무선 인터넷 프록시 서버 클러스터 시스템에서 라운드 로빈을 이용한 해싱 기법

곽 후 근[†] · 정 규 식[††]

요 약

무선 인터넷 프록시 서버 클러스터 환경에서의 캐싱은 인터넷 트래픽, 웹 유저의 요청 및 응답 시간을 줄여주는 효과를 가진다. 이때, 캐시의 히트율(Hit ratio)을 증가시키는 한 가지 방법은 해쉬 함수를 이용하여 동일 요청 URL을 동일 캐시에 할당하는 방법이다. 해싱을 이용한 방법의 문제점은 해쉬의 특성으로 인해 클라이언트의 요청이 일부 캐시 서버로 집중되고 전체 시스템의 성능이 일부 캐시 서버에 종속된다는 점이다. 이에 본 논문에서는 해싱과 라운드 로빈 방식의 장점을 결합하여 클라이언트의 요청을 일부 캐시 서버가 아닌 전체 캐시 서버에 균일하게 분포시키는 개선된 부하 분산 방법을 제안한다. 기존 해싱 방법에서는 요청 URL에 대한 해쉬값이 계산되면 캐시 서버가 컴파일 시간에 정적으로 할당되는 반면, 제안된 방법에서는 라운드 로빈 방법을 사용하여 실행 시간에 동적으로 할당된다. 제안된 방법은 무선 인터넷 프록시 서버 클러스터 환경에서 구현되었고, 16대의 컴퓨터를 이용하여 실험을 수행하였다. 실험 결과는 기존 해싱 방법에 비해 클라이언트의 요청을 캐시 서버들 사이로 균일하게 분포시키고, 이에 따라 전체 무선 인터넷 프록시 서버의 성능이 52%에서 112%까지 향상됨을 확인하였다.

키워드 : 무선 인터넷, 프록시 서버, 클러스터, 부하 분산, 해싱, 라운드 로빈

A Hashing Scheme using Round Robin in a Wireless Internet Proxy Server Cluster System

Hukeun Kwak[†] · Kyusik Chung[††]

ABSTRACT

Caching in a Wireless Internet Proxy Server Cluster Environment has an effect that minimizes the time on the request and response of Internet traffic and Web user. As a way to increase the hit ratio of cache, we can use a hash function to make the same request URLs to be assigned to the same cache server. The disadvantage of the hashing scheme is that client requests cannot be well-distributed to all cache servers so that the performance of the whole system can depend on only a few busy servers. In this paper, we propose an improved load balancing scheme using hashing and Round Robin scheme that distributes client requests evenly to cache servers. In the existing hashing scheme, if a hashing value for a request URL is calculated, the server number is statically fixed at compile time while in the proposed scheme it is dynamically fixed at run time using round robin method. We implemented the proposed scheme in a Wireless Internet Proxy Server Cluster Environment and performed experiments using 16 PCs. Experimental results show the even distribution of client requests and the 52% to 112% performance improvement compared to the existing hashing method.

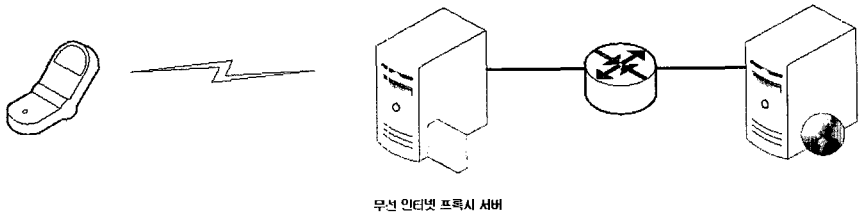
Key Words : Wireless internet, Proxy server, Cluster, Load Balancing, Hashing, Round Robin

1. 서 론

무선 인터넷에 대한 관심이 증가하는 가운데 핸드폰, PDA등의 무선 인터넷 단말기의 수요가 늘어나며 보편화 되어가고 있다. 그리고 무선 인터넷 서비스도 기존의 정보검색 위주의 간단한 서비스에서 전자 상거래나 멀티미디어 서비스 등의 복잡한 서비스로 사용자들의 욕구가 상승하고 있다. 이에 따라 무선 인터넷 대역폭의 효율적인 사용과 빠른 이용 시간을 위하여 무선 인터넷 프록시 서버의 필요성이 증대되고 있다. 무선 인터넷 프록시 서버는 무선 사용자를 유선 인터넷 서버에 연결 시켜주는 역할을 한다.

(그림 1)은 무선 인터넷에 사용되는 무선 인터넷 프록시 서버를 나타내고 있다. 무선 클라이언트가 무선 네트웍을 통해 무선 인터넷 프록시 서버로 요청을 보낸다. 무선 인터넷 프록시 서버가 클라이언트가 요청한 데이터를 가지고 있다면 바로 응답을 하지만, 없다면 유선 네트웍을 통해 웹 서버에게 클라이언트를 대신해서 요청을 보내게 된다. 최종적으로 무선 인터넷 프록시 서버는 웹 서버로부터 받은 요

※ 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 정 회 원 : 숭실대학교 전자공학과 대학원 postdoc
†† 정 회 원 : 숭실대학교 정보통신전자공학부 교수
논문접수 : 2006년 6월 2일, 심사완료 : 2006년 11월 1일



(그림 1) 무선 인터넷 프록시 서버

청을 저장 및 압축하고, 무선 네트워을 통해 클라이언트에게 응답을 한다.

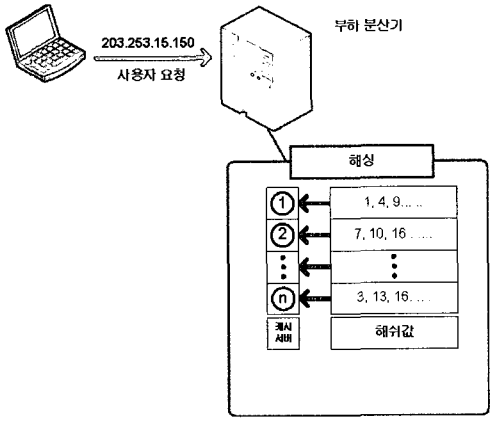
무선 인터넷 프록시 서버는 무선 인터넷의 낮은 대역폭 해결하기 위해 캐싱(Caching)[1]과 압축(Distillation)[2]을 사용하며, 대용량 트래픽에 대한 확장성(Scalability)이 고려되어야 한다. TranSend[3]는 대용량 트래픽에 대한 확장성을 고려하여 클러스터링으로 구현된 무선 인터넷 프록시 서버이다. 본 논문에서는 무선 인터넷 프록시 서버 클러스터인 TranSend를 확장성과 구조적인 관점에서 개선한 CD-A(CD & All-in-one)구조[4]를 사용하였다.

무선 인터넷 프록시 서버는 제한적인 대역폭을 효율적으로 사용하기 위해 캐싱(Caching)을 하게 된다. 캐시에 저장된 데이터의 히트율(Hit Ratio)을 높이기 위해 사용되는 방법 중에 하나는 해쉬 함수를 이용하여 동일 요청 URL을 동일 캐시에 할당하는 방법[5]이다. 동일 요청 URL에 대해 동일 캐시 서버를 할당하게 되면 캐시 서버 수에 무관하게 캐시 메모리 전체 크기를 일정하게 할 수 있다. 반면에 일반적으로 사용하는 RR 부하 분산 방식과 같이 동일 URL에 대해 서로 다른 캐시 서버를 할당하게 되면 각 캐시 서버가 모든 캐시 데이터를 가져야 하므로 캐시 메모리 전체 크기가 캐시 서버 수에 비례하여 증가한다.

무선 인터넷 프록시 서버를 클러스터로 구성함에 있어 동일 요청 URL을 동일 캐시에 할당 하는 것을 보장하는 것은 클러스터 간 중복되는 공간을 줄이고, 효율적인 캐싱을 통해 사용자가 요청한 데이터를 빠르게 얻을 수 있게 한다. 이를 위해 널리 사용되는 캐시 선택 방법은 해싱(Hashing)을 이용한 방법이다. 즉, 사용자 혹은 목적지 URL의 해싱값을 이용하여 캐시를 선택함으로써 동일 URL에 대한 동일 캐시를 선택되는 것을 보장한다.

(그림 2)는 무선 인터넷 프록시 서버 클러스터에서 동일 요청 URL을 동일 캐시에 할당하는 방법을 나타낸다. 사용자(203.253.15.150)가 www.daum.net으로 이미지(URL)을 요청하면, 부하 분산기[6]가 사용자의 Source IP(203.253.15.150)를 통해 해쉬값을 생성한다. 해쉬값을 통해 해당 해쉬값에 할당된 캐시 서버(1에서 n까지의 캐시 서버 중의 하나)로 이미지를 요청하고, 이후의 과정은 동일 해쉬값에 대해서는 동일 캐시 서버로 데이터를 요청한다.

본 논문에서는 기존의 해싱을 이용한 부하 분산 방식이 가지는 문제점을 분석하고 이를 해결할 새로운 해싱과 라운드 로빈 방식을 이용한 부하 분산 방식을 제안한다. 본 논문의 구성은 다음과 같다. 제 2장에서는 무선 인터넷 프록



(그림 2) 해싱을 이용한 부하 분산

시 서버 클러스터에서 기존의 해싱을 이용한 부하 분산 방식과 그 문제점을 소개한다. 3장에서는 기존의 해싱을 이용한 부하 분산 방식의 문제점을 해결하는 새로운 해싱과 라운드 로빈 방식을 이용한 부하 분산 방식을 설명하고, 4장에서는 실험 및 토론을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

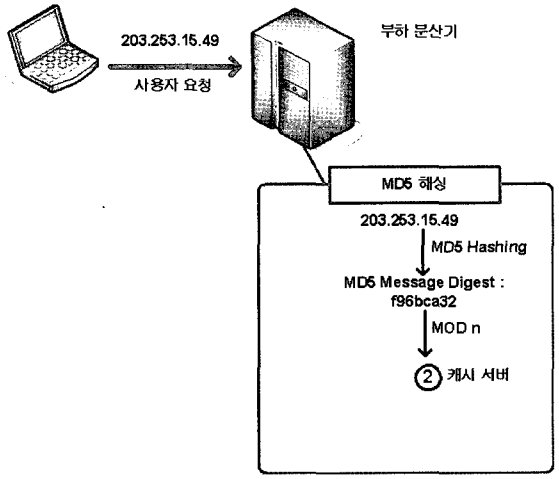
2. 기존 연구

캐시 선택 시에 사용할 수 있는 해싱 방법 중에는 MD5 해싱(Message Digest ver.5 Hashing)[7] 방법과 Modified Consistent Hashing[8] 등이 있다.

2.1 MD5 Hashing[7]

MD5(Message Digest ver.5)는 1992년에 MIT의 Ronald Rivest가 개발하였다. 입력되는 메시지는 임의의 길이를 가질 수 있고, 이는 2^{64} bits 이하여야 한다. 메시지는 512 비트 단위 블록(Block)들로 분할되며, 이것은 각각 16개의 32 비트 서브(Sub) 블록으로 분할된다. 이 메시지 블록들은 512 비트의 배수 길이가 되도록 0으로 패딩(Padding)되고, 마지막 블록에서 64 비트 구간은 길이 필드로 사용된다. 메시지 축약(Message Digest)의 결과는 128 비트(=32 비트 X 4 서브 블록)이다.

MD5 해싱은 사용자 요청 URL에 대해 고정 길이의 해시값이 나오고, 이를 캐시의 수에 매핑하는 방식으로 동작한다. 요청이 들어올 때 마다 Message Digest를 생성, 클러스터에 할당하기 때문에 버킷을 위한 별도의 메모리 공간 소



(그림 3) MD5 Hashing

요가 없고, 캐시의 추가 및 삭제 시에 영향을 받지 않는 유동적인 구조를 가진다. 그러나, 매 요청마다 해시값을 계산하여 클러스터로 분해하기 때문에 사용자의 요청수 증가에 따른 각 요청의 Message Digest 계산 시간이 커진다는 단점을 가진다.

(그림 3)은 무선 인터넷 프록시 서버인 TranSend에서 사용하는 MD5 해싱을 통한 캐시 클러스터 선택을 나타낸다. 부하 분산기가 사용자의 요청 URL(203.253.15.49)을 통해 Message Digest(f96bca32)를 생성한다. 생성된 Message Digest(f96bca32)를 캐시 개수(N)로 나머지 연산(Mod)을 수행 한다(f96bca32 Mod N = 2). 그리고 나머지 연산을 통해 나온 값(2)과 동일한 캐시(2번 캐시)로 요청을 할당한다.

2.2 Modified Consistent Hashing[8] (MOD-CH)

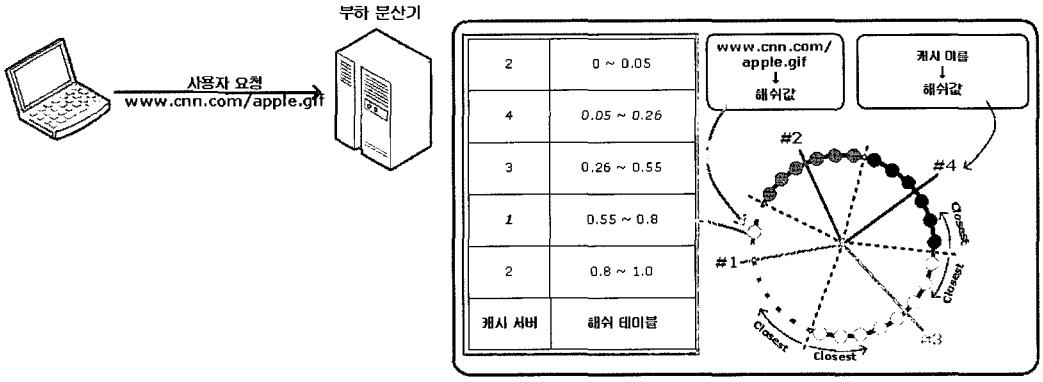
Modified Consistent Hashing은 기존의 Consistent Hashing 방법[9]의 단점인 클러스터 내에서 요청을 균일하게 할당하지 못하는 것을 보완하고자 제안된 방식이다. 이는 기존의 Consistent Hashing에서 요청 URL에 대한 캐시 서버의 할당과 관련하여 해쉬 포인트가 Unit Circle에서 시계방향으로 가까운 서버로 할당되는 방식을 수정하였다. 즉, 해쉬 포인트가 방향과 상관없이 가까운 캐시 서버에 할당되게 하여 캐시 서버 간에 요청이 균일하게 분포되도록 하였다. 그러나 캐시 서버의 할당과 관련하여 여전히 MD5 해쉬를 사용함으로 완전한 균일 분포를 이루지 못하는 단점을 가진다. 이로 인해 서버의 확장에 따른 성능의 선형적인 증가를 보장하지 못하며, 순간적인 요청 집중 현상(Hot-Spot)을 처리할 수 없는 단점을 가진다.

(그림 4)는 Modified Consistent Hashing의 전체적인 구조를 나타낸다. DNS(Load Balancer: 부하 분산기)는 캐시 이름에 대해 해쉬 함수(MD5 해쉬)를 사용하여 (0,1]의 영역에 할당한다. 예를 들어, 해쉬 값이 0과 0.05사이의 값이라면 캐시 서버 2를 할당한다. 사용자(Client)가 컨텐츠(http://www.cnn.com/apple.gif)에 대해서 요청을 보내면, 요청 URL에 대한 해쉬 값을 Unit Circle안의 한 지점에 할당(0.6)하고 방향과 상관없이 가장 가까운 캐시 서버(1번 캐시)에 요청을 할당한다. 해당 캐시 서버는 자신의 정보를 사용자에게 보내어 사용자가 웹 브라우저를 통해서 해당 캐시 서버로 접속하여 서비스 받도록 한다. 캐시 서버를 추가할 경우 추가된 캐시 서버는 Unit Circle 내부의 한 지점에 할당되며, 영역 내부의 서버만 분할될 뿐 나머지 캐시 서버들에게 영향을 주지 않는다. 캐시 서버를 삭제할 경우에도 추가의 경우와 마찬 가지로 동작된다.

2.3 접근 방식

본 절에서는 캐시의 추가 및 삭제 시에 영향을 받지 않는 MD5 해싱을 이용한 부하 분산의 문제점을 정리하고, 3장에서는 이를 해결하는 새로운 부하 분산 알고리즘을 제안한다. MD5 해싱(일반 해싱)의 단점은 해쉬의 특성상 전체 사용자의 요청이 일부 캐시 서버로 집중된다는 점이다. 이러한 집중은 무선 인터넷 프록시 서버 클러스터의 전체 성능을 요청이 집중되는 일부 캐시 서버에 종속시킨다(즉, 전체적인 서버 성능을 떨어뜨린다).

본 논문에서는 해쉬를 사용하되 전체적인 사용자의 요청을 캐시 서버 사이로 균일하게 분포시키는 방법을 제안한다. 제안된 방법은 해싱과 라운드 로빈 방식을 이용하는 방식으로 사용자의 요청이 일부 캐시 서버로 균일하게 분포되지 않는 것을 방지한다.



(그림 4) Modified Consistent Hashing

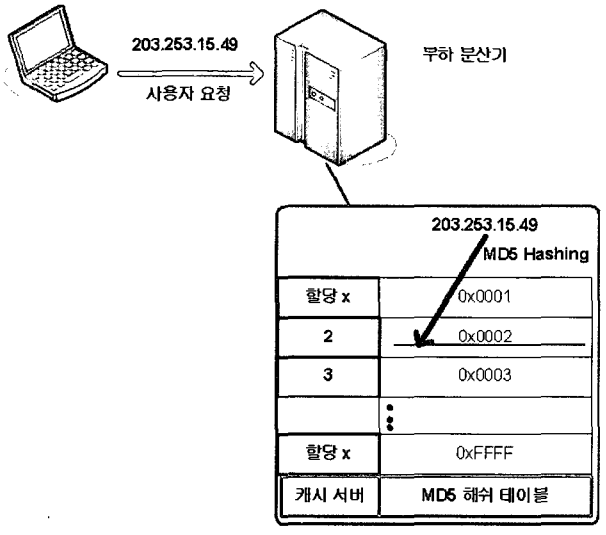
3. 라운드 로빈을 이용한 해싱 기법 (RR-Hashing)

제안된 방식은 동일 요청 URL을 동일 캐시에 할당하도록 하기 위해 해쉬를 이용하되 이들의 특성으로 인해 사용자의 전체 요청이 일부 캐시 서버로 집중되는 것을 막자는 것이다. 기존 방식이 사용자 URL의 해쉬값을 사용된 캐시 서버 수로 나누어 사용자 요청을 할당했다면, 제안된 방식은 사용자 URL의 해쉬값을 라운드 로빈 방식으로 관리하여 사용자의 요청을 캐시 서버로 균일하게 할당하였다.

(그림 5)는 MD5 및 라운드 로빈 방식을 적용하여 캐시를 선택하는 과정을 나타낸다. 부하 분산기(Load Balancer)가 사용자의 요청 URL(203.253.15.49)에 대해 MD5 해싱을 적용하여 Message Digest(해쉬값: 0x0002)를 생성한다. 해쉬 테이블을 검색하여 Message Digest(해쉬값: 0x0002)에 해당하는 곳에 캐시 서버가 할당되어 있는지 확인한다. 캐시 서버가 할당되어 있다면(그림에서는 캐시 서버 2), 그 캐시 서버(2)로 사용자 요청을 보낸다. 만약, 캐시 서버가 할당되어 있지 않다면 새로운 캐시 서버를 할당하고, 사용자의 요청을 새로운 캐시 서버로 보낸다.

캐시 서버가 N개라 가정하고 라운드 로빈 방식을 적용하여 새로운 캐시 서버를 할당하는 방법은 다음과 같다. 처음 캐시 서버 할당을 요청할 때는 캐시 서버 1을 할당한다. 두 번째 캐시 서버 할당을 요청할 때는 캐시 서버 2를 할당한다. N+1번째 캐시 서버 할당을 요청할 때는 캐시 서버 1을 다시 할당한다. 이와 같은 방식으로 해쉬값에 따라 순서대로 해쉬값을 할당하고 이들을 해쉬 테이블을 할당한다면 사용자의 요청을 보다 균일하게 캐시 서버에 할당할 수 있게 된다.

<표 1>은 기존 방법과 제안된 방법을 확장성 측면에서 비교한 것이다. 확장성(Scalability)은 성능(Performance), 저장 공간(Storage) 및 요청 집중(Hot-Spot)으로 분류하였고, 이를 정성적인 방법으로 비교하였다. 정량적인 비교는 4장의 실험 결과에 나타내었다. 일반적인 부하 분산 알고리즘



(그림 5) 라운드 로빈을 이용한 해싱 기법

<표 1> 기존 방법 vs. 제안된 방법 (정성적 비교)

확장성	RR	MD5-Hashing	MOD-CH	RR-Hashing
성능	O	X	X	O
저장 공간	X	O	O	O
요청 집중	O	X	X	O

의 경우(Round-Robin) 성능 및 요청 집중에 대한 확장성을 가지나 저장 공간에 대한 확장을 가지지 않는다. 저장 공간에 대한 확장성을 가지지 않는 이유는 일반적인 부하 분산 방식이 동일 요청 URL을 동일 캐시에 할당하지 않기 때문이다. 즉, 일반적인 부하 분산 알고리즘에서는 모든 캐시 서버가 중복된 데이터를 가지고 있음으로 전체적인 캐시의 합이 커진다. 이에 비해, MD5 Hasing 방법과 MOD-CH (Modified Consistent Hashing)은 동일 요청 URL을 동일 캐시에 할당함으로써 전체 캐시의 합을 일정하게 유지할 수 있기 때문에 저장 공간에 대한 확장성을 가진다. 그러나 해쉬의 특성으로 인해 일부 캐시 서버로 요청이 집중됨으로써 성능에 대한 확장성이 없음을 알 수 있다. 또한, 해쉬의 특성상 동일 URL에는 동일 캐시 서버가 할당됨으로 요청 집중이 발생했을 때(동일 URL이 집중적으로 요청되었을 때) 이를 처리할 수 없고, 이로 인해 성능이 크게 떨어진다.

제안된 방법은 동일 요청 URL을 동일 캐시에 할당하기 위해 해쉬를 사용하였고(저장 공간의 확장성 보장), 일부 캐시 서버로 요청이 균일하게 분포되지 않는 것을 막기 위해 라운드 로빈 방식을 사용하였다(성능의 확장성 보장). 라운드 로빈 방식으로 인해 캐시 서버로 요청이 균일하게 분포하면 성능 측면에서 확장성을 가지게 된다. 요청 집중이 발생한 URL에 대해서는 라운드 로빈 방식으로 처리를 하도록 하여 요청 집중이 발생한 특정 URL에 대해서는 모든 캐시 서버들이 이를 처리하도록 하였다. 요청 집중이 사라지면(특정 URL에 대한 요청이 줄어들면), 특정 URL을 처리하는 원래의 캐시 서버만이 해당 URL을 다시 처리하게 함으로써 요청 집중에 대한 확장성을 가지도록 하였다.

4. 실험 및 토론

4.1 실험 환경

<표 2>는 실험에 사용된 하드웨어와 소프트웨어를 나타낸다. 프록시 서버는 PC 16대로 구성되어 있고, 부하 분산을 위하여 LVS[6]라는 부하 분산기를 사용하였다. Apache Bench와 Web Stone이라는 프로그램을 클라이언트에서 수행하여, 프록시 서버에 영상(이미지) 혹은 텍스트 파일을 요청하는 방식으로 실험하였다. 표에서 클라이언트와 LVS가 서버보다 하드웨어 성능이 좋은 이유는 확장성 실험을 할 때 클라이언트와 LVS에서는 병목이 발생하지 않는 상황에서 프록시 서버 내 호스트들 사이의 확장성(성능, 저장 공간, 요청 집중)을 확인하고자 했기 때문이다.

본 논문에서 논의된 4개의 해싱 방법(RR, MD5-Hashing,

〈표 2〉 실험 B에 사용된 하드웨어 및 소프트웨어

		하드웨어		소프트웨어	개수
		CPU (MHz)	RAM (MB)		
클라이언트 / 관리자		P-4 2260	256	Apache Bench[10], Webstone[12]	10 / 1
LVS		P-4 2400	512	NAT/DR[6]	1
서버	캐시	P-2 400	256	Squid[11]	16
	압축기			JPEG-6b	
웹 서버		P-4 2260	256	Apache	1

MOD-CH, 및 RR-Hashing(제안된 방법))들이 구현되었고, 세 가지 요청 파일을 대상으로 실험을 수행하였다. 첫 번째는 요청 하는 파일들이 가중치를 가지지 않는 가장 이상적인 경우로써, 모든 파일들이 동일한 가중치를 가지고 요청된다. 두 번째는 요청 하는 파일들이 가중치를 가지는 가장 현실적인 경우로써, 각 파일들이 각각의 가중치를 가진다. 이때, 실제 사용자의 요청 파일 분포를 모델링 하여 적은 크기의 파일에는 높은 가중치를 큰 크기의 파일에는 낮은 가중치를 주었다. 마지막으로 세 번째는 요청 집중이 발생하는 최악의 경우로써, 모든 파일들이 동일한 가중치를 가지고 요청되는 가운데 특정 파일에 대해서만 요청을 집중하도록 하였다.

4.2 실험 결과

4.2.1 가중치를 가지지 않는 경우 (이상적인 경우)

<표 3>과 (그림 6)은 300 Bytes에서 100 Kbytes의 크기를 가지는 이미지를 가중치 없이 클라이언트가 요청을 보냈을 때, 무선 인터넷 프록시 서버가 초당 처리하는 요청의 개수를 나타낸다. 표에서 Variation은 1 Kbytes에서 10 Kbytes의 크기의 이미지들을 랜덤하게 요청한 경우이다. 그림에서 보면 MD5-Hashing의 경우 서버의 개수가 증가해도 초당 처리수가 비례적으로 증가하지 않는 반면에, RR과 제안된 방법(RR-Hashing)은 서버의 개수가 증가함에 따라 초당 요청수가 비례적으로 증가하는 것을 볼 수 있다. 이는

〈표 3〉 호스트 개수에 따른 초당 요청수

(a) RR 부하 분산

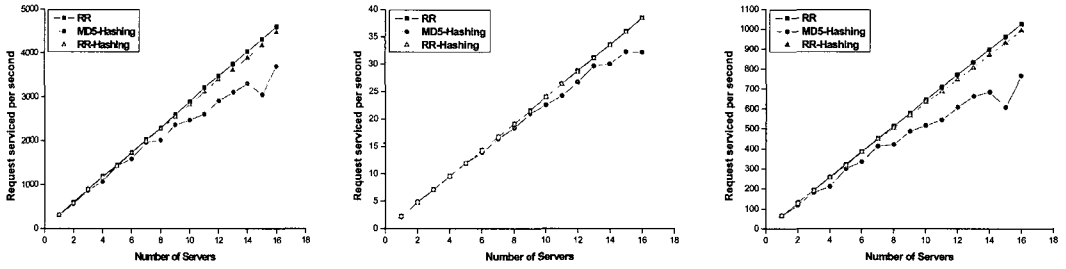
호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	316	596	895	1188	1446	1732	2025	2285	2598	2890	3212	3474	3750	4034	4315	4603
1 Kbytes	233	444	664	882	1090	1282	1516	1732	1921	2137	2353	2577	2799	3026	3244	3458
10 Kbytes	36	71	110	143	179	215	251	287	322	358	393	429	465	501	539	574
100 Kbytes	2.35	4.82	7.16	9.57	11.94	14.29	16.76	19.15	21.58	23.97	26.37	28.78	31.16	33.59	35.99	38.46
Variation	66	129	193	258	323	386	451	513	578	646	709	772	834	898	961	1026

(b) MD5 Hashing 부하 분산

호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	316	564	873	1069	1430	1589	1964	2013	2362	2470	2607	2908	3103	3305	3046	3694
1 Kbytes	233	415	636	771	1056	1160	1455	1489	1735	1809	1931	2148	2277	2390	2195	2703
10 Kbytes	36	67	104	123	172	191	231	232	277	297	311	342	373	388	345	434
100 Kbytes	2.15	4.92	7.15	9.54	11.93	13.85	16.38	18.32	20.99	22.49	24.20	26.71	29.68	30.03	32.20	32.16
Variation	65	117	181	211	299	336	414	421	486	517	544	608	662	684	608	766

(c) 제안된 방식

호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	315	578	893	1168	1415	1730	2004	2270	2536	2821	3113	3399	3615	3890	4178	4474
1 Kbytes	230	437	654	869	1062	1258	1502	1710	1919	2136	2301	2531	2707	2949	3137	3326
10 Kbytes	36	72	109	146	182	215	252	288	315	351	390	418	455	485	522	555
100 Kbytes	2.24	4.74	7.11	9.52	11.93	14.31	16.74	19.16	21.52	23.96	26.38	28.61	31.11	33.46	35.89	38.38
Variation	65	128	193	255	317	383	449	503	565	634	687	748	807	870	931	993



(a) 300 Bytes (b) 100 KBytes (c) Variation

(그림 6) 호스트 개수에 따른 초당 요청수

MD5-Hashing이 해쉬의 특성으로 인해 클라이언트의 요청이 캐시 서버들 사이로 균일하게 분포하지 않고, 이로 인해 전체적인 캐시 서버의 성능이 일부 캐시 서버에 종속된다(저장 공간상의 확장성을 가지지만, 성능상의 확장성을 가지지 않음). 이에 반해 제안된 방법은 해싱에 라운드 로빈 방법을 적용하여 클라이언트의 요청을 캐시 서버들 사이로 균일하게 분포시킴으로써 기존 방법에 비해 전체적인 캐시 서버의 성능이 크게 향상되었음을 볼 수 있다(성능과 저장 공간상의 확장성을 가짐). RR은 비교의 목적으로 실험된 것으로, 모든 캐시 서버가 동일 클라이언트 요청 데이터를 가짐으로써 가장 좋은 성능을 나타냄을 볼 수 있다(성능상의 확장성을 가지지만, 저장 공간상의 확장성을 가지지 않음).

4.2.2 가중치를 가지는 경우 (실제의 경우)

<표 4>와 (그림 7)은 실제 웹 Traces(Surge-100[13], UNC-2001[14], Berkeley-1998[15])의 파일 크기 분포 및 가중치를 고려하여 클라이언트가 요청을 보냈을 때, 무선 인터넷 프록시 서버가 초당 처리하는 요청의 개수를 나타낸다. 그림에서 보면 기존 방식(MD5-Hashing, MOD-CH)의 경우 서버의 개수가 증가해도 초당 처리수가 비례적으로 증가하지 않는 반면에, RR과 제안된 방법(RR-Hashing)은 서버의 개수가 증가함에 따라 초당 요청수가 비례적으로 증가하는 것을 볼 수 있다. 이는 기존 방법이 해쉬의 특성으로 인해 클라이언트의 요청이 캐시 서버들 사이로 균일하게 분포하지 않고, 이로 인해 전체적인 캐시 서버의 성능이 일부 캐시 서버에 종속되었음을 의미한다(저장 공간상의 확장성을 가지지만, 성능상의 확장성을 가지지 않음). 이에 반해 제안된 방법은 해싱에 라운드 로빈 방법을 적용하여 클라이언트의 요청을 캐시 서버들 사이로 균일하게 분포시킴으로써 기존 방법에 비해 전체적인 캐시 서버의 성능이 크게 향상되었음을 볼 수 있다(성능과 저장 공간상의 확장성을 가짐). RR은 비교의 목적으로 실험된 것으로, 모든 캐시 서버가 동일 클라이언트 요청 데이터를 가짐으로써 가장 좋은 성능을 나타냄을 볼 수 있다(성능상의 확장성을 가지지만, 저장 공간상의 확장성을 가지지 않음).

〈표 4〉 호스트 개수에 따른 초당 요청수

(a) Surge-100 요청

캐시 서버 수	1	2	4	8	16
RR	270.50	344.52	617.96	1170.00	2147.02
MD5-Hashing	268.23	305.69	469.61	678.40	814.61
MOD-CH	267.77	306.35	484.00	693.89	802.17
RR-Hashing	266.25	361.89	567.93	912.46	1722.06

(b) UNC-2001

캐시 서버 수	1	2	4	8	16
RR	246.72	433.34	787.28	1537.47	3058.60
MD5-Hashing	246.23	421.00	654.05	934.20	1312.74
MOD-CH	246.65	400.44	556.84	827.77	1082.38
RR-Hashing	245.06	359.18	614.60	986.38	1827.58

(c) Berkeley-1998

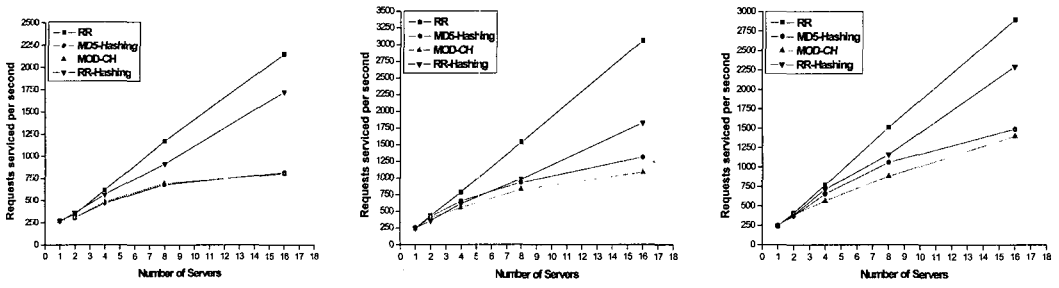
캐시 서버 수	1	2	4	8	16
RR	237.87	409.74	771.98	1512.32	2899.10
MD5-Hashing	245.48	372.53	655.09	1062.59	1489.66
MOD-CH	245.07	387.31	567.13	883.82	1400.37
RR-Hashing	245.57	383.21	716.02	1163.73	2294.99

4.2.3 요청 몰림(Hot-Spot)의 경우 (최악의 경우)

<표 5>와 (그림 8)은 MD5 Hashing과 제안된 방법에서 요청 몰림(Hot-Spot)이 발생한 경우의 실험 결과이다. 요청 몰림은 특정 파일을 지속적으로 요청하는 경우로써, 표와 그림에서 보면 MD5 Hashing이 요청이 하나의 파일에 집중되는 경우 성능상의 확장성이 없음을 볼 수 있다. 이는 해쉬의 특성으로 특정 파일은 특정 서버만이 처리할 수 있고, 프록시 서버의 전체 성능은 요청이 집중되는 특정 파일을 처리하는 서버에 종속되게 된다. 이에 반해 제안된 방법은 요청 집중이 발생해도 성능상의 확장성을 보장함을 알 수 있다. 이는 제안된 방법은 요청 집중이 발생한 특정 파일에 한해서 모든 서버들이 동시에 이 파일을 처리할 수 있는 구조로 되어 있기 때문이다. 시간이 지나 더 이상 특정 파일에 대한 요청 집중이 발생하지 않으면 특정 파일은 다시 특정 서버가 처리하게 된다.

4.2.4 기존 방법 vs. 제안된 방법

<표 6>은 가중치를 가지지 않는 경우, 가중치를 가지는 경우 및 요청 집중(Hot-Spot)의 경우에 기존 방법과 제안된 방법의 성능 비교를 나타낸다. 가중치를 가지지 않는 경우



(a) Surge-100 (b) UNC-2001 (c) Berkeley-1998

(그림 7) 호스트 개수에 따른 초당 요청수

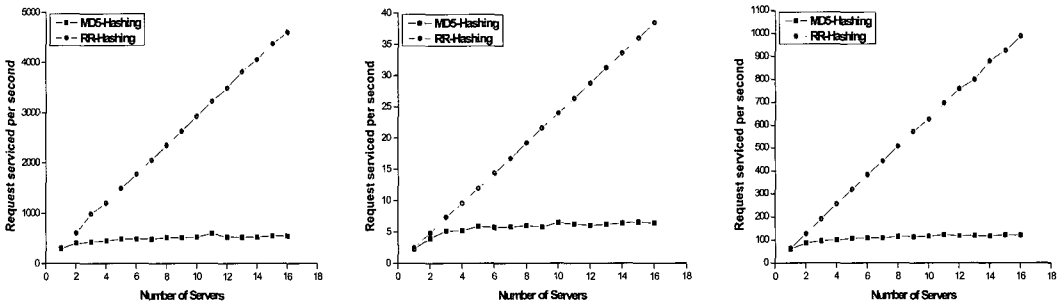
〈표 5〉 호스트 개수에 따른 초당 요청수

(a) MD5 Hashing 부하 분산

호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	299	405	432	451	485	486	486	515	511	521	591	523	520	520	544	538
1 Kbytes	232	316	326	341	365	366	366	386	385	394	433	426	425	420	421	415
10 Kbytes	35	48	54	57	62	62	62	65	65	66	70	67	68	67	70	68
100 Kbytes	2.29	3.96	5.15	5.18	5.88	5.71	5.81	6.00	5.75	6.48	6.27	6.04	6.19	6.40	6.57	6.38
Variation	62	87	96	100	107	109	108	114	114	117	122	118	118	118	121	120

(b) 제안된 방식

호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	317	599	983	1191	1487	1766	2051	2340	2626	2923	3220	3491	3815	4055	4371	4600
1 Kbytes	232	441	662	878	1093	1304	1510	1748	1958	2136	2401	2606	2793	3038	3254	3456
10 Kbytes	36	71	109	143	181	216	253	285	325	360	397	431	467	501	546	567
100 Kbytes	2.47	4.83	7.36	9.53	11.90	14.34	16.70	19.15	21.55	23.96	26.33	28.75	31.18	33.57	36.00	38.42
Variation	65	127	191	256	319	383	443	506	572	626	696	758	798	881	925	987



(a) 300 Bytes (b) 100 Kbytes (c) Variation

(그림 8) 호스트 개수에 따른 초당 요청수

에서는 기존 방법에 비해 19.34%에서 29.58%의 성능 향상을 가진다. 가중치를 가지는 경우에는 기존 방법에 비해 52.16%에서 113.02%의 성능 향상을 가진다. 마지막으로 요청 집중의 경우에는 기존 방법에 비해 502.19%에서 754.90%의 성능 향상을 가진다. 가중치를 가지지 않는 경우는 이상적인 경우로써, 모든 요청 파일들을 가중치 없이 동일한 비율로 요청함으로써 가장 낮은 성능 향상률을 보인다. 가중치를 가지는 경우는 실제적인 경우로써, 각 요청 파일들을 가중치 비율로 요청함으로써 중간 정도의 성능 향상을 보인다. 마지막으로 요청 집중의 경우는 최악의 경우로써, 특정 파일을 집중적으로 요청함으로써 가장 높은 성능 향상률을 보인다.

〈표 6〉 성능 비교 (%)

(a) 가중치를 가지지 않는 경우

16대 기준	300 Bytes	1 Kbytes	10 Kbytes	100 Kbytes	Variation
MD5-Hashing	3694	2703	434	32.16	766
RR-Hashing	4474	3326	555	38.38	993
평균 성능 향상률	21.10%	23.06%	27.81%	19.34%	29.58%

(b) 가중치를 가지는 경우

16대 기준	Surge-100	UNC-2001	Berkeley-1998
MD5-Hashing	814.61	1312.74	1489.66
MOD-CH	802.17	1082.38	1400.37
RR-Hashing	1722.06	1827.58	2294.99
평균 성능 향상률	113.02%	52.61%	58.82%

(c) 요청 몰림(Hot-Spot)의 경우

16대 기준	300 Bytes	1 Kbytes	10 Kbytes	100 Kbytes	Variation
MD5-Hashing	538	415	68	6.38	120
RR-Hashing	4600	3456	567	38.42	987
평균 성능 향상률	754.90%	732.57%	728.58%	502.19%	719.71%

4.3 토론

제안된 해싱과 라운드 로빈 방식을 이용한 부하 분산 방법이 기존 해싱을 이용한 부하 분산 방법에 비해 성능이 좋아진 이유는 사용자의 요청을 모든 서버에 균일하게 분포시킨 것으로 설명할 수 있다. 기존 방식은 사용자의 목적지 URL을 이용하여 해싱을 적용하여 부하가 일부 서버로 균일하게 분포되지 않는 반면(성능과 요청 집중 상에서 확장성을 가지지 않음), 제안된 방식은 사용자의 목적지 URL을 이용하여 해싱을 적용하고 이를 라운드 로빈 방식으로 관리함으로써 전체 사용자의 요청을 캐시 서버로 사이로 균일하게 분산하였다(성능과 요청 집중 상에서 확장성을 가짐).

제안된 방식의 단점은 해쉬 테이블을 관리하기 위해 메모리가 필요하다는 점이다. 본 논문에서 사용된 메모리는 65536 bytes이다. 이는 로컬에서 할당할 수 있는 메모리의 한계로 인해 MD5 해쉬로 계산할 수 있는 해쉬값(128 bit)를 모두 사용하지 않고 이중 16bit(0xFFFF)만을 할당한 결과이다. 16 bit만을 이용하게 되면 서로 다른 두개의 URL에 대해 동일 해쉬값이 나오는 경우가 발생하게 되고 이는 사용자의 요청이 캐시 서버 사이로 Round-Robin처럼 균일하게 할당되지 않는 것을 의미한다.

5. 결 론

제안된 방법은 클라이언트가 요청한 URL의 해쉬값을 이용하여 캐시 서버를 선택할 때, 라운드 로빈(Round-Robin)을 이용하는 것이다(저장 공간상의 확장성). 이러한 방법을 이용하게 되면 클라이언트가 요청한 URL의 해쉬값의 특성에 영향을 받지 않고 요청된 URL을 캐시 서버에 균일하게 할당할 수 있다. 사용자 요청 URL을 캐시 서버에 균일하게 할당하게 되면 시스템이 전체적으로 확장성을 가지게 되며, 이는 시스템이 높은 성능을 유지하도록 만들어준다(성능 상의 확장성). 해싱의 특성상 하나의 URL로 요청이 몰리는 경우(Hot-Spot)가 발생하면, Hot-Spot이 발생한 해당 URL 요청에 대해서만 Round-Robin 방법을 사용함으로써 Hot-Spot 문제를 해결할 수 있다(요청 집중상의 확장성).

제안된 방법을 테스트하기 위해 16대의 서버로 이루어진 클러스터, LVS 부하 분산기, 10대의 클라이언트들과 하나의 웹 서버로 구성된 실험 환경을 사용하였다. 클라이언트들은 인터넷을 통해 얻은 웹 Traces(Berkeley-1998, UNC-2001, Surge-100, Webstone 및 다양한 크기의 이미지와 HTML 페이지들)에 기반하여 요청들을 생성하였다. 실험 결과는 제안된 방법이 기존 방법들에 비해 성능 및 요청 집중 상의 확장성 측면에서 우수함을 알 수 있다.

기존 방법의 경우 동일 URL 요청을 동일 캐시 서버에 할당하기 위해 해싱을 사용할 경우 낮은 확장성과 성능을 가지게 된다. 이는 요청 URL의 해쉬값을 이용하여 캐시 서버에 할당 할 때 캐시 서버들로 요청 URL이 균일하게 분포되지 않는 해쉬의 특성에 기인한다(성능과 요청 집중 상의 확장성을 가지지 않음). 제안된 방법은 요청 URL을 캐시 서버에 할당할 때 해쉬의 특성에 영향을 받지 않게 함으로써 요청 URL을 캐시 서버에 균일하게 할당하였다. 이러한 균일한 할당은 높은 확장성과 성능을 보장한다(성능과 요청 집중 상의 확장성을 가짐).

향후 연구 방향은 해쉬 기반의 동적 서버 부하 분산을 고려해 볼 수 있다. 본 논문에서 제안된 방법은 서버의 상태를 고려하지 않고 1/N의 형식으로 균일하게 캐시 서버로 요청을 할당하는 방식이다. 만약, 서버의 부하 상태가 서로 틀리다면 1/N의 균일한 분포는 전체적인 확장성을 보장하지 못한다. 이를 해결하기 위해 요청 URL을 캐시 서버에 할당할 때, 캐시 서버의 동적인 정보(CPU 이용률, 메모리 이용률 등)를 고려하는 알고리즘이 필요하다.

참 고 문 헌

[1] A. Feldmann, R. Caceres, F. Douglis, G. Glass and M. Rabinovich, "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments," In Proceedings of the INFOCOM Conference, 1999.

[2] A. Savant, N. Memon and T. Suel, "On the Scalability of an Image Transcoding Proxy Server," In IEEE International Conference on Image Processing, Barcelona, Spain, 2003.

[3] A. Fox, "A Framework For Separating Server Scalability and Availability From Internet Application Functionality," Ph. D. dissertation, U. C. Berkeley, 1998.

[4] 곽후근, 정규식, "무선 인터넷 프록시 서버 클러스터 성능 개선", 한국정보과학회:정보통신, Vol.32, No.3, pp.415-426, 2005.

[5] F. Baboescu, "Proxy Caching with Hash Functions," Technical Report CS2001-0674, 2001.

[6] LVS(Linux Vitual Server), http://www.linuxvirtualserver.org

[7] D. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, 1992.

[8] David Karger and al. "Web Caching with consistent hashing," In WWW8 conference, 1999.

[9] F. Baboescu, "Proxy Caching with Hash Functions," Technical Report CS2001-0674, 2001.

[10] AB(Apache Bench), http://www.apache.org.

[11] Squid Web Proxy Cache, http://www.squid-cache.org.

[12] Mindcraft, Inc., "WebStone : The Benchmark for Web Server," http://www.mindcraft.com/web-stone.

[13] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," In Proc. ACM SIGMETRICS Conf., Madison, WI, Jul. 1998.

[14] H. Felix, K. Jeffay, and F. Smith, "Tracking the Evolution of Web Traffic," Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp.16-25, 2003.

[15] B. A. Mah, "An Empirical Model of HTTP Network Traffic," Proceedings of INFOCOM, pp. 592-600, 1997.

곽 후 근

E-mail : gobarian@q.ssu.ac.kr

1996년 호서대학교 전자공학과(학사)

1998년 숭실대학교 전자공학과(석사)

1998년~2006년 숭실대학교 전자공학과 (박사)

1998년 8월~2000년7월 (주) 3R 부설 연구소 주임 연구원

2006년 3월~현재 숭실대학교 전자공학과 대학원 (postdoc)

관심분야 : 네트워크 컴퓨팅 및 보안

정 규 식

E-mail : kchung@q.ssu.ac.kr

1979년 서울대학교 전자공학과(공학사)

1981년 : 한국과학기술원 전산학과 (이학석사)

1986년 미국 University of Southern California(컴퓨터공학석사)

1990년 미국 University of Southern California(컴퓨터공학박사)

1998년2월~1999년2월 미국 IBM Almaden 연구소 방문 연구원

1990년9월~현재 숭실대학교 정보통신전자공학부, 교수

관심분야 : 네트워크 컴퓨팅 및 보안