확장성 있는 무선 인터넷 프록시 서버 클러스터를 위한 동적 해싱 기반의 부하분산

곽 후 근[†] · 김 동 승[††] · 정 규 식[†††]

요 약

대용량 무선 인터넷 프록시 캐시 서버 클러스터에서는 성능 및 저장 공간의 확장성이 중요하게 되었다. 여기에서 성능의 확장성은 캐시 서버를 추가함에 따라 클러스터 성능이 선형적으로 증가함을 의미하고 저장 공간의 확장성은 캐시 데이터가 서버들에게 분할 저장되어 있어서 캐시 서버의 수에 상관없이 캐시 데이터를 저장하는 클러스터안의 공간의 합은 일정함을 의미한다. 대용량 서버 클러스터에서 많이 사용되는 라운드로빈 기반 부하분산 방법은 성능의 확장성은 보장되지만, 요청 URL 데이터가 모든 서버에 저장되어야 하므로 저장 공간의 확장성이 없는 단점을 가진다. 해싱기반 부하분산 방법은 모든 요청 URL 데이터가 서버들에 분할 저장되어 있어서 저장 공간의 확장성을 가진다. 그러나 그 방법은 사용자의 요청 패턴 불균형 또는 특정서버로의 요청 집중(Hot-Spot) 시에 성능 확장성이 없다.

본 논문에서는 성능 및 저장 공간의 확장성을 보장하는 새로운 동적 해싱 부하분산기법을 제안한다. 제안된 기법에서는 주기적으로 과부하 상태의 캐시 서버에 할당된 요청들을 찾아서 다른 캐시서버로 동적으로 재할당한다. 제안된 방법을 16대의 컴퓨터를 사용하여 실험을 수행하였고, 실험 결과를 통해 제안된 방법이 기존 방법과는 달리 성능 및 저장 공간의 확장성을 보장함을 확인하였다.

키워드 : 중복 저장, 저장 공간, 확장성, 요청 패턴, 요청 집중, 동적 해싱

A Dynamic Hashing Based Load Balancing for a Scalable Wireless Internet Proxy Server Cluster

Hukeun Kwak[†] · Dongseung Kim[††] · Kyusik Chung[†††]

ABSTRACT

Performance scalability and storage scalability become important in a large scale cluster of wireless internet proxy cache servers. Performance scalability means that the whole performance of the cluster increases linearly according as servers are added. Storage scalability means that the total size of cache storage in the cluster is constant, regardless of the number of cache servers used, if the whole cache data are partitioned and each partition is stored in each server, respectively. The Round-Robin based load balancing method generally used in a large scale server cluster shows the performance scalability but no storage scalability because all the requested URL data need to be stored in each server. The hashing based load balancing method shows storage scalability because all the requested URL data are partitioned and each partition is stored in each server, respectively. but, it shows no performance scalability in case of uneven pattern of client requests or Hot-Spot.

In this paper, we propose a novel dynamic hashing method with performance and storage scalability. In a time interval, the proposed scheme keeps to find some of requested URLs allocated to overloaded servers and dynamically reallocate them to other less-loaded servers. We performed experiments using 16 PCs and experimental results show that the proposed method has the performance and storage scalability as different from the existing hashing method.

Key Words : Duplication, Storage, Scalability, Request pattern, Hot-Spot, Dynamic hashing

1. 서 론

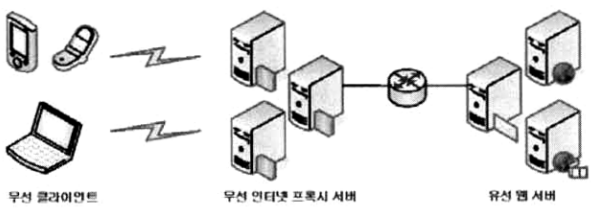
무선 인터넷은 무선으로 음성/데이터/영상 정보를 송수신할 수 있는 서비스라고 정의할 수 있으며, 무선 환경에서 인터넷을 비롯한 다양한 데이터 통신망에 접속하여 정보를 송수신하는 기술에 기반한다. 무선 인터넷은 무선랜을 이용한 인터넷 접속에 비하여 전송 용량 및 속도면에서 제한이 있으나 이동성이라는 커다란 장점을 가지고 있다. 무선인터넷은 이동 통신망의 발전에 따라 1세대 AMPS 망을 이용한 CDPD(Cellular Digital Packet Data)와 같은 초기 형태에서

※ 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11167-0) 지원 및 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 정 회 원 : 숭실대학교 정보통신전자공학부 Postdoc
†† 정 회 원 : 고려대학교 전기공학과 교수
††† 정 회 원 : 숭실대학교 정보통신전자공학부 교수(교신저자)
논문접수 : 2007년 8월 6일, 심사완료 : 2007년 11월 13일

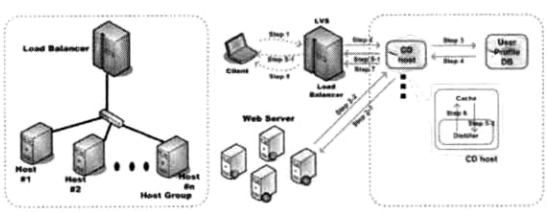
출발하여, 2세대인 IS-95/GSM을 거쳐 3세대인 IMT-2000 기반의 무선 인터넷으로 발전하는 과정에 있다. 이에 따라 무선 인터넷 대역폭의 효율적인 사용과 빠른 이용 시간을 위하여 캐시(Cache)와 압축(Distillation or Transconding) 기능을 가지는 무선 인터넷 프록시 서버의 필요성이 증대되고 있다. 무선 인터넷 프록시 서버는 무선 사용자를 유선 인터넷 서버에 연결 시켜주는 역할을 한다. (그림 1)은 무선 인터넷에 사용되는 무선 인터넷 프록시 서버를 나타내고 있다.

무선 인터넷 프록시 서버는 급증하는 사용자의 요청에 대한 확장성(Scalability)을 보장하기 위해 클러스터링 구조[1]를 가진다. 여기서 확장성이란 성능상의 확장성과 저장 공간상의 확장성으로 나눌 수 있다. 성능상의 확장성이란 서버를 추가함에 따라 이에 비례하여 성능이 증가해야 함을 의미하고, 저장 공간상의 확장성[2-8]이란 서버를 추가해도 전체적인 캐시의 크기는 일정하게 유지되어야 함을 의미한다. 저장 공간의 확장성은 캐시의 중복 저장이 있느냐/없느냐의 차이이다. 중복 저장이 있으면 서버 추가시 전체 캐시된 데이터가 증가되고, 중복 저장이 없으면 서버를 추가하여도 전체 캐시된 데이터는 동일하다.

무선 인터넷 프록시 서버는 무선 인터넷의 낮은 대역폭 해결하기 위해 캐싱(Caching)[9]과 압축(Distillation)[10]을 사용하며, 대용량 트래픽에 대한 확장성(Scalability)이 고려되어야 한다. TranSend[11]는 대용량 트래픽에 대한 확장성을 고려하여 클러스터링으로 구현된 무선 프록시 서버이다. 본 논문에서는 무선 인터넷 프록시 서버 클러스터인 TranSend를 확장성과 구조적인 관점에서 개선한 LVS-Cache 구조[12]를 사용하였다. 여기서, LVS는 Linux Virtual Server의 약자로써 부하 분산기로 동작한다. 자세한 내용은 2.1절을 참조하시오. (그림 2)는 LVS-Cache 프록시 시스템의 전체적인 구조를 나타낸다.



(그림 1) 무선 인터넷 프록시 서버



(a) 구조 (b) 동작 과정

(그림 2) LVS-Cache 프록시 시스템 구조

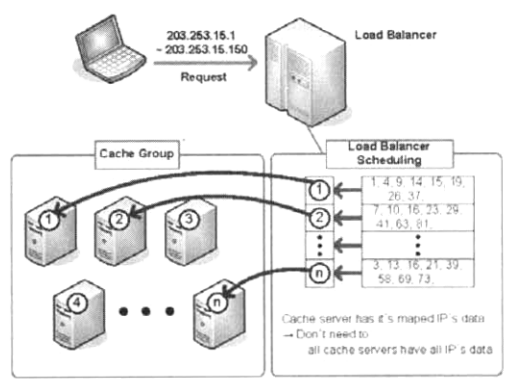
LVS-Cache의 전체 구조는 (그림 2(a))와 같고 자세한 설명은 다음과 같다. LVS-Cache 구조는 부하 분산을 위한 LVS와 Cache 호스트들로 구성되며 호스트는 CD(Cache & Distiller)로 구성된다. LVS는 클라이언트의 요청을 받아서 각 호스트들에게 전달하는 역할을 담당하며, CD는 클라이언트의 요청을 처리하는 Cache와 데이터에 대한 압축을 수행하는 Distiller의 역할을 함께 수행한다.

LVS-Cache의 동작 과정은 (그림 2(b))와 같고 자세한 설명은 다음과 같다. LVS가 스케줄링 알고리즘을 통하여 클라이언트의 요청을 각 호스트에 전달하게 되면, 호스트 내의 CD는 해당 데이터가 Cache에 존재하면 LVS에 전달하고, 존재하지 않으면 웹서버로부터 데이터를 요청하여 얻어온 후 압축과정을 거쳐 LVS에 전달하고 LVS는 그 데이터를 클라이언트에 보내는 방법으로 동작한다.

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 3)은 무선 인터넷 프록시 서버 클러스터에서 동일 요청 URL을 동일 캐시에 할당하는 방법을 나타낸다.

무선 인터넷 프록시 서버 클러스터에서 해시 함수를 이용하여 동일 요청 URL을 동일 캐시에 할당하는 방법에 대한 전체적인 동작 과정은 다음과 같다.

- 사용자 IP 주소는 예를 들면 (203.253.15.150)이 www.daum.net으로 이미지(URL)을 요청한다.
- 부하 분산기가 사용자의 Source IP(Destination IP 혹은 URL 등)을 통해 해시값을 생성한다.
- 해시값을 통해 해당 해시값에 할당된 캐시 서버로 이미지를 요청한다.
- 이후의 과정은 동일 해시값에 대해서는 동일 캐시 서버로 데이터를 요청한다.



(그림 3) 무선 인터넷 프록시 서버 클러스터에서 동일 요청 URL을 동일 캐시에 할당하는 방법

본 논문에서는 기존의 해싱을 이용한 부하 분산 방식이 가지는 문제점을 분석하고 이를 해결할 새로운 동적 해싱 기법을 제안한다. 본 논문의 구성은 다음과 같다. 제 2장에서는 무선 인터넷 프록시 서버 클러스터에서 기존의 해싱을 이용한 부하 분산 방식과 그 문제점을 소개한다. 3장에서는 기존의 해싱을 이용한 부하 분산 방식의 문제점을 해결하는 새로운 동적 해싱 방식을 이용한 부하 분산 방식을 설명하고, 4장에서는 실험 및 토론을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 기존 연구

MD5 Hashing[2]은 사용자 요청 URL에 대해 고정 길이의 해시 값이 나오고, 이를 캐시의 수에 매핑하는 방식으로 동작한다. 요청이 들어올 때 마다 메시지 축약을 생성, 클러스터에 할당하기 때문에 버킷을 위한 별도의 메모리 공간 소요가 없고, 캐시의 추가 및 삭제 시에 영향을 받지 않는 유동적인 구조를 가진다. 그러나 매 요청마다 해시 값을 계산하여 클러스터로 분산하기 때문에 사용자의 요청 수 증가에 따른 각 요청의 메시지 축약 계산 시간이 커진다는 단점을 가진다.

Consistent Hashing[3]의 캐시 포인트는 이진트리(Binary Tree) 형식으로 저장될 수 있고, 캐싱된 URL에 대한 검색은 이진트리 내에서 단일 검색(Single Searching) 방식이 이용된다. N개의 캐시 서버에서 검색을 위한 시간이 O(log$n$) 내에서 소요되기 때문에 캐시 서버가 증가하여도 캐싱된 URL에 대한 검색 시간이 별로 걸리지 않는 장점을 가진다. 그러나 요청 URL과 캐시 서버 이름에 대해서 MD5를 통하여 할당하기 때문에 해시 특성상 요청이 균일하게 분포되지 못하는 단점을 가지고 있고, 요청이 균일하게 분포되지 못하므로 서버 확장에 따른 성능의 선형적 증가를 보장하지 못한다. 또한 캐시 서버의 부하 상황을 부하 분산에 반영하지 못하기 때문에 순간적인 요청 몰림 현상(Hot-Spot)에 대처할 수 없는 단점을 가진다.

Cache Array Routing Protocol[4]은 일반적인 해시 기반 부하 분산 방식과 같이 ICP(Inter Cache Protocol) 요청(Query) 메시지 없이 해시에 의하여 데이터를 캐싱하고 있는 서버로 요청을 할당하며, 데이터의 중복 저장을 방지하는 효과를 가진다. 또한 캐시 서버 이름과 요청 URL의 해시 값에 대한 할당이 비교적 균일하게 분포되고 캐시 서버를 추가 할 때 전체적인 클러스터의 성능이 선형적으로 증가하므로 서버의 추가에 따른 확장성을 보장할 수 있다. 그러나 사용자의 요청에 대해서 해시 값을 계산하고 캐시 서버들의 해시 값과 조합하여 캐시 서버를 선택하는 방법을 사용하기 때문에 요청 처리에 대한 절차가 다소 복잡하고, 캐시 서버의 부하 상황을 반영하지 못하기 때문에 순간적인 요청 몰림 현상(Hot-Spot)에 대해서 대처할 수 없는 단점을 가진다.

Modified Consistent Hashing[5]은 기존 Consistent Hashing의 특징을 그대로 가지면서 각 서버의 요청 할당에 대한 표준 편차를 줄여 전체적으로 요청이 균일하게 분포되도록 하였다. 그러나 캐시 서버의 할당과 관련하여 여전히 랜덤 함수를 사용함으로 완전한 균일 분포를 이루지 못하는 단점을 가진다. 이로 인해 서버의 확장에 따른 성능의 선형적인 증가를 완벽하게 보장하지 못하며, 캐시 서버의 부하 상황을 반영하지 못하기 때문에 순간적인 요청 몰림 현상(Hot-Spot)에 대해서 대처할 수 없는 단점을 가진다.

Adaptive Load Balancing[6]은 2004년에 제안된 방식으로서 기존의 방식과 달리 해시 기반 부하 분산과 함께 라운드 로빈 부하 분산 방식을 동시에 사용하여 부하 분산 한다. 이 방식은 기본적으로 해시 기반 부하 분산 방식을 사용하지만, 요청 몰림(Hot-Spot) 상황의 처리에 초점을 둔다. 즉, 가장 자주 요청되는 데이터에 대해서 목록으로 관리하고, 해당요청이 발생하면 전체 캐시 서버에 라운드 로빈으로 분산시키는 방법을 사용한다. Adaptive Load Balancing은 다른 해시 기반 부하 분산 방식과 달리 요청 몰림 현상(Hot-Spot)에 대해서 적응적으로 처리할 수 있다는 장점을 가지지만, 그 이외의 사항인 캐시 서버의 추가/삭제 문제, 요청의 균일 분포 및 서버의 확장성에 대해서는 고려를 하지 않고 있다.

Extended Consistent Hashing[7]은 기존의 Consistent Hashing 방식의 변형으로 요청 컨텐츠를 캐시 서버에 할당할 때 컨텐츠의 부하 정보를 반영하였다. 이러한 방식을 이용하여 요청에 대해 캐시 서버를 할당하게 되면 요청 몰림 현상(Hot-Spot)을 처리할 수 있다. Extended Consistent Hashing은 요청 컨텐츠의 요청 빈도 및 이들의 평균 부하에 따라 캐시 서버 할당 영역을 설정하여 요청 몰림 현상(Hot-Spot)에 대해서 효과적으로 처리할 수 있다. 그리고 Consistent Hashing에 비해 요청의 균일한 분포를 나타낼 수 있는 장점을 가진다. 그러나 Consistent Hashing의 특성을 가지므로 전체적으로 균일한 요청의 분포를 나타내지 못하며, 이에 따라 캐시 서버의 확장에 따른 성능의 선형적 증가도 가지지 못하는 단점을 가지고 있다.

Cache Clouds Dynamic Hashing[8]은 기존의 일반적으로 작은 캐시 클러스터 그룹 환경에서 운영되는 해시 기반 부하 분산 방식과 달리 네트웍 끝단(Edge) 부분에서 내부 네트워크와 캐시 서버의 전체적인 성능 향상을 위하여 캐시 서버 간의 협동성을 가지게 하는 것이다. 이 방식은 Cache Clouds라는 개념을 소개하고, Cache Clouds 내에서 캐시 서버의 부하 정보를 고려하여 캐시 서버를 할당하는 동적 해싱 방식을 제안하였다. 제안된 방법은 캐시 서버의 부하 정보를 반영한 동적 해싱 방법을 사용하나, 요청의 균일한 분포 문제와 확장성에 따른 성능 증가 및 요청 몰림 문제에 대해서 고려되지 않은 것을 볼 수 있다.

본 절에서는 MD5 해싱을 이용하는 기존 해싱 방법의 문제점을 분석하고, 분석된 문제점을 바탕으로 본 연구의 접근 방식을 소개한다. MD5 해싱을 이용하는 기존 해싱 방법의 단점은 가중치를 가지는 요청 패턴이나 Hot-Spot 문제

로 인해 일부 캐시 서버로 요청이 집중된다는 점이다. 이러한 집중은 무선 인터넷 프록시 서버 클러스터의 전체 성능을 요청이 집중되는 일부 캐시 서버에 종속시키다. 즉, 전체적인 서버의 확장성 및 성능이 떨어진다.

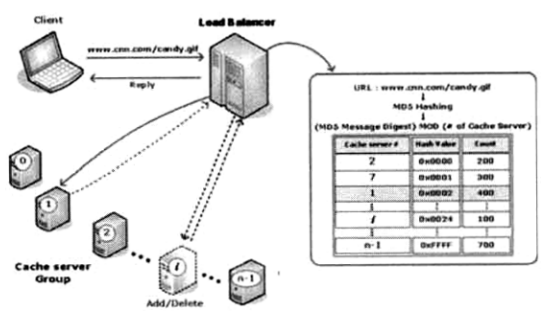
본 논문에서는 동적 해싱 기법을 사용하여 사용자의 요청 패턴 혹은 Hot-Spot 에서도 전체적인 사용자의 요청을 캐시 서버 사이로 균일하게 분포시키는 방법을 제안한다. 제안된 방법은 요청이 몰리는 캐시 서버에 할당된 해시값을 요청이 적은 캐시 서버로 재할당하는 방식으로 사용자의 요청 패턴 혹은 Hot-Spot으로 인해 요청이 일부 캐시 서버로 몰리는 것을 방지한다.

3. 제안된 동적 해싱 기법

3.1절은 제안된 방법의 전체적인 구조를 설명하고, 3.2절은 제안된 방법의 구체적인 동작 과정을 설명한다.

3.1 전체 구조

(그림 4)는 제안된 동적 해싱 기법의 전체적인 구조를 나타낸다. 사용자의 요청 URL은 MD5 해싱을 통해 해시값(MD5 Message Digest)을 가지게 되고, 이 값은 캐시 서버 개수와 MOD 연산을 통해 캐시 서버를 할당받게 된다. 이렇게 할당받은 해시값과 캐시 서버는 해시 테이블에 저장되고, 이후에 동일한 요청 URL에 대해서는 동일 해시값에 의해 동일 캐시 서버를 할당받게 된다. 이외에 해시 테이블에는 Count라는 값이 존재하고, 이는 각 해시값(URL)이 요청되는 회수를 카운트하는 하는 것이다. 이러한 Count 값은 동적 해싱 시에 캐시 서버를 재 할당할 해시값을 선택하는 기준으로 사용된다. 이에 대한 자세한 설명은 3.2절의 동작 과정에 설명되어 있다.



(그림 4) 제안된 동적 해싱 기법

3.2 동작 과정

제안된 동적 해싱 기법의 구체적인 동작 과정은 다음과 같다. 동작 과정 중의 업데이트 주기 및 해시값 개수는 4.2절의 실험 결과 부분을 참고하시오.

단계 1 : 주기적으로 사용자의 요청 패턴 혹은 Hot-Spot으로 인해 과부하 상태에 놓인 캐시 서버가 없는지 확인한다. 본 논문에서는 한 서버에 동시 연결 개수가 최대 400개가 넘으면 과부하 상태로 정의하였다. 서버의 사양에 따라 과부하 상태는 변할 수 있다.

단계 2 : 과부하 상태에 놓인 캐시 서버가 있다면, 과부하 캐시 서버가 처리하는 URL(해시값) 중 가장 많이 요청되는 URL(해시값)을 선택한다.

단계 3 : 과부하 캐시 서버에서 가장 많이 요청되는 URL(해시값)에 가장 부하가 적은 캐시 서버를 재 할당한다.

단계 4 : 업데이트해야할 URL(해시값) 개수에 따라 2, 3 단계를 반복한다. 업데이트해야할 URL(해시값) 개수가 2개라면 다음과 같이 URL(해시값)에 캐시 서버를 재 할당해주어야 한다.

- 과부하 캐시 서버 중 가장 많이 요청되는 URL(해시값)을 가장 부하가 적은 캐시 서버로 재할당한다.
- 2번째 과부하 캐시 서버 중 가장 많이 요청되는 URL(해시값)을 2번째로 부하가 적은 캐시 서버로 재할당한다.

단계 5 : 1-4 단계를 주기적으로 반복한다.

<표 1>은 기존 해싱 방법과 제안된 동적 해싱 방법을 사용자의 요청 패턴, Hot-Spot 및 확장성 관점에서 비교한 표이다.

4. 실험 및 토론

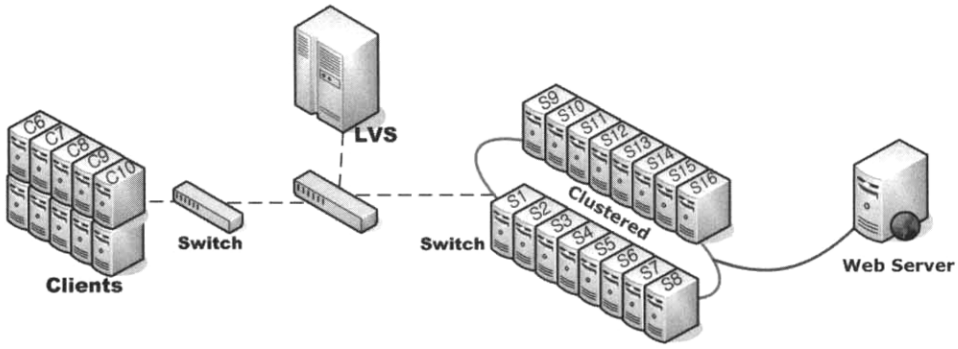
4.1절에서는 실험에 사용된 하드웨어와 소프트웨어 및 실험에 사용된 변수를 다루고, 4.2절에서는 기존 방법과 제안된 방법의 성능을 비교한 실험 결과를 다룬다.

4.1 실험 방법

제안된 방법을 테스트하기 위해, 본 논문에서는 (그림 5)에서 보이는 실험 환경을 구축하였다. 실험 환경은 4 종류의 서버들(클라이언트, LVS 기반의 Front-End 서버, 클러스터 서

〈표 1〉 기존 해싱 방법 vs. 제안된 동적 해싱 방법

	MD5 Hashing	MD5 Hashing 응용	제안된 동적해싱
사용자의 요청 패턴	X	X	O
Hot-Spot	X	O	O
확장성	X	X	O



(그림 5) 실험 환경

〈표 2〉 실험에 사용된 하드웨어와 소프트웨어

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

〈표 3〉 실험에 사용된 중요 변수

Variables	Values
Maximum number of cache servers	16
Distribution policy	RR, MD5, 제안된 방법(Dynamic Hashing)
Web traces (Types of client requests)	Surge : Image + HTML (100 files, weights)
Number of simultaneous connections per client to a server through LVS	63
Performance indicator	Requests serviced per second, Bits serviced per second

버들 및 웹 서버)로 구성된다. 서버들의 사양은 <표 2>와 같다.

클라이언트들은 인터넷으로부터 얻을 수 있는 Webstone [13,14]과 Surge[15,16]를 이용해서 요청을 생성한다. 이러한 요청들은 텍스트 파일 및 이미지들로 구성된다. KTCPVS는 해싱 방법에 기반해서 요청들은 어디로 어떻게 보낼 것인지를 결정한다. 본 논문에서 기존의 해싱 방법(RR, MD5[2], 제안된 방법-Dynamic Hashing)들이 구현되었고, 이들의 결과를 토대로 성능을 분석한다. 요청된 페이지들이 16개의 서버들 중 하나의 저장(캐싱)되면, 선택된 서버는 요청을 LVS를 통하지 않고 클라이언트에게 직접 보낸다. LVS는 들어오는 요청에 대한 처리만을 담당한다. 요청된 페이지가 캐시 서버에 저장되어 있지 않다면, 웹 서버에게 요청된다. 웹 서버로부터 새로운 컨텐츠를 받았다면, 이들은 선택된 캐시 서버에 저장(캐싱)된 후에 클라이언트에게 직접 전송될 것이다. Squid[17, 18]는 웹 서버로부터 받은 컨텐츠를 저장(캐싱)하는데 사용된다.

웹 서버로부터 받은 새로운 페이지가 저장될 때, 라운드 로빈 방식은 이들을 클러스터 내의 모든 서버에 저장된다. 반면, 다른 알고리즘은 선택된 하나의 캐시 서버에만 저장된다. 해당 요청에 대한 새로운 컨텐츠를 하나의 캐시 서버에만 저장하게 되면, 캐시 중복 문제를 없앰으로써 스토리지(저장 공간)의 확장성을 보장하게 된다. 라운드 로빈 방법은 비교 목적을 위해 사용된다. 왜냐하면, 모든 스케줄링 알고리즘 중에 가장 좋은 성능을 가지기 때문이다. 라운드 로빈 방법에서는 모든 서버가 동일한 파일들을 가지고 있음으로 특별한 요청 분배 방법이 필요 없고, 이는 특정 페이지에 대한 Hot-Spot 문제가 발생하지 않음을 의미한다. <표 3>은 실험에 사용된 중요 변수들을 나타낸다. 여러 변수들 중에 초당 처리되는 요청의 수는 제안된 방법들의 성능을 비교하는 중요한 변수로 사용된다.

서버들의 구성 및 다양한 변수들을 가지고, 초당 처리된 요청 수 및 초당 처리된 비트수 관점에서 실험 결과를 나타낸다.

4.2 실험 결과

<표 4> 및 (그림 6)은 업데이트 주기와 관련된 실험 결과를 보여준다. 업데이트 해시값 개수를 1로 고정하고 업데이트 주기를 1초에서 5초로 변화하면 1초에서 가장 좋은 성

〈표 4〉 업데이터 주기에 따른 초당 요청 수 (업데이트 해시값 개수 = 1개)

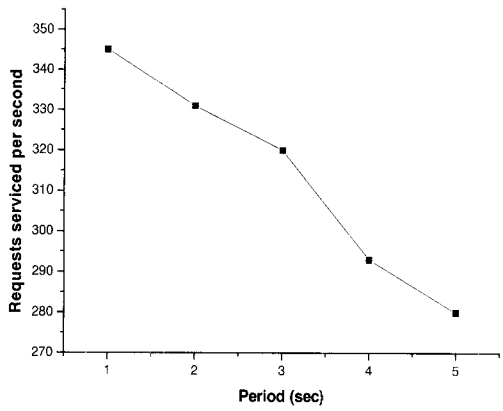
주기 (초)	1	2	3	4	5
초당 요청 수	345	331	320	293	280

〈표 5〉 업데이트 해시값 개수에 따른 초당 요청 수 (업데이트 주기 = 1초)

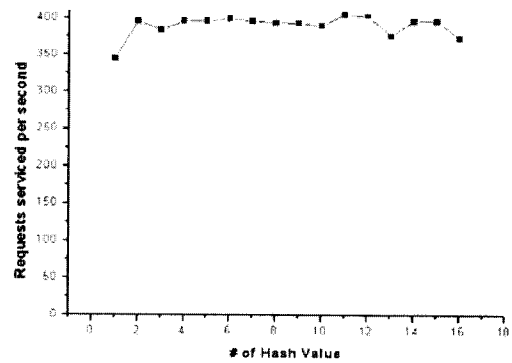
해시값 (개)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
초당 요청 수	345	396	383	396	396	399	396	394	393	390	405	403	376	396	396	373

〈표 6〉 서버의 개수에 따른 초당 처리된 요청 수 (Req/s)

해싱 방법	1	2	4	8	16
RR	39.47	78.85	157.35	215.13	431.42
MD5-Hashing	39.48	76.5	96.5	114.6	142.87
Dynamic-Hashing-8	38.68	74.2	151.73	213.78	398.03
Dynamic-Hashing-2	39.47	76.17	147.27	243.98	395.82



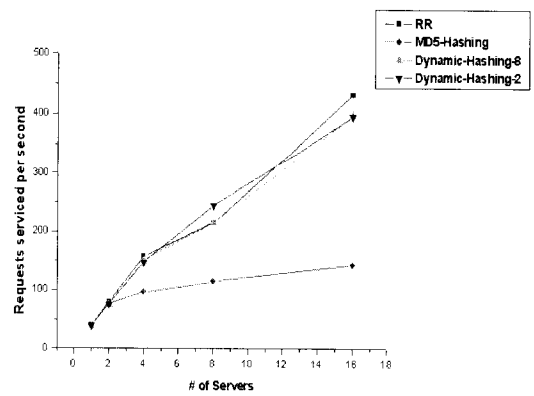
(그림 6) 업데이트 주기에 따른 초당 요청 수
(업데이트 해시값 개수 = 1개)



(그림 7) 업데이트 해시값 개수에 따른 초당 요청 수
(업데이트 주기 = 1초)

능을 보임을 알 수 있다. <표 5> 및 (그림 7)은 업데이트 해시값 개수와 관련된 실험 결과를 보여준다. 업데이트 주기를 1로 고정하고 업데이트 해시값 개수를 1개에서 16개로 변화하면 1개일 때를 제외하고 나머지 개수에서는 비슷한 성능을 보임을 알 수 있다. 이때, 업데이트 해시값 개수는 작을수록 좋다. 왜냐하면, 업데이트 해시값 개수가 많을수록 해시값이 많이 이동하게 되고, 이는 동일 데이터의 중복 저장 문제를 야기하기 때문이다. 본 논문에서는 이러한 실험 결과를 기반으로, 업데이트 주기를 1초로 설정하고 업데이트 해시값 개수를 2로 설정한 후 실험을 수행하였다.

<표 6>과 (그림 8)은 기존 방법(RR, MD5-Hashing)과 제안된 방법(Dynamic-Hashing-8, Dynamic-Hashing-2)에서 서버의 개수에 따른 초당 처리된 요청 수를 나타낸다. 제안된 방법에서 Dynamic-Hashing-8은 비교를 위해 업데이트 해시값 개수를 8개로 설정하고 실험을 수행한 것이다. 결과를 보면 2개로 놓고 실험한 Dynamic-Hashing-2와 비슷한 실험 결과를 가짐을 알 수 있다. 그럼으로 업데이트 해시값 개수를 2개로 설정해도 실험 결과에는 차이가 없음을 알 수 있다. 기존 실험의 경우 예상한데로 RR이 가장 좋은 성능을 보이고[19], MD5-Hashing이 가장 나쁜 성능을 보인다. RR의 경우 성능이 좋은 대신에 중복 저장으로 인해 저장 공간의 확장성은 없다.



(그림 8) 서버의 개수에 따른 처리된 초당 요청 수
(Req/s)

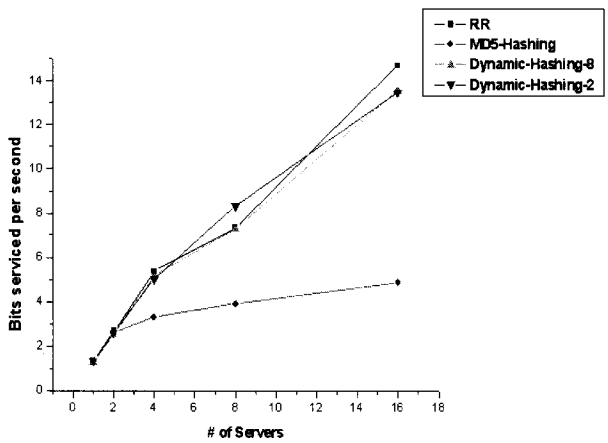
반면, MD5-Hashing의 경우 요청 URL(해시값)이 캐시 서버에 고정(Static)되어 있어 사용자의 요청 패턴이나 Hot-Spot을 처리할 수 없기 때문에 가장 나쁜 성능을 보임을 알 수 있다. 제안된 Dynamic-Hashing 기법은 캐시 서버의 부하에 따라 요청 URL(해시값)을 부하가 적은 캐시 서버로 옮김으로써(Dynamic) 저장 공간의 확장성을 유지하면서 RR과 비슷한 성능을 보인다. 즉, 기존 방법(MD5-Hashing)에 비해 제안된 방법(Dynamic-Hasing-8, Dynamic-Hashing-2)이 더 높은 성능상의 확장성을 가진다.

〈표 7〉 서버의 개수에 따른 초당 처리된 비트 수 (MBit/s)

해싱 방법	1	2	4	8	16
RR	1.34	2.68	5.36	7.32	14.69
MD5-Hashing	1.34	2.6	3.29	3.9	4.86
Dynamic-Hashing-8	1.32	2.53	5.17	7.28	13.55
Dynamic-Hashing-2	1.34	2.59	5.01	8.3	13.48

〈표 8〉 서버의 개수에 따른 성능 향상률 (기존 방법 vs. 제안된 방법)

서버의 개수	1대	2대	4대	8대	16대
제안된 방법과 RR의 성능차	0.00%	-3.40%	-6.41%	13.41%	-8.25%
제안된 방법과 MD5의 성능차	-0.03%	-0.43%	52.61%	112.90%	177.05%



(그림 9) 서버의 개수에 따른 처리된 초당 비트 수 (Mbit/s)

<표 7>과 (그림 9)는 기존 방법과 제안된 방법에서 서버의 개수에 따른 초당 처리된 비트수로 표현한 것이다.

<표 8>은 서버의 개수에 따른 제안된 방법의 성능 향상률을 나타낸다. 표에서 보면 제안된 방법은 RR에 비해 약간 성능이 떨어짐을 알 수 있는데, 이는 RR이 데이터를 중복 저장함으로써 좋은 성능을 유지하기 때문이다. RR의 경우 성능상의 문제점은 발생하지 않지만, 저장 공간상의 문제점이 발생한다. 예를 들어, 100Kbytes 이미지 1,000개를 요청한다면 제안된 방법에서 필요한 16대의 서버에 필요한 총 저장 공간은 100Mbytes가 되나, RR의 경우에는 모든 서버가 동일한 이미지를 저장하고 있음으로 16대의 서버에 필요한 총 저장 공간은 1.6GBytes(=16대 x 100Mbytes)가 된다.

반면에, 제안된 방법은 MD5-Hashing에 비해 높은 성능 향상률을 가짐을 알 수 있다. 이는 MD5-Hashing의 경우 사용자의 요청 URL(해시값)이 캐시 서버에 고정(Static)되어 있는 반면에, 제안된 방법은 사용자의 요청 URL(해시값)이 서버의 부하에 따라 이동(Dynamic)하기 때문이다.

제안된 방법의 장점은 성능 및 저장 공간의 확장성을 동시에 유지할 수 있다는 점이다. RR의 경우 성능의 확장성을 유지할 수 있지만, 저장 공간의 확장성을 유지할 수 없다. 반면, MD5-Hashing은 저장 공간의 확장성을 유지할 수 있지만, 성능의 확장성을 유지할 수 없다. 이에 반해, 제안된 방법은 캐시 서버의 부하에 따라 해시값을 동적으로 캐시 서버 상에서 옮길 수 있음으로 인해 성능 및 저장 공간의 확장성을 가진다.

제안된 방법의 단점은 일부 요청 URL(해시값)에 대한 데이터의 중복 저장 문제이다. 그러나 이는 RR처럼 모든 요청 URL에 대한 중복 저장이 아니라, 과부하가 발생한 캐시 서버의 일부 URL(해시값)에 국한된다.

5. 결 론

본 논문에서는 기존 해싱 기법의 단점을 보완하는 새로운 동적 해싱 기법을 제안하였다. 기존 해싱 기법이 요청 URL(해시값)을 캐시 서버에 정적(Static)으로 고정시켰다면, 제안된 방법은 캐시 서버의 부하에 따라 요청 URL(해시값)을 캐시 서버들 사이로 동적(Dynamic)으로 이동시켰다. 제안된 방법은 실험을 통해 성능 및 저장 공간의 확장성을 가짐을 확인하였다.

향후 연구 방향으로는 동적 해싱 기법을 캐시 서버의 추가/삭제에도 적용하는 것이다. 캐시 서버가 추가되면, 기존 캐시 서버들이 가지는 요청 URL(해시값)을 새로운 캐시 서버로 이동하는 것이다. 반대로 캐시 서버가 삭제되면, 삭제된 캐시 서버가 가지고 있던 요청 URL(해시값)을 남아 있는 캐시 서버들로 분산 이동하는 것이다. 또한 첫 번째 과부하 서버가 매우 많은 양의 부하를 가질 경우 그 부하를 연속해서 몇 군데에 나눠주는 방법도 가능하리라 생각된다.

참 고 문 헌

[1] LVS(Linux Vitual Server), http://www.linuxvirtualserver.org.
[2] D. Rivest, "The MD5 Message Digest Algorithm", RFC 1321, 1992.
[3] David Karger and al. "Web Caching with consistent hashing", In WWW8 conference, 1999.
[4] Micorsoft Corp., "Cache Array routing protocol and microsoft proxy server 2.0", White Paper, 1999.
[5] F. Baboescu, "Proxy Caching with Hash Functions", Technical Report CS2001-0674, 2001.
[6] Toyofumi Takenaka, Satosi Kato, and Hidetosi Okamoto, "Adaptive load balancing content address hashing

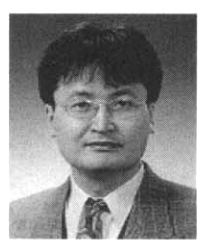
routing for reverse proxy servers", IEEE International Conference on Communications, Vol. 27, No. 1, pp. 1522-1526, 2004.
[7] S. Lei and A. Grama, "Extended consistent hashing: an efficient framework for object location", Proceeding of 24th International Conference on Distributed Computing Systems, pp. 254-262, 2004.
[8] L. Ramaswamy, Ling Liu, and A. Iyengar, "Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks", Proceedings of 25th IEEE International Conference on Distributed Computing Systems, pp. 229-238, 2005.
[9] 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.
[10] 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.
[11] A. Fox, "A Framework for Separating Server Scalability and Availability from Internet Application Functionality", Ph. D. Dissertation, U. C. Berkeley, 1998.
[12] 곽후근, 정규식, "통합형 무선 인터넷 프록시 서버 클러스터 구조", 한국정보처리학회논문지A, 제13-A권 제3호, 2006.
[13] Mindcraft, Inc., "WebStone : The Benchmark for Web Server", http://www.mindcraft.com/web- stone.
[14] J. Nakano, P. Montesinos, K. Gharachorloo, and J. Torrellas, "ReViveI/O: efficient handling of I/O in highly-available rollback-recovery servers", The 12th Internation Symposium on High-Performance Computer Architecture, pp. 200-211, 2006.
[15] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", In Proc. ACM SIGMETRICS Conf., Madison, WI, Jul. 1998.
[16] R. Zhang, T. Abdelzaher, and J. Stankovic, "Efficient TCP connection failover in Web serer clusters", 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1219-1228, March 2004.
[17] Squid Web Proxy Cache, http://www.squid-cache.org.
[18] W. Liao and P. Shih, Architecture of proxy partial caching using HTTP for supporting interactive video and cache consistency, 11th International Conference Computer Communications and Networks, pp. 216-221, 2002.
[19] 곽후근, 정규식, "캐시 서버 클러스터를 위한 동적 서버 정보 기반의 해싱 기법", 한국정보과학회 HPC 연구회 학술발표회 논문집, Vol. 17, No. 2, pp. 3-11, 2006.

곽 후 근

e-mail : gobarian@q.ssu.ac.kr
1996년 호서대학교 전자공학과(학사)
1998년 숭실대학교 전자공학과 대학원 (석사)
1998년~2006 숭실대학교 전자공학과 대학원(박사)
1998년 8월~2000년 7월 (주) 3R 부설 연구소 주임 연구원
2006년 3월~현재 숭실대학교 정보통신전자공학부(postdoc)
관심분야 : 네트워크 컴퓨팅 및 보안

김 동 승

e-mail : dkim@classic.korea.ac.kr
1978년 서울대학교 전자공학과(학사)
1980년 한국과학기술원 전자전기공학과 (석사)
1980년~1983년 경북대학교 전임강사
1988년 미국 University of Southern California(박사)
1988년~1989년 미국 University of Southern California PostDoc
1989년~1995년 포항공과대학 부교수
1995년~현재 고려대학교 전기공학과 교수
관심분야 : highly available scalable WWW server, dynamic load balancing in WWW servers



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)
1998년 2월~1999년 2월 : 미국 IBM Almaden 연구소 방문 연구원
1990년 9월~현재 숭실대학교 정보통신전자공학부, 교수
관심분야 : 네트워크 컴퓨팅 및 보안