공유 캐시 디렉토리 기반의 무선 인터넷 프록시 서버 클러스터

곽 후 근[†] · 정 규 식[††]

요 약

본 논문에서는 무선 인터넷 프록시 서버 클러스터를 사용하여 무선 인터넷의 문제와 요구들을 캐싱(Caching), 압축(Distillation) 및 클러스터(Clustering)를 통하여 해결하려고 한다. 무선 인터넷 프록시 서버 클러스터에서 고려되어야 하는 것은 시스템적인 확장성, 단순한 구조, 캐시간 협동성(Cooperative Caching), Hot Spot에 대한 처리 등이다. 본 연구자들은 기존 연구에서 시스템적인 확장성과 단순한 구조를 가지는 CD-A라는 구조를 제안하였으나 캐시간 협동성이 없다는 단점을 가진다. 이의 개선된 구조로 해쉬를 이용하여 사용자의 요청을 처리하는(캐시간 협동성을 가지는) 구조를 생각해 볼 수 있으나 이 역시 Hot Spot(과부하)을 처리할 수 없다는 단점을 가진다. 이에 본 논문에서는 시스템적인 확장성, 단순한 구조, 캐시간 협동성, Hot Spot(과부하)을 처리할 수 있는 공유 캐시 디렉토리 기반의 무선 인터넷 프록시 서버 클러스터를 제안한다. 제안된 방법은 하나의 캐시 디렉토리를 공유하는 방법으로 기존 구조의 장점과 캐시간 협동성 및 Hot Spot(과부하)을 처리할 수 있다는 장점을 가진다. 16대의 컴퓨터를 사용하여 실험을 수행하였고 실험 결과 Hot Spot(과부하) 상황에서 제안된 방법이 높은 성능 향상을 가짐을 확인하였다.

키워드 : 무선 인터넷, 프록시 서버, 클러스터링, 캐시간 협동성, Hot Spot(과부하), 공유 캐시 디렉토리

A Shared Cache Directory based Wireless Internet Proxy Server Cluster

Hukeun Kwak[†] · Kyusik Chung[††]

ABSTRACT

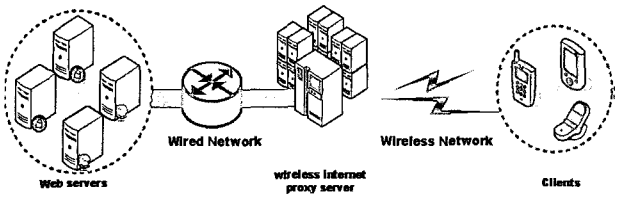
In this paper, wireless internet proxy server clusters are used for the wireless internet because their caching, distillation, and clustering functions are helpful to overcome the limitations and needs of the wireless internet. A wireless Internet proxy server cluster needs a systematic scalability, simple communication structure, cooperative caching, and serving Hot Spot requests. In our former research, we proposed the CD-A structure which can be scalable in a systematic way and has a simple communication structure but it has no cooperative caching. A hash based load balancing can be used to solve the problem, but it can not deal with Hot Spot request problem. In this paper, we proposed a shared storage based wireless internet proxy server cluster which has a systematic scalability, simple communication structure, cooperative caching, and serving Hot Spot requests. The proposed method shares one cache directory and it has advantages: advantages of the existing CD-A structure, cooperative caching, and serving Hot Spot requests. We performed experiments using 16 PCs and experimental results show high performance improvement of the proposed system compared to the existing systems in Hot Spot requests.

Key Words : Wireless Internet, Proxy Server, Clustering, Cooperative Caching, Hot Spot, Shared Storage

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. 그러나 무선 인터넷의 사용이 증가하는 만큼 무선 인터넷의 본질적인 문제 역시 무시할 수 없는 요소로 부각되고 있다. 현재까지 나와 있는 무선 인터넷의 근본적인 문제점은 낮은 대역폭, 빈번하게 연결이 끊김, 단말기 내의 낮은 컴퓨팅 파워 및 작은 화면, 단말기 사용자의 이동성, 네트워크 프로토콜, 보안 등이 있다. 그리고 급속도로 증가하는 수요에 따라 무선 인터넷 서버는 대용량 트래픽을 처리할 수

※ 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11167-0)지원으로 수행되었음.
† 준 회 원 : 숭실대학교 전자공학과 대학원 졸업
†† 정 회 원 : 숭실대학교 정보통신전자공학부 교수
논문접수 : 2006년 4월 17일, 심사완료 : 2006년 6월 8일



(그림 1) 무선 인터넷 프록시 서버

있는 확장성이 요구되어지고 있다.

본 논문에서는 이러한 무선 인터넷의 문제점 및 요구들을 캐싱(Caching)[1], 압축(Distillation)[2] 및 클러스터링(Clustering)을 통하여 해결하는 방법으로 클러스터링 기반의 무선 인터넷 프록시 서버를 사용하였다. (그림 1)은 무선 인터넷에 사용되는 무선 인터넷 프록시 서버를 나타내고, 이는 무선 사용자를 유선 인터넷 서버에 연결 시켜주는 역할을 한다.

본 연구자들은 기존 논문[3]에서 기존의 클러스터링 기반의 무선 인터넷 프록시 서버인 TranSend[4]를 개선한 CD-A라는 구조를 제안하였다. CD-A는 TranSend의 확장성 문제를 해결하기 위해 만든 분산 구조(Distributed Structure)이다. 이러한 구조적인 분산 구조는 캐시의 분산을 가지고 오고 데이터 분산을 어떻게 할 것인가(데이터를 공유할 것인가 아니면 독립적으로 저장할 것인가)에 대한 문제를 가지게 된다. CD-A에서 캐시에 저장하는 데이터의 분산 문제를 해결하는 구조로써 스케줄링 시에 해싱(Hashing)을 이용하는 방법이 있다. 해싱을 이용하게 되면 CD-A 분산 구조가 가지는 데이터 분산 문제를 해결할 수 있지만, 요청이 하나의 서버로 몰리는 Hot-Spot이 발생하게 되면 클러스터의 전체 성능이 요청이 몰리는 서버에 종속되는 단점을 가진다. 이에 본 논문에서는 CD-A 및 이의 개선 구조(해싱을 이용한 스케줄링)가 가지는 문제점을 지적하고 이를 해결하는 새로운 구조를 제안한다. 그리고 실험을 통해 이들의 성능을 비교하는데 초점을 맞춘다.

본 논문의 구조는 다음과 같다. 2장에서는 기존 무선 인터넷 프록시 서버와 이들이 가지는 문제점을 소개한다. 3장에서는 기존 무선 인터넷 프록시 서버가 가지는 문제점들을 해결하는 새로운 구조를 설명하고, 4장에서는 실험 및 토론을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 연구 배경

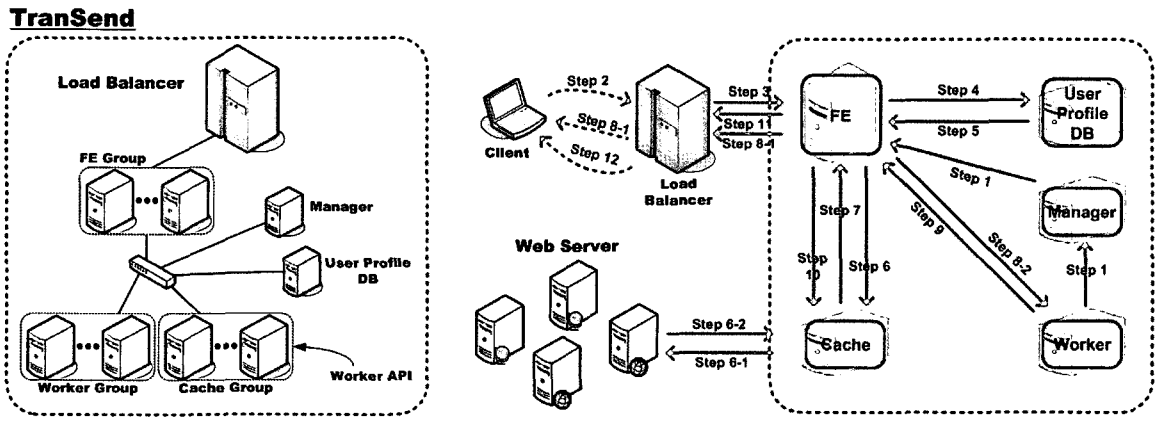
CD-A는 TranSend의 확장성 문제를 해결하기 위해 만든 분산 구조(Distributed Structure)이다. 이러한 구조적인 분산 구조는 캐시의 분산을 가지고 오고 데이터 분산을 어떻게 할 것인가에 대한 문제를 가지게 된다. 즉, 데이터를 공유할(Shared) 것이냐 아니면 각기 독립적으로 사용할 것이냐의 문제를 가지게 된다. 데이터를 공유하게 되면 서버들이 가지게 되는 전체 캐시의 합이 캐시 서버에 무관하게 일정하게 유지할 수 있는 반면, 데이터를 독립적으로 가지게 되면 캐시 서버 수에 비례하여 전체 캐시의 합이 증가하게 된다. 이러한 문제를 해결하는 구조로써 스케줄링 시에 해싱(Hashing)을 이용하는 방법이 있다. 해싱을 이용하게 되면 특정 요청을 특정 서버에서만 처리하게 됨으로 데이터를 공유하게 되는 장점을 가지게 되지만, 요청이 하나의 서버로 몰리는 Hot-Spot이 발생하게 되면 클러스터의 전체 성능이 요청이 몰리는 서버에 종속되는 단점을 가진다. 이러한 각 구조의 단점을 자세하게 기술하면 다음과 같다.

2.1 TranSend

2.1.1 구조

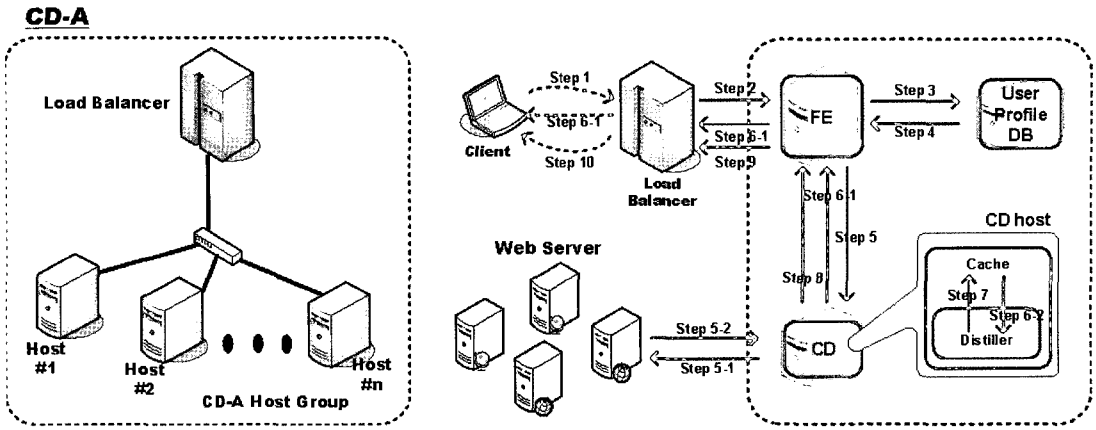
TranSend[4] 구조는 각 모듈로써 Front End(FE, MS: Manager Stub), User Profile DB, Cache($), Worker(W, Worker API, WS:Worker Stub), Manager, Graphical Monitor가 있다. FE는 Client 요청에 대한 외부 인터페이스를 담당하며, User Profile DB는 사용자와 관련된 정보(Preference)를 저장한다. Cache는 Client의 요청을 처리하며, Worker(Datatype-Specific Distiller)는 데이터에 대한 압축을 수행한다. Manager는 Distiller를 관리하고, Graphical Monitor는 시스템 전체의 상태를 볼 수 있게 해준다.

(그림 2) (a)는 TranSend의 구조를 나타내며, 각 모듈에



(a) 구조 (b) 동작 과정

(그림 2) TranSend 무선 인터넷 프록시 서버



(a) 구조 (b) 동작 과정

(그림 3) CD-A 무선 인터넷 프록시 서버

대한 자세한 설명은 참고문헌[5]에 기술되어 있다. TranSend에서는 각각의 모듈들(Front End, Cache, Distiller)이 클러스터링 되는 구조로 되어 있다.

2.1.2 동작 과정

TranSend 구조에서 FE는 부하 분산기(Load Balancer)를 통해 사용자 요청을 받고 이를 캐시로 보낸다. 요청한 데이터가 캐시에 없다면 캐시는 이 데이터를 웹 서버에 요청한다. 캐시가 웹 서버에서 받아온 데이터는 다시 FE로 보내진다. 압축이 필요한 경우, FE는 데이터를 Distiller로 보내어 압축하고, 이를 다시 받아온다. FE는 이 압축된 데이터를 캐시에 저장하고 사용자에게 응답한다.

(그림 2) (b)는 구체적인 동작 과정을 나타내고 이에 대한 자세한 설명은 참고문헌[5]에 기술되어 있다.

2.2 CD-A

2.2.1 구조

CD-A는 TranSend 모듈(FE, Cache, Distiller)에서 Distiller를 없애고 Cache에 압축(Distillation) 기능을 추가한 것이다(이하 CD 모듈: Cache & Distiller). 그리고 이 모듈(FE, CD)을 하나의 호스트에 넣고(이하 CD-A 구조: CD[3] & All-in-one[5]) LVS(Linux Virtual Server)[6]를 사용하여 부하 분산을 하는 것이다. (그림 3) (a)는 CD-A의 구조를 나타낸다. TranSend에서는 각각의 모듈들(FE, Cache, Distiller) 각각이 클러스터링 되어 있는 반면에 CD-A에서는 각 모듈들(FE, CD)을 하나의 호스트에 통합하고 이러한 호스트들을 클러스터링하는 구조로 되어 있다.

Distiller를 없애고 Cache에 압축 기능을 추가한 이유는 복잡한 구조를 단순화하고 불필요한 통신을 줄이기 위해서이다. 그리고 각 모듈들(FE, CD)을 하나의 호스트에 넣은 이유는 시스템적으로 확장하는 구조를 만들기 위해서이다. 즉, TranSend에서는 새로운 모듈을 추가 시에 동작과정중의 병목 현상을 보이는 모듈을 찾아서 추가해야하는(No Systematic) 반면에, CD-A는 병목에 상관없이 새로운 호스트를 추가하면(Systematic), 그 호스트 내의 모듈(FE, CD) 중에 필요한 모듈이 별도의 설정 없이 상대적으로 많이 사용되는 장점을 가진다.

2.2.2 동작 과정

(그림 3) (b)는 CD-A 구조의 구체적인 동작 과정을 나타낸다. 사용자의 부하 분산기에게 데이터를 요청하고, 부하 분산기는 FE에게 데이터 요청을 보낸다. FE는 User Profile DB로부터 사용자 정보(Preference)를 얻고, CD에게 사용자 요청을 보낸다. CD 안에 요청 데이터가 없다면, 외부 웹서버에 데이터를 요청하고 이 데이터의 압축 유무를 판단하여 압축을 수행한다. 그리고 CD는 이 압축 데이터를 저장하고 FE로 보낸다.

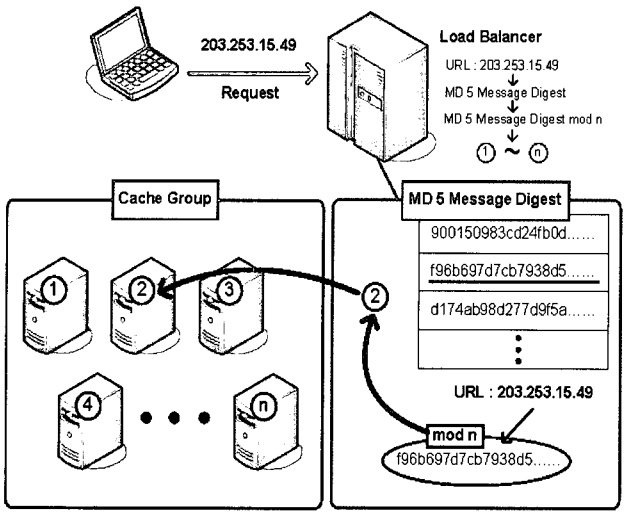
2.3 해쉬를 이용한 스케줄링 (MD5 Hashing)

TranSend 구조를 개선한 CD-A 구조는 시스템적인 확장성과 단순한 구조를 가지나 캐시간 협동성이 없다는 단점을 가진다. 즉, 다른 캐시 서버에 동일 데이터가 저장되어 있어도 로컬에 데이터가 없다면 웹 서버로 다시 데이터를 요청한다는 점이다. 이로 인해 전체 캐시의 합의 캐시 서버의 수 및 사용자의 요청에 비례하여 증가한다. 즉, 하나의 요청에 대한 캐시 데이터는 모든 캐시 서버에 저장된다.

이를 개선하는 방법으로 사용자의 요청을 캐시 서버로 할당 할 때 해쉬를 사용하는 방법이 있다. 사용자의 요청을 해싱하여 이를 캐시 서버로 매핑하면 동일 요청은 동일 캐시 서버에서 처리함으로 캐시간 협동을 보장하게 된다. 캐시간 협동이 보장되면 전체 캐시의 합은 캐시 서버의 수와 무관하게(Mutually Exclusive) 사용자의 요청에만 비례하여 증가한다. 즉, 하나의 요청에 대한 캐시 데이터는 전체 캐시에서 하나의 캐시에만 저장된다.

(그림 4)는 CD-A에서 해쉬(MD5[7]) 스케줄링을 이용해서 사용자 요청을 캐시 서버로 할당하는 것을 나타내고, 이를 요약하면 다음과 같다.

- 부하 분산기가 사용자의 요청 목적지 주소(혹은 소스 주소, URL)를 통해 해쉬값(MD5 메시지 다이제스트)을 생성한다.



(그림 4) 해싱(MD5)을 이용한 스케줄링

- 생성된 해쉬값을 캐수 개수(N)로 나머지 연산(Mod)한다.
- 나머지 연산을 통해 나온 값과 동일한 캐시로 요청을 할당한다.

2.4 접근 방식

본 절에서는 CD-A 무선 인터넷 프록시 서버 및 이를 개선한 해쉬를 이용한 스케줄링의 문제점을 정리하고, 3장에서는 이를 해결할 새로운 무선 인터넷 프록시 구조를 제안한다.

2.4.1 CD-A 구조의 문제점

- 캐시간 협동성(Cooperative Caching) : 다른 캐시에 현재 요청하는 데이터가 있어도 이를 이용하지 못하고, 이를 자신의 로컬 캐시에 저장함으로 요청에 대한 응답 시간이 증가하고 전체적인 캐시 합이 커지는 단점을 가진다. 이의 문제를 해결하기 위해 Cache 라우팅 테이블(Cache Routing Table)[8]이나 CARP(Cache Array Routing Protocol)[9]의 적용을 고려해 볼 수 있다. 그러나 이러한 방법들은 요청받은 데이터가 자신에게 없으면 다른 서버에게 요청하여 자신의 로컬 캐시에 저장 및 응답하는 방식으로 전체적인 응답 시간이 증가하고 전체 캐시 합이 약간 커지는 단점을 가진다.

2.4.2 해쉬를 이용한 스케줄링(MD5 Hashing)의 문제점

- Hot Spot(과부하) : 특정 목적지 주소(혹은 소스 주소, URL)로 요청이 몰리면 해쉬의 특성상 특정 캐시로 요청이 몰리게 된다. 이로 인해 무선 인터넷 프록시 서버의 전체 성능이 요청이 몰리는 특정 캐시에 의존되어, 성능이 크게 감소한다.

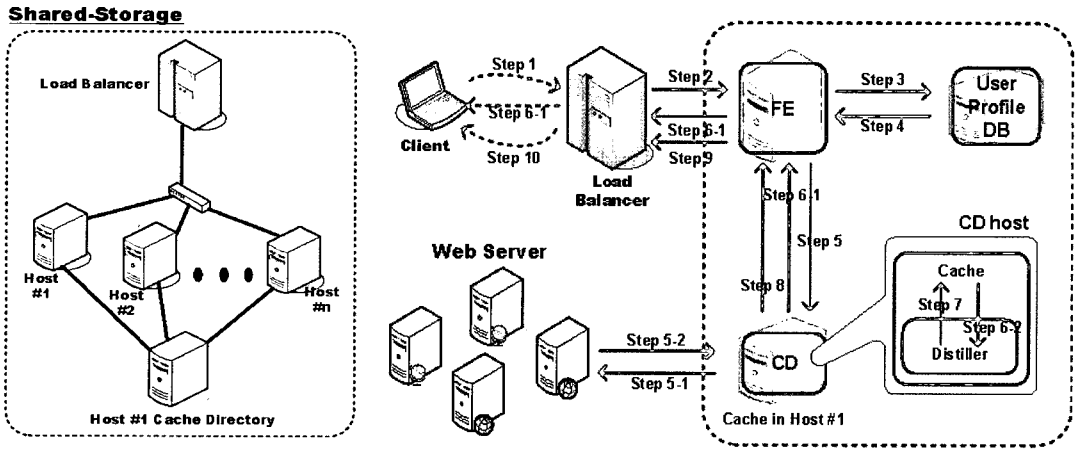
2.4.3 본 연구의 접근 방식

본 논문에서는 캐시간 협동성을 가지면서 Hot Spot(과부하)을 처리할 수 있는 새로운 구조를 제안한다.

3. 제안된 구조

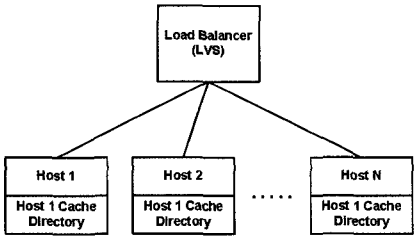
3.1 접근 방식

(그림 5) (a)는 2.3절에서 분석된 CD-A와 MD5 Hashing의 문제점을 기반으로 이를 해결할 수 있도록 제안된 새로운 구조이다. 제안된 새로운 구조에서는 1번 호스트의 캐시 디렉토리를 공유하고 다른 호스트들이 이 공유 캐시 디렉토리를 이용하는 것이다. 기존 방법이 각 호스트 내의 캐시 디렉토리를 사용하는 반면, 제안된 방법에서는 1개의 캐시 디렉토리를 사용하도록 되어 있다. 제안된 구조에서 1개의 캐시 디렉토리를 공유하고 모든 호스트들이 이를 사용하는 이유는 다음과 같다. 하나의 캐시 디렉토리를 공유함으로써 캐시간 협동성을 유지할 수 있고, Hot Spot(과부하)이 발생해도 하나의 호스트에서 이를 처리하지 않고 모든 호스트가 분담해서 처리함으로 전체적인 성능이 Hot Spot(과부하)을 처리하는 서버에 종속되지 않는다.



(a) 구조 (b) 동작 과정

(그림 5) 제안된 무선 인터넷 프록시 서버



(그림 6) 캐시 디렉토리 공유

리눅스 상에서 호스트 1번의 캐시 디렉토리를 공유하기 위해 NFS(Network File System)[10]을 사용하였다. 1번 호스트가 NFS 서버를 실행하고 공유할 디렉토리를 exports하면, 다른 호스트는 이를 자신의 캐시 디렉토리에 mount하여 사용하는 방법으로 실험을 수행하였다. 이를 그림으로 표현하면 (그림 6)과 같다.

(그림 6)에서 보면 1번의 캐시 디렉토리에 오류(Fault)가 발생하면 네트워크 전체가 마비 혹은 성능이 떨어지는 문제가 발생하게 된다. 이러한 단일장애점(A single of failure)이 되는 것을 막기 위해 공유 캐시 디렉토리는 Backup 캐시 디렉토리를 고려할 수 있다. Backup 캐시 디렉토리의 기능은 공유 캐시 서버와 주기적으로 정보를 주고 받다가 공유 캐시 디렉토리에 문제가 발생하면 이의 기능을 Backup 캐시 디렉토리가 대신 수행하는 것이다.

3.2 동작 과정

(그림 5) (b)는 제안된 구조의 전체적인 동작 과정을 나타낸다. 기본적인 동작 과정은 CD-A와 같고, 캐시가 데이터가 저장된 캐시 디렉토리를 이용할 때는 모두 1번 호스트의 캐시 디렉토리를 이용한다는 점이 틀리다. 이에 대한 자세한 설명은 다음과 같다.

단계 1: 사용자는 부하 분산기(LB: Load Balancer)에게 데이터를 요청한다.

단계 2: 부하 분산기는 FE에게 데이터 요청을 보낸다. (이 FE는 부하 분산기에서 스케줄링 방식에 의해 선택된 것이다.)

단계 3: FE는 User Profile DB에게 사용자 정보(Preference)를 요청한다.

단계 4: User Profile DB는 사용자 정보를 FE에게 보낸다.

단계 5: FE는 CD에게 사용자 요청을 보낸다.

단계 5-1: CD 안에 요청 데이터가 없다면, 외부 웹서버에 데이터를 요청한다.

단계 5-2: 웹서버는 데이터를 CD에 보내고, CD는 이를 (1번 호스트의 캐시 디렉토리에) 저장한다.

단계 6: CD는 데이터를 확인한다.

단계 6-1: 데이터가 압축된 형태라면, 사용자에게 데이터를 보낸다.

단계 6-2: 데이터가 압축된 형태가 아니라면, 사용자 정보를 이용하여 데이터를 압축한다.

단계 7: CD는 압축된 데이터를 (1번 호스트의 캐시 디렉토리에) 저장한다.

단계 8: CD는 압축된 데이터를 FE에게 보낸다.

단계 9: FE는 부하 분산기로 압축된 데이터를 보낸다.

단계 10: 부하 분산기는 사용자 요청에 응답한다.

3.3 기존 구조와의 비교

<표 1>은 CD-A 및 MD5 Hashing을 제안된 시스템과 구조적으로 비교한 표이다.

〈표 1〉 기존 구조 vs. 제안된 구조(LC)

	CD-A	MD5 Hashing	제안된 구조
확장성(Scalability)	Systematic	Systematic	Systematic
구조(Structure)	LVS-FE-Cache	LVS-FE-Cache	LVS-FE-Cache
캐시간 협동성	X	O	O
Hot-Spot	O	X	O

4. 실험 및 토론

4.1 실험 환경

<표 2>는 실험에 사용된 하드웨어와 소프트웨어를 나타낸다. 무선 인터넷 프록시 서버는 PC 16대로 구성하였고 TranSend 및 제안된 시스템에서 FE의 부하 분산과 All-in-one 시스템의 부하 분산을 위하여 LVS라는 Load Balancer를 사용하였다. Apache Bench라는 프로그램을 Client에서 수행하여 프록시 서버에 영상(이미지)를 요청하는 방식으로 실험하였다. 표에서 Client와 LVS가 Host보다 하드웨어 성능이 좋은 이유는 확장성 실험을 할 때 Client와 LVS에서는 병목이 발생하지 않는 조건에서 프록시 서버 내 호스트들 사이의 확장성을 확인하고자 했기 때문이다.

〈표 2〉 실험용 하드웨어 & 소프트웨어

		하드웨어		소프트웨어	개수
		CPU (Hz)	RAM (MB)		
Client		P4 2.26 M	256	AB[11]	2
LVS		P4 2.4 G	512	DR[6]	1
Host	Cache	P2 400 M	256	Squid[12]	16
	Distiller			JPEG-6b[13]	

4.2 실험 방법

<표 3>은 실험에 사용된 변수를 정리한 것이다. 사용자의 요청 개수는 약 200초 동안 프록시 서버가 처리할 수 있는 최대 개수를 사용하였다. 요청 시간을 200초 이상으로 하면 전체적인 실험 결과에는 영향을 미치지 않는 것을 확인하였고, 전체적인 실험 시간을 고려하여 200초로 제한하였다. 사용자의 요청 컨텐츠는 웹에서 가장 많은 사용 빈도를 가지는 JPEG 이미지[14]를 사용하였으며 요청 크기는 300 bytes에서 100 Kbytes 사이의 이미지를 사용하였다. 요청의 다양화를 위해 Variation Kbytes를 사용하였고, 같은 이름(vari.jpg)으로 1 Kbytes에서 10 Kbytes 까지 10개의 서로 다른 크기의 이미지를 저장하고 사용자가 이를 요청하도록 하였다[15]. 이렇게 요청을 하게 되면 같은 동일 서버 내에서 동일 이름으로 서로 다른 크기의 이미지를 요청하는 효과를 가지게 된다.

MD5 Hashing는 목적지 주소(Destination IP)를 대상으로 해쉬값을 얻었으면, 이러한 실험을 위해 사용자는 255개의 목적지 주소를 반복해서 요청하도록 하였다. Hot Spot(과부하)

〈표 3〉 실험에 사용된 변수

사용자의 요청 개수	◦ 약 200초 동안 프록시가 처리할 수 있는 최대 개수
요청 이미지	◦ JPEG
요청 크기	◦ 300 bytes, 1 K, 10 K, 100 Kbytes, Variation
요청 방식	◦ 같은 크기 : 다른 이름(1K00.jpg, 1K01.jpg 등)으로 랜덤(Random)하게 요청 ◦ Variation : 1K-10K 사이의 이미지를 랜덤하게 요청(Vari01.jpg -> 1K, Vari02.jpg -> 2K 등) ◦ 목적지 주소 : 255개의 주소(IP)를 반복해서 요청 ◦ Hot Spot(과부하) : 255개의 주소(IP)를 반복해서 요청하되, 2번 중에 1번은 동일 IP로 요청
스케줄링 (LVS)	◦ CD-A : RR(Round-Robin) 방식 ◦ MD5 Hashing : MD5 Hash 방식 ◦ Shared-Storage : LC(Least-Connection) 방식
사용자 정보 (Preference)	◦ 이미지 Quality = 중간
웹 서버	◦ Cache 서버 자체에 둠 (프록시내의 성능 평가에 초점을 맞춤)
병목 (bottleneck)	◦ 호스트의 CPU 점유율 중 가장 높은 호스트 (본 실험에서는 ethernet이나 system bus 병목은 발생하지 않는다.)

실험을 할 때는 255개의 목적지 주소를 사용하되, 두 번 중에 한번 은 동일 목적지 주소를 요청하게 하여 하나의 호스트로 요청이 몰리도록 하였다. 표에서 보면 제안된 공유 디렉토리에서는 스케줄링 알고리즘이 해쉬가 아닌 LC(Least Connection) 방식이 사용된 것을 볼 수 있다. 이것은 제안된 구조가 하나의 캐시 디렉토리를 공유함으로써 캐시간 협동성을 얻기 위해 해쉬 알고리즘을 사용할 필요가 없음을 의미한다.

각 구조의 실험은 다음과 같은 과정으로 수행하였다.

1) 기본 CD-A 시스템을 구성한다. (Host 1)
2) AB(Apache Bench)를 이용하여 TranSend로 JPEG 이미지를 약 200초 동안 무선 인터넷 프록시 서버가 처리할 수 있는 최대 개수로 요청한다.
3) 초당 요청 개수를 측정한다.
4) CD-A 호스트를 추가한다.
5) 실험에 사용된 호스트 수(16대)만큼 2)-4)를 반복한다.

4.3 실험 결과

4.3.1 RR

<표 4>와 (그림 7)은 CD-A 구조에서 Round-Robin 스케줄링 방식을 사용하여 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다. (그림 6)을 보면 호스트의 개수에 따라 성능이 비례적으로(Linear) 증가하는 것을 볼 수 있는데 이는 Round-Robin 스케줄링 특성에 기인한다. 즉, 캐시의 협동성을 고려하지 않고 요청 순서데로 캐시 서버를 할당하게 되면 성능은 캐서 서버에 수에 비례하여 증가하지만 이에 따라 캐시 합도 비례하여 증가하게 된다.

4.3.2 MD5 Hashing

<표 5>와 (그림 8)은 CD-A 구조에서 MD5 해싱 스케줄링 방식을 사용하여 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다. (그림 7)을 보면 호스트의 개수에 따라 성능이 비례적으로(Linear) 증가하지 않는 것을 볼 수 있는데 이는 MD5 해싱 스케줄링 특성에 기인한다. 즉, 해쉬의 특성상 사용자 전체 요청이 일부 캐시 서버로 몰리게 되고 이는 전체 서버의 성능이 일부 캐시 서버에 종속됨을 의미한다. 그러나 저장 공간의 측면에서 보면 동일 목적지 주소가 동일 서버에 할당됨으로 캐시간 협동을 유지하면서 캐시 공간의 합을 일정하게 유지함을 알 수 있다.

<표 6>과 (그림 9)는 MD5 Hashing 스케줄링 상황에서 Hot Spot(특정 목적지 주소로 요청이 몰림)을 가정하고 실험을 수행한 결과이다. 실험은 목적지 주소 255개를 반복 요청할 때 두 번 중에 한번은 동일 목적지 주소를 요청하도록 하였다. 실험 결과를 보면 Hot Spot(과부하)이 발생하지 않은 상황에 비해 성능이 크게 떨어지고 호스트 개수를 증가해도 성능이 향상되지 않음을 볼 수 있다. 이는 Hot Spot(과부하)이 발생하면 서버의 전체적인 성능이 요청이 몰리는 서버에 종속되기 때문이다.

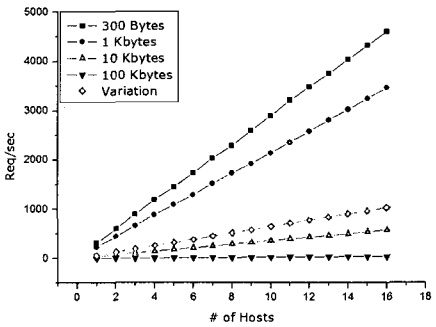
4.3.3 제안된 구조

<표 7>과 (그림 10)은 제안된 구조에서 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다. 제안된 구조는 CD-A의 구조적 장점(호스트 개수에 비례하여 성능이 향상됨)을 그대로 유지하면서 캐시간 협동성 및 Hot Spot(과부하)의 영향을 받지 않음을 알 수 있다. 왜냐하면 캐시 디렉토리를 공유함으로써 캐시 저장 공간을 동일하게 유지하면서(캐시간 협동성) 몰리는 요청을 서버들이 분담해서 처리(Hot Spot의 처리)할 수 있기 때문이다.

〈표 4〉 호스트 개수에 따른 초당 요청수 (RR)

# of Hosts	1	2	3	4	5	6	7	8
300 Bytes	316	596	895	1188	1446	1732	2025	2285
1 Kbytes	233	444	664	882	1090	1282	1516	1732
10 Kbytes	36	71	110	143	179	215	251	287
100 Kbytes	2.27	4.78	7.16	9.57	11.94	14.29	16.76	19.15
Variation	66	129	193	258	323	386	451	513

# of Hosts	9	10	11	12	13	14	15	16
300 Bytes	2598	2890	3212	3474	3750	4034	4315	4603
1 Kbytes	1921	2137	2353	2577	2799	3026	3244	3458
10 Kbytes	322	358	393	429	465	501	539	574
100 Kbytes	21.58	23.97	26.37	28.78	31.16	33.59	35.99	38.46
Variation	578	646	709	772	834	898	961	1026



(그림 7) 호스트 개수에 따른 초당 요청수 (RR)

〈표 5〉 호스트 개수에 따른 초당 요청수 (MD5 Hashing 스케줄링)

# of Hosts	1	2	3	4	5	6	7	8
300 Bytes	316	564	873	1069	1430	1589	1964	2013
1 Kbytes	233	415	636	771	1056	1160	1455	1489
10 Kbytes	36	67	104	123	172	191	231	232
100 Kbytes	2.15	4.92	7.15	9.54	11.93	13.85	16.38	18.32
Variation	65	117	181	211	299	336	414	421

# of Hosts	9	10	11	12	13	14	15	16
300 Bytes	2362	2470	2607	2908	3103	3305	3046	3694
1 Kbytes	1735	1809	1931	2148	2277	2390	2195	2703
10 Kbytes	277	297	311	342	373	388	345	434
100 Kbytes	20.99	22.49	24.20	26.71	29.68	30.03	32.20	32.16
Variation	486	517	544	608	662	684	608	766

〈표 6〉 호스트 개수에 따른 초당 요청수 (Hot Spot)

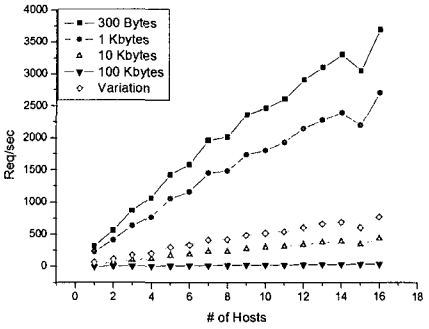
# of Hosts	1	2	3	4	5	6	7	8
300 Bytes	299	405	432	451	485	486	486	515
1 Kbytes	232	316	326	341	365	366	366	386
10 Kbytes	35	48	54	57	62	62	62	65
100 Kbytes	2.29	3.96	5.15	5.18	5.88	5.71	5.81	6.00
Variation	62	87	96	100	107	109	108	114

# of Hosts	9	10	11	12	13	14	15	16
300 Bytes	511	521	591	523	520	520	544	538
1 Kbytes	385	394	433	426	425	420	421	415
10 Kbytes	65	66	70	67	68	67	70	68
100 Kbytes	5.75	6.48	6.27	6.04	6.19	6.40	6.57	6.38
Variation	114	117	122	118	118	118	121	120

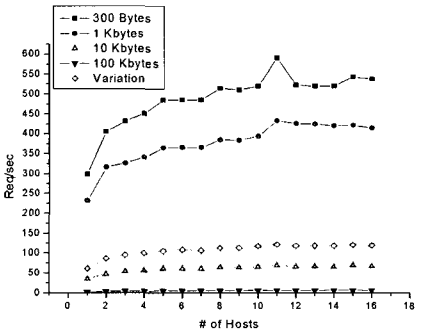
〈표 7〉 호스트 개수에 따른 초당 요청수 (공유 캐시 디렉토리)

# of Hosts	1	2	3	4	5	6	7	8
300 Bytes	310	539	749	962	1118	1297	1522	1709
1 Kbytes	225	406	587	763	882	1038	1218	1398
10 Kbytes	35	69	104	135	172	205	241	274
100 Kbytes	2.45	4.78	7.31	9.36	11.69	14.06	16.40	18.68
Variation	63	118	175	235	292	352	402	454

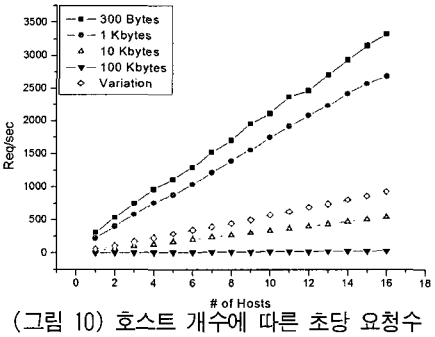
# of Hosts	9	10	11	12	13	14	15	16
300 Bytes	1961	2110	2378	2468	2709	2937	3156	3326
1 Kbytes	1561	1752	1920	2087	2234	2424	2575	2691
10 Kbytes	307	339	380	409	443	479	513	550
100 Kbytes	21.23	23.41	25.78	28.18	30.47	32.72	35.13	37.70
Variation	508	584	635	700	747	811	868	934



(그림 8) 호스트 개수에 따른 초당 요청수 (MD5 Hashing 스케줄링)



(그림 9) 호스트 개수에 따른 초당 요청수 (Hot Spot)



(그림 10) 호스트 개수에 따른 초당 요청수 (공유 캐시 디렉토리)

4.3.4 RR, MD5 Hashing vs. 제안된 구조

<표 9>는 제안된 구조와 기존 구조를 성능 과점에서 비교한 것이다. 제안된 방법은 RR 방법에 비해 -10.89%의 성능 감소를 가지고, 이미지 크기가 큰 경우 보다 작은 경우에 더 많은 성능 감소를 가진다. 전체적인 성능 감소는 RR 스케줄링 방식 자체가 캐시간 협동을 고려하지 않고 사용자의 요청을 모든 서버가 동일하게 처리하는 결과로 볼 수 있고, 이미지 크기가 작은 경우에 더 큰 성능 감소를 가지는 이유는 이미지 크기가 작을 수록 공유 캐시 디렉토리에 접근(액세스) 하는 시간이 상대적으로 많아서 발생하는 부하로 해석할 수 있다.

반면, 제안된 구조는 MD5 해쉬 스케줄링 방식에 비해 약간의 성능 향상(2.40%)을, Hot-Spot이 발생한 경우에는 2배 이상 높은 성능 향상(288.20%)을 가짐을 알 수 있다. 이는 MD5 해쉬 방식이 해쉬의 특성으로 일부 서버에 종속되고 Hot-Spot이 발생하는 경우에는 특정 서버로 전체 성능이 종속되는 것에 반해, 제안된 방법은 공유 캐시 디렉토리를 이용해서 캐시간 협동성을 가지면서 Hot-Spot을 효과적으로 처리함을 알 수 있다.

<표 10>은 기존 구조와 제안된 구조를 정성적으로 분석한 내용을 나타낸다. RR의 경우 확장성과 Hot-Spot을 처리할 수 있지만 캐시간 협동성이 없음을 알 수 있다. MD5 Hashing의 경우는 캐시간 협동성을 가지지만 해싱의 특성으로 인해 확장

〈표 9〉 성능 비교 (%) : 정량적 분석

%	300 bytes	1 Kbytes	10 Kbytes	100 Kbytes	Variation	Average
제안된 구조 vs. RR	-22.55	-16.93	-4.32	-1.12	-9.51	-10.89
제안된 구조 vs. MD5	-12.14	-3.98	13.99	4.27	9.87	2.40
제안된 구조 vs. MD5(Hot-Spot)	257.05	274.68	345.73	232.89	330.65	288.20

〈표 10〉 기존 구조 vs. 제안된 구조 : 정성적 분석

	RR	MD5 Hashing	제안된 구조
확장성(Scalability)	O	X	O
캐시간 협동성	X	O	O
Hot-Spot	O	X	O

성과 Hot-Spot을 처리하지 못함을 알 수 있다. 이에 반해, 제안된 구조는 공유 캐시 디렉토리로 인해 캐시간 협동성을 가지면서 확장성 및 Hot-Spot을 처리할 수 있음을 알 수 있다.

4.4 토론

제안된 구조가 기존 방법에 비해 좋은 성능을 가지는 이유는 공유 캐시 디렉토리로 설명할 수 있다. 제안된 구조는 서버들 중 하나의 서버 내 캐시 디렉토리를 모든 서버들이 공유함으로써 캐시간 협동성을 유지하면서 Hot-Spot을 처리할 수 있다.

제안된 구조의 단점은 공유 캐시 디렉토리를 사용함으로써 이로 인해 기존 RR 방식에 비해 성능이 떨어진다는 점이다. 이러한 성능 저하는 전체 서버가 하나의 캐시 디렉토리를 사용하고 이를 네트워크으로 접근하는데서 오는 부하를 의미한다. 그럼으로 본 논문에서 제안하는 구조는 네트워을 많이 사용하는 경우(Network intensive job)보다는 네트웍 부하가 적은 CPU를 더 많이 사용하는 경우(CPU intensive job)에 보다 적합하다.

5. 결 론

본 논문에서는 무선 인터넷 프록시 서버의 4가지 이슈(시스템적인 확장성, 단순한 구조, 캐시간 협동성, Hot Spot에 대한 처리)에 대해 기술하고, 기존 구조들이 가지는 문제점을 4가지 이슈 관점에서 분류하였다. 공유 캐시 디렉토리를 이용하는 제안된 구조는 기존 구조의 장점(시스템적인 확장성, 단순한 구조)을 유지하면서 캐시간 협동성 및 Hot Spot (과부하)을 효과적으로 처리할 수 있도록 하였다. 실험을 통해 제안된 구조가 기존 구조들에 비해 구조 개선 및 성능 향상에 기여함을 확인하였다.

향후 연구 방향을 요약하면 다음과 같다.

- 공유 캐시 디렉토리의 속도 개선 : 제안된 방식은 서버들 중 1개의 서버를 선택하고, 이 서버의 캐시 저장 디렉토리를 공유하여 다른 서버들이 이를 이용하도록 하였다. 즉, 1대의 캐시 서버를 사용하고 이를 네트웍을 통해 액세스함으로써 발생하는 속도 저하를 개선해야 한다.
- 네트워크 시뮬레이터를 이용한 실험 : 실제 무선 환경은 네트워크 링크 관점에서 좀더 신뢰도가 떨어지고, 단말 간 성능 및 대역폭 차이가 많이 발생한다. 이를 테스트하기 위해서는 신뢰도가 높은 네트워크 시뮬레이터를 이용한 실험이 필요하다.

참 고 문 헌

[1] A. Savant, N. Memon and T. Suel, "On the scalability of an image transcoding proxy server," International Conference on Image Processing, to appear, 2003.

[2] 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.

[3] 곽후근, 정규식, "무선 인터넷 프록시 서버 클러스터 성능 개선", 한국정보과학회논문지 : 정보통신, Vol.32, No.3, pp.415-426, 2005. 6.

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

[5] 곽후근, 우재용, 정윤재, 김동승, 정규식, "클러스터링 기반의 무선 인터넷 프록시 서버", 한국정보과학회논문지 : 정보통신, Vol.31, No.1, pp.101-111, 2004. 2.

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

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

[8] R. Malpani, J. Lorch, and D. Berger, "Making WWW caching servers cooperate," 4th International WWW conference, 1995.

[9] V. Valloppillil and K. Ross, "Cache array routing protocol v1.0," 1998.

[10] NFS(Network File System), http://www.faqs.org/rfcs/rfc1094.html.

[11] AB(Apache Bench), http://httpd.apache.org/docs-2.0/programs/ab.html.

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

[13] T. Lane, P. Gladstone and et. al., "The independent jpeg group's jpeg software release 6b.", ftp://ftp.uu.net/graphics/jpeg/jpegsrc.v6b.tar.gz.

[14] T. Kelly and J. Mogul, "Aliasing on the World Wide Web: Prevalence and Performance Implications", Proceedings of the 11th International World Wide Web Conference, pp. 281-292, 2002.

[15] S. Chandra, A. Gehani, C. Ellis and A. Vahdat, "Transcoding Characteristics of Web Images", Proceedings of the SPIE Multimedia Computing and Networking Conference, 2001.

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~2000년 7월 (주)3R 부설 연구소 주임연구원

2003년 6월~현재 (주) 펌킨넷코리아 선임연구원

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)

1998년 2월~1999년 2월 미국 IBM Almaden 연구소 방문연구원

1990년 9월~현재 숭실대학교 정보통신전자공학부 교수

관심분야 : 네트워크 컴퓨팅 및 보안