공유 스토리지 기반의 무선 인터넷 프록시 서버 클러스터

곽후근° 정규식
숭실대학교 정보통신전자공학부
{gobarian°, kchung}@q.ssu.ac.kr

A Shared Storage based Wireless Internet Proxy Server Cluster

Hukeun Kwak°, Kyusik Chung
School of Electronic Engineering, Soongsil University

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 Spot을 처리할 수 없다는 단점을 가진다.

이에 본 논문에서는 시스템적인 확장성, 단순한 구조, 캐시간 협동성, Hot Spot을 처리할 수 있는 공유 스토리지 기반의 무선 인터넷 프록시 서버 클러스터를 제안한다. 제안된 방법은 하나의 캐시 디렉토리를 공유하는 방법으로 기존 구조의 장점과 캐시간 협동성 및 Hot Spot을 처리할 수 있다는 장점을 가진다. 16대의 컴퓨터를 사용하여 실험을 수행하였고 실험 결과 Hot Spot 상황에서 제안된 방법이 높은 성능 향상을 가짐을 확인하였다.

1. 서론

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 있는 확장성이 요구되어지고 있다.

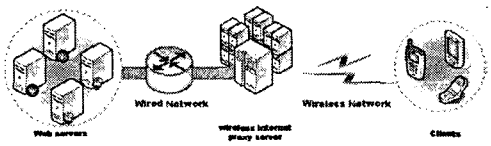


그림 1 무선 인터넷 프록시 서버

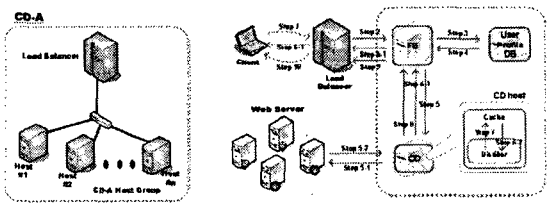
본 논문에서는 이러한 무선 인터넷의 문제점 및 요구들을 캐싱(Caching)[1], 압축(Distillation)[2] 및 클러스터링(Clustering)을 통하여 해결하는 방법으로 클러스터링 기반의 무선 인터넷 프록시 서버를 사용하였다. 그림 1은 무선 인터넷에 사용되는 무선 인터넷 프록시 서버를 나타내고, 이는 무선 사용자를 유선 인터넷 서버에 연결 시켜주는 역할을 한다.

본 연구자들은 기존 논문[3]에서 기존의 클러스터링 기반의 무선 인터넷 프록시 서버인 TranSend[4]를 개선한 CD-A라는 구조를 제안하였다. 본 논문에서는 CD-A 및 이의 개선 구조가 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

2.1 CD-A (CD & All-in-one)



(a) 구조 (b) 동작 과정

그림 2 CD-A 무선 인터넷 프록시 서버

(1) 구조

CD-A[3]는 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 & All-in-one[5]) LVS(Linux Virtual Server)[6]를 사용하여 부하 분산을 하는 것이다. 그림 2(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 위해서 이다. 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) 동작 과정

그림 2(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.

2.2 해쉬를 이용한 스케줄링

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 3은 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.
- 생성된 해쉬값을 캐수 개수(N)로 나머지 연산(Mod)한다.
- 나머지 연산을 통해 나온 값과 동일한 캐시로 요청을 할당한다.

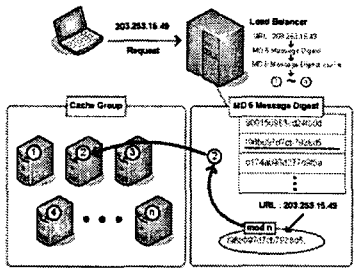


그림 3 해싱(MD5)을 이용한 스케줄링

2.3 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.

(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.

(2) 해쉬를 이용한 스케줄링의 문제점

- Hot Spot : 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 종속되어, 성능이 크게 감소한다.

(3) 본 연구의 접근 방식

본 논문에서는 캐시간 협동성을 가지면서 Hot Spot을 처리할 수 있는 새로운 구조를 제안한다.

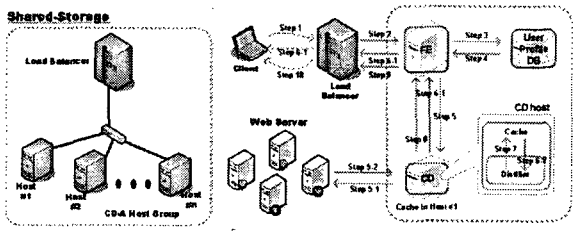
3. 제안된 구조

(1) 구조

그림 4(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 Spot이 발생해도 하나의 호스트에서 이를 처리하지 않고 모든 호스트가 분담해서 처리함으로 전체적인 성능이 Hot Spot을 처리하는 서버에 종속되지 않는다.

(2) 동작 과정

그림 4(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.



(a) 구조 (b) 동작 과정

그림 4 제안된 무선 인터넷 프록시 서버

(3) 기존 구조와의 비교 (CD-A, MD5 Hashing vs. 제안된 구조)

표 1은 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 및 방법

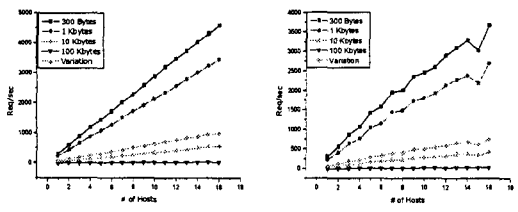
무선 인터넷 프록시 서버는 PC 16대로 구성하였고 TranSend 및 제안된 시스템에서 FE의 부하 분산과 All-in-one 시스템의 부하 분산을 위하여 LVS라는 Load Balancer를 사용하였다. Apache Bench라는 프로그램을 Client에서 수행하여 프록시 서버에 영상(이미지)를 요청하는 방식으로 실험하였다. 표에서 Client와 LVS가 Host보다 하드웨어 성능이 좋은 이유는 확장성 실험을 할 때 Client와 LVS에서는 병목이 발생하지 않는 조건에서 프록시 서버 내 호스트들 사이의 확장성을 확인하고자 했기 때문이다. 리눅스 상에서 호스트 1번의 캐시 디렉토리를 공유하기 위해 NFS(Network File System)[12]을 사용하였다. 1번 호스트가 NFS 서버를 실행하고 공유할 디렉토리를 exports하면, 다른 호스트는 이를 자신의 캐시 디렉토리에 mount하여 사용하는 방법으로 실험을 수행하였다.

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 이미지를 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. 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 실험을 할 때는 255개의 목적지 주소를 사용하되, 두 번 중에 한 번 은 동일 목적지 주소를 요청하게 하여 하나의 호스트로 요청이 몰리도록 하였다.

4.3 실험 결과

(1) CD-A

그림 5(a)는 CD-A 구조에서 Round-Robin 스케줄링 방식을 사용하여 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다. 그림 6을 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.



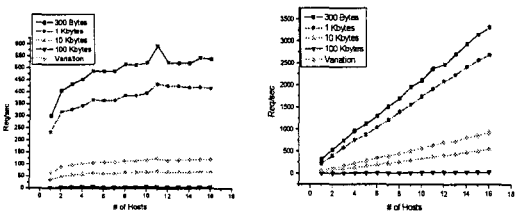
(a) CD-A (b) MD5 스케줄링

그림 5 호스트 개수에 따른 초당 요청수

(2) MD5 Hashing

그림 5(b)는 CD-A 구조에서 MD5 해싱 스케줄링 방식을 사용하여 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다. 그림 6을 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.

그림 6(a)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 Spot이 발생하지 않은 상황에 비해 성능이 크게 떨어지고 호스트 개수를 증가해도 성능이 향상되지 않음을 볼 수 있다. 이는 Hot Spot이 발생하면 서버의 전체적인 성능이 요청이 몰리는 서버에 종속되기 때문이다.



(a) Hot Spot (b) 공유 스토리지

그림 6 호스트 개수에 따른 초당 요청수

(3) 제안된 구조

그림 6(b)는 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 Spot의 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.

(4) CD-A, MD5 Hashing vs. 제안된 구조

표 2는 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 2 성능 비교 (%)

%	300 bytes	1 Kbytes	10 Kbytes	100 Kbytes	Vari.	Avg.
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

4.4 토론

제안된 구조가 기존 방법에 비해 좋은 성능을 가지는 이유는 공유 스토리지로 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 네트웍으로 접근하는데서 오는 부하를 의미한다.

5. 결론

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 스토리지를 이용하는 제안된 구조는 기존 구조의 장점(시스템적인 확장성, 단순한 구조)을 유지하면서 캐시간 협동성 및 Hot Spot을 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 스토리지의 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.

참고 문헌

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

[9] Virtual Server via DRouting, http://www.linuxvirtualserver.org/VS-DRouting.html.

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

[11] 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.

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