD의 메시지 흐름도이고, (그림 5)는 개략적인 알고리즘이다.



(그림 4) DCOORD의 메시지 흐름도

```
if object i is requested by a client
{  if object i is in the local cache                    // local hit
          move object i to the tail of the list and send i to the
     client
  else
  {       k = Hash(URL of object i)
          request the location of i to neighbor proxy k
          if response from k is one of the neighbor caches
                  try to fetch i from that neighbor cache
          if object i cannot be fetched from neighbors
                  fetch i from the origin Web server
          send object i to the client
          add object i to the tail of the list
          while size of i is larger than free space in local
     cache
                  remove the head of the list
          while size of the current working set is larger than
     working set rate
                  remove the head of working set area
  }
}
else if object i is requested by one of the neighbor caches
{ send object i to that requesting neighbor cache          //
     remote hit
   if object i is in the local cache and is not in the working set
          move object i to the head of the list
}
else if location of object i is requested by neighbor cache m

{ if the location of object i is stored in the directory
          send the location to m
                              // the cache has the directory info of
     the object
  else send NULL to m
  the location of object i is updated to m in the directory
}
```

(그림 5) DCOORD 알고리즘

3.4 성능분석

3.4.1 트레이스와 메트릭

본 논문에서는 성능평가를 위해 NLANR(National Laboratory for Applied Network Research)의 샌디에고(SD)와 어바나샴페인(UC) 두 곳의 로그를 사용하였다.

일반적으로 웹캐싱에서는 성능평가를 위한 메트릭(Metric)으로 히트율(Hit Ratio)과 바이트 히트율(Byte Hit Ratio)을 많이 사용한다. 히트율은 전체 요구의 수에 대해 클라이언트의 요구가 캐쉬에 있는 수의 비율이고, 오브젝트의 크기에 가중 값을 주는 바이트 히트율도 많이 사용된다. 이 때 작은 오브젝트를 많이 요구하는 경우 히트율이 높아도 큰 오브젝트 한두 개가 캐쉬에 있는 것 보다 전체 네트웍 트래픽을 많이 줄이지는 못할 것이다. 그 외에도 지연시간, 산출량 등이 메트릭으로 사용될 수 있다.

본 논문에서는 주요한 메트릭으로 히트율을 사용하기보다는 전체적으로 얼마나 비용을 절약하는가 하는 데 초점을 맞춘다. 즉 모든 히트가 다 같은 비용을 수반하지는 않기 때문이다. 본 논문에서 다루는 공유 웹 캐싱에서는 여러 개의 캐쉬서버가 오브젝트를 캐싱하게 된다. 이때 오브젝트가 어떤 캐쉬에 있느냐에 따라 그 오브젝트를 최종적으로 클라이언트에 전달해 주는 전체 비용이 달라질 수 있다.

본 논문에서는 CSR(Cost-Saving Ratio)을 메트릭으로 사용하는데 CSR은 로컬 히트, 원격 히트, 캐쉬 미스 각각에 대해 얼마나 비용을 절감했느냐는 정도가 된다. 캐쉬히트가 되면 가까운 캐쉬에서 오브젝트를 가져오니 비용이 절감될 것이고 캐시미스가 되면 멀리 떨어져 있는 원래의 웹서버에서 오브젝트를 가져와야 되므로 가장 비용이 많이 들 것이다.

$$CSR = \sum c_i \cdot h_i \;/\; \sum c_i \cdot r_i$$

c_i : cost to fetch object i
h_i : number of hit (object i)
r_i : number of reference (object i)

그 비용의 상대적인 비교는 로컬히트 < 원격히트 < 캐쉬미스의 순서가 된다. 이 때, CSR은 비용 절감이므로 로컬히트일 때 가장 크고 캐쉬 미스일 때는 비용을 절감할 수 없으므로 가장 작다.

3.4.2 CSR

본 논문에서는 제안한 DCOORD를 대표적인 두 개의 공유 웹 캐싱 프로토콜인 ICP, CARP와 비교한다. 기존에 주로 사용하였던 히트율 대신 비용에 기반한 메트릭을 사용한다. 여기서 비용은 원하는 오브젝트를 최초 HTTP 요구를 시작한 클라이언트에 보내는 데 필요한 오버헤드를 말한다. 원하는 오브젝트를 클라이언트에 보내는 비용은 그 오브젝트가 로컬 캐쉬, 원격 캐쉬, 웹서버 중 어디에 있느냐에 따라 비용이 달라질 것이다. 본 실험에서는 비용을 로컬 히트일 경우 0, 캐쉬미스일 경우 1로 설정한다. 원격히트일 경우의 비용은 공유 캐쉬를 구성하는 캐쉬들이 얼마나 밀접하게(tightly-coupled) 연결되어 있느냐에 따라 그 값이 크게 달라질 수 있을 것이다. 즉 캐쉬들이 한 대학교 내의 학과별로 있다면 다른 원격 캐쉬에서 오브젝트를 가져오는 비용이 로컬 캐쉬에서 가져오는 비용보다 조금 더 클 것이고, 행정구역상의 각 지역(서울, 부산)별로 있다면 상당히 커질 수 있을 것이다.

CSR(Cost-Saving Ratio)은 이러한 비용에 의존하는 것으로 비용이 적을수록 CSR은 커진다. 즉 더 많은 비용이 절감되는 것이다.

CSR은 아래와 같이, 로컬 히트일 때의 비용 절감과 원격 히트일 때의 비용 절감을 모두 합하여 전체 참조회수로 나누어 계산하게 된다.

$$CSR = \sum c_i \cdot h_i \;/\; \sum r_i$$
$$= (\sum (lc_i \times lh_i) + \sum (rc_i \times rh_i)) \;/\; \sum r_i$$

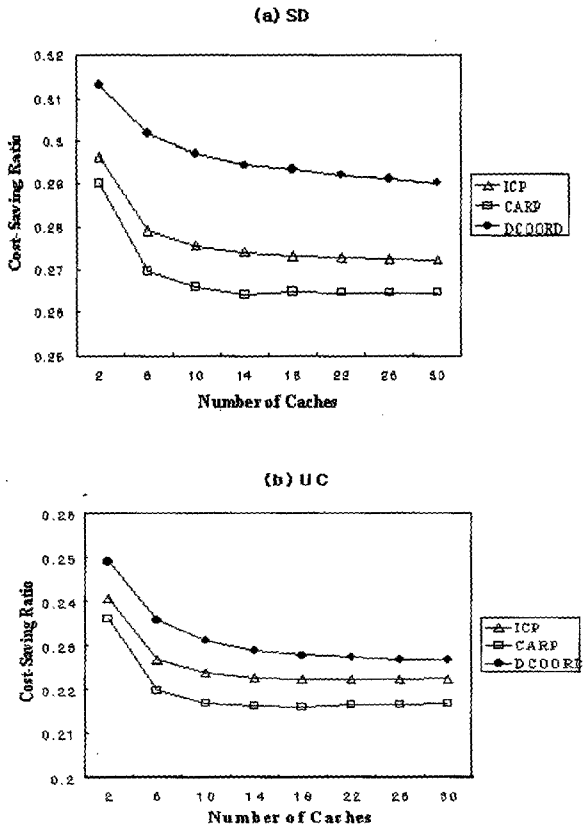
lc_i : 오브젝트 i에 대한 로컬 cost saving
rc_i : 오브젝트 i에 대한 원격 cost saving
lh_i : 오브젝트 i에 대한 로컬 히트 수
rh_i : 오브젝트 i에 대한 원격 히트 수
r_i : 오브젝트 i에 대한 전체 참조 회수

(그림 6)은 캐쉬 수의 증가에 따른 CSR을 보여준다. 캐쉬의 수를 2개에서 30개까지 변화시켜 보았다. 캐쉬 수가 증가함에 따라 CSR이 초기에는 다소 떨어지나 10개 이후에는 크게 떨어지지 않는다. 캐쉬의 수가 많아지게 되면 로컬 캐쉬가 전체 캐쉬에서 차지하는 비중이 줄어들게 되므로 로컬 히트보다는 원격 히트가 아무래도 많아지게 되므로 CSR도 줄어드는 경향을 보인다.

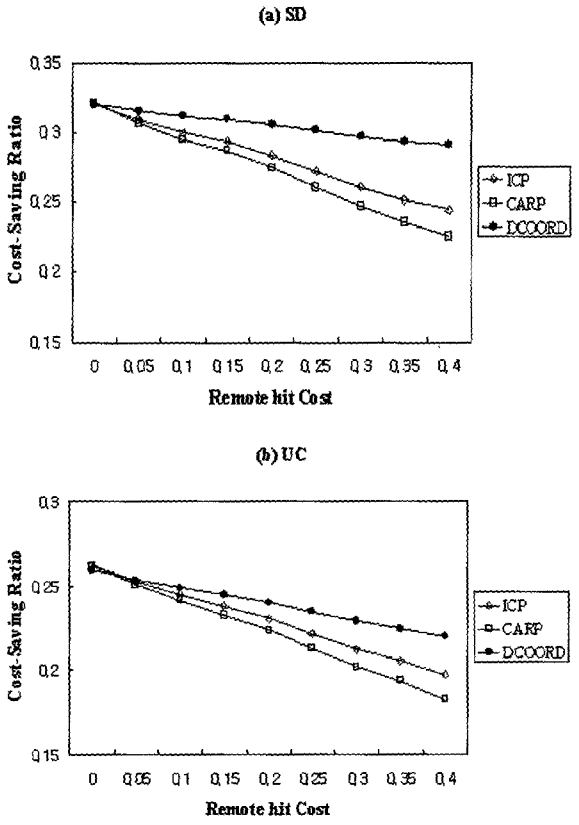
상대적인 CSR은 DCOORD가 SD로그나 UC로그 모두 ICP와 CARP 보다 더 좋은 값을 보여준다. SD로그일 경우 DCOORD가 ICP 보다 5.6%~ 7.8%, CARP 보다 7.9%~ 11.9% 정도 좋은 값을 보여준다.

ICP가 CARP보다 CSR이 높은 것은 CARP보다 캐쉬 수가 증가함에 따라 로컬 히트율이 훨씬 높기 때문에 오브젝트를 가져오는 데 대한 비용을 많이 절감할 수 있다. 그러나 캐쉬 수가 증가함에 따라 CARP의 경우, 많은 캐쉬들에 똑같은 오브젝트를 중복하여 캐쉬하지 않기 때문에 전체 히트율(로컬히트+원격히트)이 ICP보다 커진다. 그래서 ICP와의 CSR 차이가 많이 커지지는 않는다.

DCOORD는 ICP와 유사하게 클라이언트가 요구한 오브젝트를 로컬 클라이언트로 가져와 지역성에 의해 다음 번의 참조를 로컬 히트로 하여 주고, 원격 캐쉬에서는 워킹셋 영



(그림 6) 캐쉬 수에 따른 CSR



(그림 7) 원격히트의 비용에 따른 CSR

역에 있지 않은 경우 빨리 퇴출되도록 하므로 전체 히트율도 어느 정도 유지시켜 준다.

(그림 7)은 원격 히트의 비용을 변화시켰을 때의 CSR 값을 나타내었다. 원격 히트 비용은 로컬 캐쉬가 아닌 원격 캐쉬에서 원하는 오브젝트를 가져왔을 때의 비용이다. 이 값은 로컬 캐쉬에서 가져 왔을 때의 비용보다는 클 것이고 캐쉬 미스가 나서 원래의 웹 서버에서 가져올 경우에 비하면 많이 적을 것이다.

본 실험에서는 로컬 히트 비용인 0에서부터 0.4까지 변화시켜 보았다.

원격 캐쉬 히트 비용이 0이라는 것은 사실상 로컬 캐쉬 히트 비용과 같으므로 하나의 로컬 캐쉬가 있는 것과 동일하다. 따라서 DCOORD, ICP, CARP의 값이 거의 같고 가장 좋은 CSR 값을 가진다.

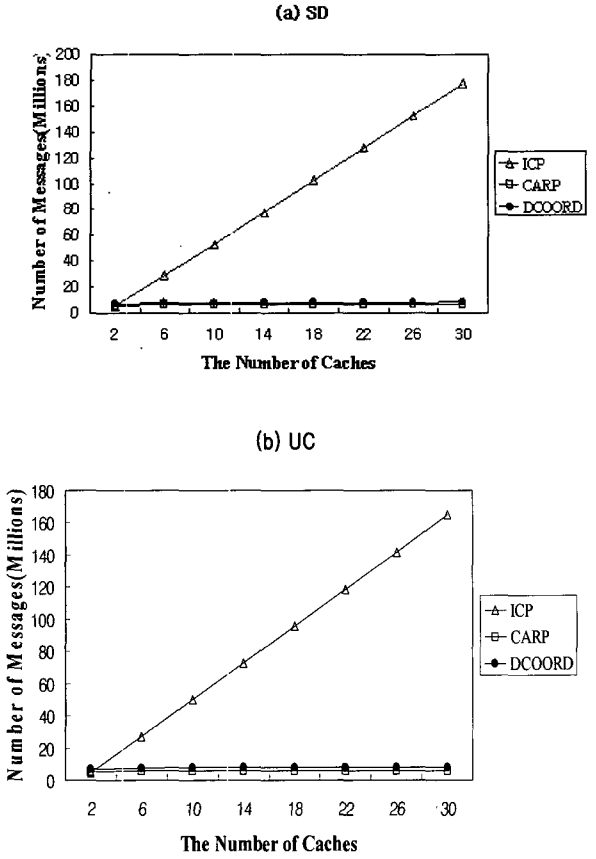
원격 캐쉬 히트 비용이 점점 커진다는 것은 원격의 캐쉬들이 네트웍 상의 여러 단계를 거치거나 거리가 많이 떨어져 있다거나 해당 원격 캐쉬의 네트웍 대역폭이 충분하지 않다는 것을 의미할 수 있다. 원격 히트 비용이 커지면 아무래도 전체의 CSR에서 원격 히트에 대한 부분만큼 CSR이 나빠지게 된다. CARP의 경우 원격 히트 비용의 변화에 보다 민감하게 CSR이 나빠짐을 알 수 있다. 세 가지 경우 중 가장 원격 히트율이 높기 때문이다. 상대적으로 DCOORD의 경우 가장 완만하게 CSR이 감소한다. 즉 원격 히트 비용의 증가에 가장 잘 적응한다고 할 수 있을 것이다.

원격 히트 비용이 커질수록 DCOORD와의 CSR의 차이가 점점 커지는데 SD로그일 경우 원격 히트 비용이 0.4일 경우 DCOORD는 ICP보다는 19.2% CARP보다는 29.1% 비용이 절감된다.

3.4.3 메시지 수

(그림 8)은 프락시서버의 수가 증가함에 따라 전체 오브젝트를 찾기 위한 캐쉬 사이의 메시지 수를 그렸다. CARP에 비해 DCOORD는 약간 더 메시지 수가 많다. 오브젝트의 위치 정보를 가지고 있는 캐쉬를 먼저 찾고 그 디렉토리가 지시하는 캐쉬로 한 번 더 찾아가야 하기 때문이다.

CARP나 DCOORD에 비해서 ICP는 캐쉬수가 증가함에 따라 메시지 수가 급격하게 늘어나게 된다. 캐쉬수가 증가함에 따라 O(n)의 복잡도(complexity)로 메시지가 늘어나게 된다. ICP에서는 N개의 캐시가 있을 때 오브젝트를 찾기 위한 ICP_QUERY 메시지 (N-1)개를 이웃 캐쉬로 보내고 이 메시지를 받은 캐쉬들은 ICP_HIT이나 ICP_MISS의 응답 메시지를 보내야 하기 때문이다. 즉 하나의 요구에 대해



(그림 8) 캐쉬 수 증가에 따른 캐쉬 간 메시지 수

2x(N-1)개의 메시지가 필요하다.

ICP의 경우 캐쉬간 전체 메시지 수는 다음과 같이 계산된다. 로컬 히트인 경우를 제외하고 원격히트나 캐쉬 미스일 경우에도 동일하게 다른 원격 캐쉬들에게 오브젝트가 있는지 질의하는 메시지가 필요하기 때문이다.

REQ x 2 x (Ncache-1) x (1-LHR)

REQ : 전체 요구(re

Ncache : 전체 캐쉬 수

LHR : 로컬 히트율(local hit ratio)

CARP의 경우 메시지 수는 하나의 요구에 대해 그 URL을 가지고 해쉬한 결과값을 가지는 캐쉬로 직접 가서 해당 오브젝트가 있는지 찾아보면 된다. 그러므로 2개의 메시지가 필요하다. 전체 메시지 수는 다음과 같이 계산된다.

REQ x 2 x (1-LHR)

DCOORD의 경우, 메시지 수는 먼저 그 오브젝트가 어디에 저장되어 있는지를 가리켜 주는 디렉토리를 찾고 그 다음 또 해당 캐쉬를 찾아서 질의를 해야 되므로 3개의 메시지가 필요하다. 전체 메시지 수는

REQ x 3 x (1-LHR)

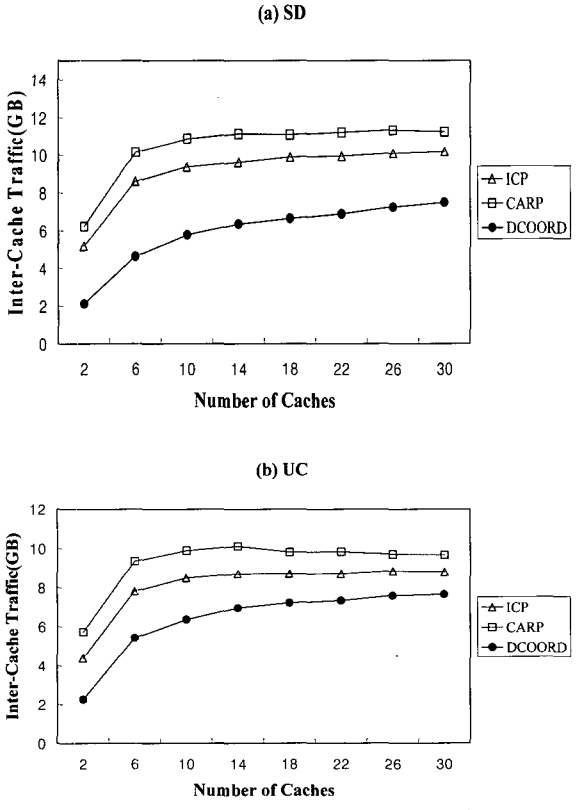
와 같이 계산할 수 있다. 위에서 보듯이 CARP가 메시지 수에서는 가장 적다.

그래프에서 보듯이 ICP는 캐쉬의 수에 따라 급격하게 증가하는 메시지 수 때문에 확장성 있는 프로토콜이 되지 못한다. 캐쉬의 수가 적을 때는 그럴 듯한 대안이 될 수 있지만 그렇지 못하면 네트웍에 상당한 부담을 주게 된다.

CARP와 DCOORD는 캐쉬 수가 증가함에 따라 그렇게 크게 증가하지는 않는다. 캐쉬 수가 많을 경우 적당한 방안이 될 수 있을 것이다. CARP의 경우 하나의 요구에 대해서 2개의 메시지가 필요하고 DCOORD의 경우 3개의 메시지가 필요하지만 전체적으로 50%의 메시지가 더 필요한 것은 아니다. 실험에서 SD 로그의 경우 약 22.7%~28.6% 의 메시지가 더 필요하였다. CARP의 경우 캐쉬의 수가 증가할수록 그 비율만큼 로컬 히트율이 작아지기 때문이다.

3.4.4 네트워크 트래픽

(그림 9)는 SD와 UC로그의 캐쉬 간의 트래픽을 측정한 것이다. 세 경우 모두 캐쉬 수가 증가함에 따라 트래픽이



(그림 9) 캐쉬 수의 증가에 따른 캐쉬 간 트래픽

증가하지만 캐쉬 수가 많이 증가해도 급격하게 증가하지는 않는다.

CARP의 경우 오브젝트가 로컬 캐쉬에 있기보다는 원격 캐쉬에 있을 확률이 훨씬 많기 때문에 이들 오브젝트를 클라이언트에 보내주기 위한 전체의 트래픽이 많아지게 된다. ICP와 DCOORD는 많은 오브젝트가 로컬 캐쉬에 있기 때문에 전체적인 트래픽이 상당히 감소된다. 그래프에서 보듯이 DCOORD, ICP, CARP의 순서로 트래픽이 적다는 것을 알 수 있다.

4. 결 론

본 논문에서는 여러 개의 캐쉬서버가 각자 자기에게 캐싱되어 있는 오브젝트들을 유기적으로 조정하여 전체의 캐싱 비용을 줄이는 DCOORD를 제안하였다.

클라이언트들의 요구에 대해 로컬히트와 원격히트는 그 비용이 현저히 차이가 난다. DCOORD는 각각의 캐쉬서버에 분산되어 있는 디렉토리에 오브젝트들의 위치 정보가 들어 있고, 이 디렉토리를 보고 해당 오브젝트가 있는 캐쉬서버로 찾아가게 된다. 오브젝트들은 워킹셋 영역이나 공유 영역에 저장된다. 클라이언트들이 최근에 요구한 오브젝트들을 로컬 캐쉬의 워킹셋 영역으로 가져온다. 이는 시간 지역성으로 인해 앞으로 곧 참조될 확률이 많기 때문에 최대한 오래 캐쉬에 보존되도록 하여, 원격히트보다 로컬 히트를 증가시켜 전체적인 비용을 줄이기 위해서다. 워킹셋 영역 위 부분에 위치하는 공유 영역은 현재 빈번히 참조하는 오브젝트들이 아니므로 다른 이웃 캐쉬에서 요구하면 오브젝트를 리스트의 헤드로 이동하여 빠른 시간 안에 퇴출되도록 한다. 이는 전체 공유 캐쉬 상에서 오브젝트의 불필요한 중복 캐싱을 줄이고 더 참조 가능성이 높은 오브젝트를 캐싱하여 히트 수를 증가시켜 준다.

성능 평가에서는 NLANR의 두 개의 실제 트레이스 로그를 사용하여 기존의 대표적인 공유 웹 캐싱 프로토콜인 ICP, CARP와 제안한 DCOORD를 비교하였다. 메트릭은 기존에 많이 사용되던 히트율 대신에 비용을 기반으로 한 CSR을 주로 사용하였다.

ICP의 경우 캐쉬의 수가 증가함에 따라 오브젝트를 찾기 위한 캐쉬서버 상호간의 메시지 수가 급격하게 증가하여 확장성 있는 기법이 되지 못한다. CARP의 경우 캐쉬의 수가 증가하거나 원격히트 비용이 커지면 상대적으로 큰 원격히트율 때문에 비용이 다소 늘어난다. DCOORD는 전체 히트율을 크게 감소시키지 않으면서 로컬 히트를 증가시켜 전체적인 비용을 줄인다. 상대적인 캐쉬 크기의 변화, 공유 캐쉬를 구성하는 캐쉬 수의 증가, 원격 히트 비용의 변화에 따른 다양한 실험을 통해 DCOORD가 기존의 ICP나 CARP에 비해 5%~30%의 비용 절감 효과가 있음을 보여 주었다.

참 고 문 헌

[1] R. Caceres, F.Douglis, A.Feldmann, G.Glass, and M.Rabi-novich. Web proxy caching: the devil is in the details. ACM Performance Evaluation Review, 26(3), pp. 11-15, December 1998.

[2] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems, Monterey, CA, December 1997.

[3] Fan, L., Cao, P., Almeida, J., and Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In Proceedings of the ACM SIGCOMM Conference, pp. 254-265, 1998.

[4] M. Cieslak, D. Foster, G. Tiwana, and R. Wilson. Web Cache Coordination Protocol v2.0. IETF Internet draft, 2000.

[5] Alex Rousskov and Duane Wessels. Cache digests. In Proceedings of the 3rd International WWW Caching Workshop, June 1998.

[6] G. Karakostas and D.N. Serpanos. Exploitation of Different Types of Locality for Web Caches. In Proceedings of the IEEE Symposium on Computers and Communications, Taormina-Giardini Naxos, Italy, July 2002.

[7] Wessels, D., and Claffy, K. RFC2186: Internet Cache Protocol(ICP), version 2. 1997.

[8] Anawat Chankhunthod, Peter Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. In Proceedings of the 1996 Usenix Technical Conference, San Diego, CA, January 1996.

[9] Evangelos P. Markatos and Catherine E. Chronaki. A top 10 approach for prefetching the Web. In Proceedings of the INET '98 conference, July 1998.

[10] B. Scott Michel, Konstantinos Nikoloudakis, Peter Reiher, and Lixia Zhang. URL forwarding and compression in adaptive Web caching. In Proceedings of IEEE INFOCOM'2000 conference, March 2000.

[11] Syam Gadde, Jeff Chase, and Michael Rabinovich. A taste of crispy Squid. In Proceedings of the Workshop on Internet Server Performance (WISP'98), June 1998.

신 용 현

Email : yshin@snut.ac.kr

1988년 서울대학교 계산통계학과(학사)

1991년 서울대학교 계산통계학과 전산과학 전공(이학석사)

2004년 서울대학교 전기컴퓨터공학부 (공학박사)

2005~현재 서울산업대학교 컴퓨터공학과 전임강사

관심분야 : 웹시스템, 시스템소프트웨어, 임베디드시스템 등