무선 인터넷 프록시 서버 클러스터 성능 개선
(A Performance Improvement Scheme for a Wireless Internet Proxy Server Cluster)

곽 후 근 † 정 규 식 ††
(Hukeun Kwak) (Kyusik Chung)

요 약 사회적으로 큰 관심의 대상이 되고 있는 무선 인터넷은 유선 인터넷과 달리 기술 환경과 그 특성상 여러 가지 제약점들을 가지고 있다. 대역폭이 낮고, 접속이 빈번하게 끊기며, 단말기내의 컴퓨팅 파워가 낮고 화면이 작다. 또한 사용자의 이동성 문제와 네트워크 프로토콜, 보안 등에서 아직 기술적으로 부족한 부분을 보이고 있다. 그리고 급속도로 증가하는 수요에 따라 무선 인터넷 서버는 대용량 트래픽을 처리할 수 있는 확장성이 요구되어지고 있다. 이에 본 논문에서는 무선 인터넷 프록시 서버 클러스터를 사용하여 앞에서 언급된 무선 인터넷의 문제와 요구들을 캐싱(Caching), 압축(Distillation) 및 클러스터(Clustering)를 통하여 해결하려고 한다. TranSend는 클러스터링 기반의 무선 인터넷 프록시 서버로 제안된 것이나 시스템적인(Systematic) 방법으로 확장성을 보장하지 못하고 불필요한 모듈간의 통신구조로 인해 복잡하다는 단점을 가진다. 기존 연구에서 시스템적인 방법으로 확장성을 보장하는 All-in-one 이라는 구조를 제안하였으나 이 역시 모듈간의 통신 구조가 복잡하고 캐시간 협동성이 없는 단점을 가진다. 이에 본 논문에서는 모듈간의 단순한 통신 구조와 캐시간 협동성을 가지는 클러스터링 기반의 무선 인터넷 프록시 서버를 제안한다. 16대의 컴퓨터를 사용하여 실험을 수행하였고 실험 결과 TranSend 시스템과 All-in-one 시스템에 비해 각각 54.86%, 4.70%의 성능 향상을 보였다. 캐시서버간 데이타를 공유할 수 있기 때문에 제안된 구조에서는 캐시서버 수에 무관하게 캐시 메모리 전체 크기를 일정하게 할 수 장점을 가진다. 반면에 All-in-one에서는 각 캐시서버가 모든 캐시 데이타를 가져야 하므로 캐시 메모리 전체 크기가 캐시 서버 수에 비례하여 증가한다.

키워드 : 무선 인터넷, 프록시 서버, 클러스터링, 확장성, 복잡성, 캐시간 협동성

Abstract Wireless internet, which becomes a hot social issue, has limitations due to the following characteristics, as different from wired internet. It has low bandwidth, frequent disconnection, low computing power, and small screen in user terminal. Also, it has technical issues to improve in terms of user mobility, network protocol, security, and etc. Wireless internet server should be scalable to handle a large scale traffic due to rapidly growing users. 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 above limitations and needs. TranSend was proposed as a clustering based wireless internet proxy server but it has disadvantages; 1) its scalability is difficult to achieve because there is no systematic way to do it and 2) its structure is complex because of the inefficient communication structure among modules. In our former research, we proposed the All-in-one structure which can be scalable in a systematic way but it also has disadvantages; 1) data sharing among cache servers is not allowed and 2) its communication structure among modules is complex. In this paper, we proposed its improved scheme which has an efficient communication structure among modules and allows data to be shared among cache servers. We performed experiments using 16 PCs and experimental results show 54.86% and 4.70% performance improvement of the proposed system compared to TranSend and All-in-one system respectively. Due to data sharing among cache servers,

· 본 연구는 숭실대학교 교내연구비지원으로 이루졌음
† 학생회원 : 숭실대학교 정보통신전자공학부
gobarian@q.ssu.ac.kr
†† 종신회원 : 숭실대학교 정보통신전자공학부 교수
kchung@q.ssu.ac.kr
논문접수 : 2004년 5월 24일
심사완료 : 2005년 1월 31일

the proposed scheme has an advantage of keeping a fixed size of the total cache memory regardless of cache server numbers. On the contrary, in All-in-one, the total cache memory size increases proportional to the number of cache servers since each cache server should keep all cache data, respectively.

Key words : Wireless internet, Proxy server, Clustering, Scalability, Complexity, Cache Cooperation

1. 서 론

현대 사회에서 정보에 대한 사람들의 관심은 갈수록 증가하고 있다. 정보에 대한 사람들의 욕구가 커지면서 기존의 유선 인터넷과 더불어 무선 이동 통신이 날로 각광받고 있다. 그래서 인터넷과 무선 이동 통신은 서로 조화되어 짧은 시간에 많은 발전을 이루었고 이제는 사람들의 생활 모습까지 변화시키고 있다. 언제 어디서든 인터넷을 통하여 정보를 검색할 뿐만 아니라 은행서비스와 전자 상거래까지 가능하게 되었다. 노트북, PDA, 핸드폰 등의 휴대용 단말기들의 사용이 점차 증가 되면서 이를 이용해 무선 인터넷에 접속하는 사람들이 늘어나고 있다. 이런 수요에 맞추어 여러 통신회사들은 더욱 많은 서비스를 늘리려고 앞 다퉈 경쟁하고 있으며, 기존의 ISP들 또한 무선 통신의 활용을 늘리기 위해서 장비와 시설을 더욱 확충하고 있는 등 무선 인터넷에 대한 관심은 사회적으로 큰 화제가 되고 있다.

현재 무선 인터넷의 사용이 증가하고 있지만 무선 인터넷의 본질적인 문제 역시 무시할 수 없는 요소로 부각되고 있다. 현재까지 나와 있는 무선 인터넷의 근본적인 문제점과 고려해 볼 수 있는 해결책은 표 1과 같다.

표 1 무선인터넷의 문제점과 해결책

문 제 점	해 결 책
낮은 대역폭	압축(Distillation)[1], 캐싱(Caching)
빈번하게 연결이 끊김	쿠키(Cookie), 캐싱(Caching)[2]
단말기내의 낮은 컴퓨팅 파워 및 작은 화면	압축(Distillation)
단말기 사용자의 이동성	Mobile IP[3], TCP Migration[4]
네트워크 프로토콜	Wireless TCP[5]
보안	프록시 서버[6, 7]

위에서 나열된 문제들을 캐싱(Caching)과 압축(Distillation)을 통하여 해결하는 방법으로 무선 프록시를 사용하며, 이의 기능으로는 캐싱과 압축을 비롯해 보안(Security)[6,7], 멀티미디어 전송[8,9], FEC(Forward Error Correction)[10], 핸드오프(Handoff)[11], 프로토콜 변환(Protocol Translation)[12], 오류 복구(Failure Recovery)[13] 등이 있다. 그림 1은 무선 인터넷에서 사용되고 있는 무선 프록시를 나타내고 있다.

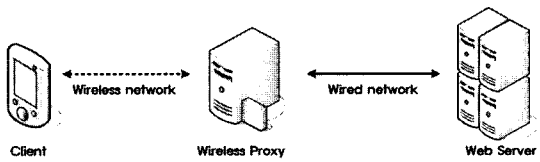


그림 1 무선 프록시

본 연구자들은 기존 논문[14]에서 기존 클러스터링의 기반의 무선 인터넷 프록시 서버인 TranSend[15]를 개선한 All-in-one 구조를 제안하였다. 본 논문에서는 TranSend와 All-in-one의 구조상의 문제점을 지적하고 이를 해결하는 새로운 구조를 제안하고 실험을 통해 이들을 비교하는데 초점을 맞춘다.

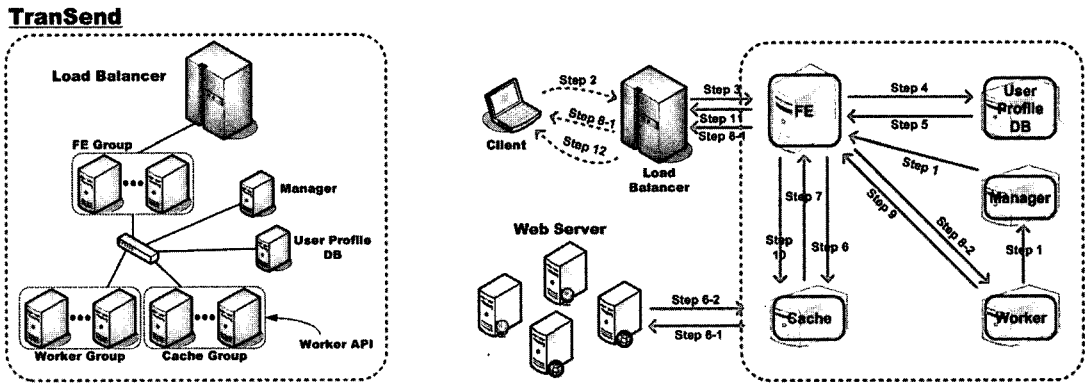
본 논문의 구조는 다음과 같다. 2장에서는 기존 무선 프록시와 이들이 가지는 문제점을 소개한다. 3장에서는 기존 무선 프록시가 가지는 문제점들을 해결하는 제안된 구조를 설명하고, 4장에서는 실험 및 토론을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 연구 배경

2.1 기존 무선 인터넷 프록시 서버

무선 프록시는 캐싱(Caching), 압축(Distillation), 대용량 트래픽에 대한 확장성(Scalability)을 고려하여야 한다. 기존 무선 프록시들[14-22]은 기본적으로 캐싱(Caching)과 압축(Distillation) 기능을 제공하고 있다. 그러나 확장성의 관점과 구조적인 관점에서는 무선 프록시들 간의 개별적 특성에 따라 차이점을 가지고 있다. 먼저, 확장성 관점에서는 일부[14,15,21]는 고려하고 있는 반면에 대부분은 고려하고 있지 않다. 구조적 관점에서도 캐싱과 압축 기능이 대부분 하나의 호스트에 통합(Unity)되어 있지만, 일부[15,16]는 다른 호스트로 분리(Separation)되어 있다.

기존 무선 프록시로는 All-in-one[14], TranSend[15], Mowgli(Mobile Office Workstations using GSM Links)[16], WebExpress[17], Class-based Proxy[18], KWU Proxy[19], Distributed Proxy[20], TranSquid[21,22] 등이 있다. 이중 TranSend는 대용량 트래픽에 대한 확



(a) 구조 (b) 동작 과정

그림 2 TranSend 프록시 시스템

표 2 TranSend의 동작 과정

과정	설명
1	Distiller는 주기적으로 부하 정보를 Manager에게 보내고, Manager는 이 부하 정보의 평균을 FE에게 보낸다.
2	사용자는 부하 분산기(LB: Load Balancer)에게 데이타를 요청한다.
3	부하 분산기는 FE에게 데이타 요청을 보낸다.
4	FE는 User Profile DB에게 사용자 정보(Preference)를 요청한다.
5	User Profile DB는 사용자 정보를 FE에게 보낸다.
6	FE는 캐시에게 사용자 요청을 보낸다.(이 캐시는 사용자 요청 URL의 MD5 Hash 값에 의해 FE가 선택한 것이다.) 6-1. 캐시안에 요청 데이타가 없다면, 외부 웹서버에 데이타를 요청한다. 6-2. 웹서버는 데이타를 캐시에 보내고, 캐시는 이를 저장한다.
7	캐시는 FE에게 요청 데이타를 보낸다.
8	FE는 캐시로부터 받은 데이타를 확인한다. 8-1. 데이타가 압축된 형태라면, 사용자에게 데이타를 보낸다. 8-2. 데이타가 압축된 형태가 아니라면, FE는 Distiller에게 캐시로부터 받은 데이타와 사용자 정보를 보내 압축을 요청한다.(이 Distiller는 Manager로부터 받은 Distiller로 부하 정보를 이용하여 FE에 의해 선택된 것이다.)
9	Distiller는 데이타를 압축하고, FE에게 압축된 데이타를 보낸다.
10	FE는 캐시에 압축된 데이타를 저장한다.
11	FE는 부하 분산기로 압축된 데이타를 보낸다.
12	부하 분산기는 사용자 요청에 응답한다.

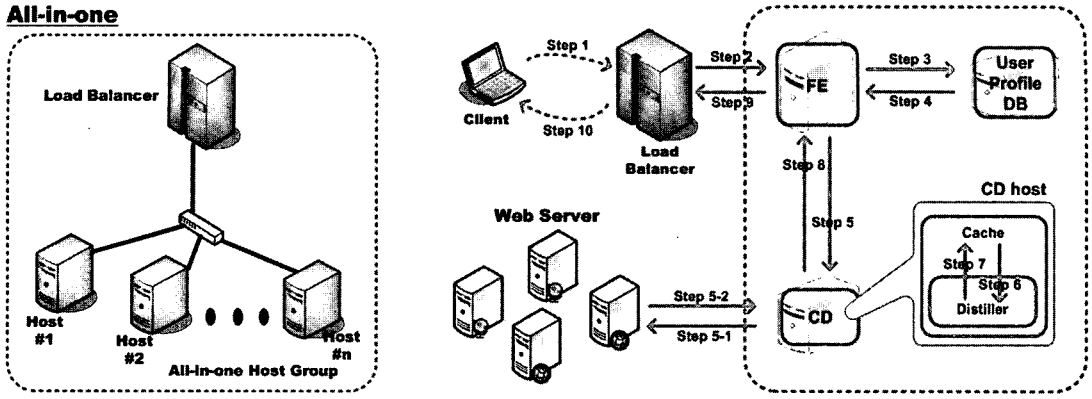
장성을 고려하여 클러스터링으로 구현된 무선 프록시로서 본 논문에서는 이를 기반으로 새로운 구조를 제안한다.

2.2 TranSend

(1) 구조

그림 2(a)는 TranSend 프록시 시스템의 전체적인 구조를 나타낸다. Transend는 각 모듈로써 Front End (FE), User Profile DB, Cache($), Worker, Manager로 구성된다. 모듈 별 구체적인 기능은 다음과 같다. 더 자세한 내용에 대해서는 참고 문헌[15]을 참조하시오.

- FE는 외부의 HTTP 인터페이스를 제공하고 사용자 요청에 대한 전체적인 관리의 기능을 하고 있다. 400개의 쓰레드(Thread)로 구성되며, 사용자 요청을 병렬적으로 처리한다. User Profile DB를 통하여 사용자를 관리(Customization)하고, Manager로부터 주기적으로 신호(beacon)가 오지 않을 경우 새로운 Manager를 실행함으로 오류를 복구(fault tolerance)하는 기능을 가진다.
- TranSend는 사용자가 자신의 환경을 고려하여 압축(Distillation) 수준 등을 설정 할 수 있는 사용자 정보(Preference) 설정 페이지를 제공한다. 이 페이지는 사용자가 프록시 자체의 주소(URL)를 요청하면 나타나고 설정이 끝나면 GDBM(GNU Database Manager) 형태로 저장된다. FE는 이 GDBM으로부터 사용자 정보를 얻어와 Distiller에게 Cache로부터 얻은 사용자 요청 데이타와 함께 넘겨준다.
- TranSend 시스템은 4개의 Harvest Cache[23] 노드(Node)로 구성되어 있다. 4개의 Cache 노드를 효율적으로 관리하기 위하여 Manager Stub에서는 사용자가



(a) 구조 (b) 동작 과정

그림 3 All-in-one 프록시 시스템

요청한 주소(URL)에 대해 MD5 Hash[24]를 적용하여 Cache를 선택하도록 한다. MD5 Hash를 사용한 이유는 동일 주소(URL)에 대해 동일 Cache가 선택되도록 하기 위함이다. 또한 Harvest Cache에 직접 데이타를 저장할 수 있게 수정하여 실제 웹 서버에서 가지고 오는 데이타뿐만 아니라 Distiller에서 압축(Distillation)한 데이타 역시 Cache에 저장할 수 있도록 하는 구조로 되어 있다.

- Worker(Datatype-Specific Distiller)는 두 가지 기능을 수행하는데, 하나는 압축 지연 시간(Distillation Latency)을 계산하는 것이고 나머지는 실제 압축을 수행하는 것이다. FE로부터 일이 주어지면 큐(Queue)에 쌓이고 FIFO, Priority, Lottery 중 하나의 방식으로 일을 처리한다. Priority나 Lottery의 사용 이유는 사용자의 IP를 기준으로 일의 우선 순위를 다르게 하기 위해서다.
- TranSend는 중앙 집중 Manager를 사용한다. 분산 Manager에 비해 중앙 집중 Manager가 갖는 장점은 부하 분산 방식을 쉽게 바꿀 수 있다는 점이고 단점은 시스템상의 병목 및 단일 장애점(a single point of failure)이 될 가능성이 존재한다는 점이다.

(2) 동작 과정

FE는 부하 분산기(Load Balancer)를 통해 사용자 요청을 받고 이를 캐시로 보낸다. 요청 데이타가 캐시에 없다면 캐시는 이 데이타를 웹 서버에 요청한다. 캐시가 데이타를 받고 이는 다시 FE로 보내진다. 압축이 필요하다면, FE는 캐시에서 받은 데이타를 Distiller로 보낸다. FE는 Distiller로부터 받은 압축된 데이타를 캐시에 저장하고 사용자에게 응답한다. 그림 2(b)와 표 2는 TranSend의 구체적인 동작 과정을 나타낸다.

2.3 All-in-one

(1) 구조

All-in-one은 TranSend에 사용된 모듈들을 모두 하나의 호스트에 넣고(이하 All-in-one) 이 호스트들을 LVS(Linux Virtual Server)[25]를 사용하여 부하 분산을 하는 것이고 그림 3(a)는 이를 나타낸다. TranSend에서는 각각의 모듈들(FE, Cache, Distiller) 각각이 클러스터링 되어 있는 반면에 All-in-one에서는 각 모듈들을 하나의 호스트들에 포함하고 이러한 호스트를 클러스터링하는 구조로 되어 있다.

TranSend에 사용된 모듈들을 모두 하나의 호스트에 넣은 이유는 시스템적으로 확장하는 구조를 만들기 위해서이다. 즉, TranSend는 새로운 모듈을 추가 시에 동작과정중의 병목을 찾아 그 모듈을 추가해야하는(No Systematic) 반면에, All-in-one은 병목에 상관없이 새로운 호스트를 추가하면(Systematic) 그 호스트 내에 모듈 중에 필요한 모듈이 상대적으로 많이 사용된다.

(2) 동작 과정

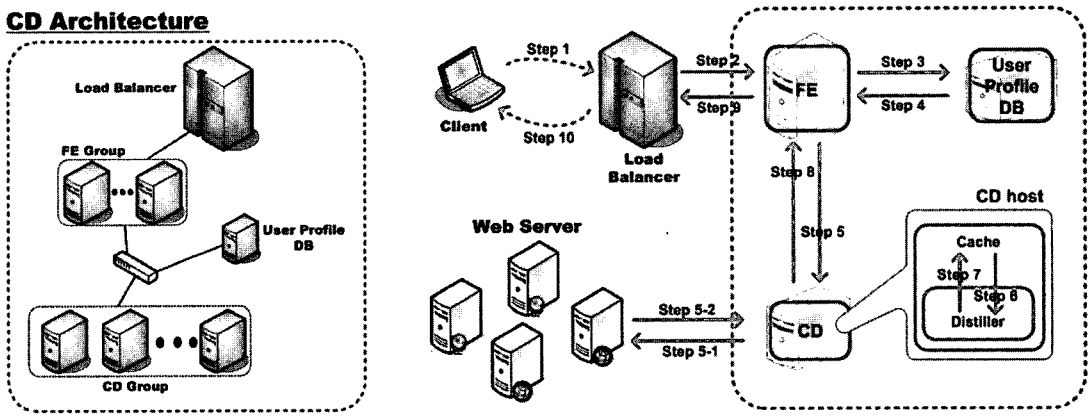
그림 3(b)는 All-in-one 프록시 시스템의 구체적인 동작 과정을 나타낸다. 이 동작과정은 TranSend과 거의 동일하다. 차이점은 다음과 같다. TranSend에서는 FE에서 Distiller로 요청한다. Distiller가 여러 개 있으면 그 중 하나를 선택하는 과정이 추가된다. 또한 FE에서 Cache로 요청할 때 Cache가 여러 개 있으면 그 중 하나를 선택하는 과정이 추가된다.

2.4 접근 방식

본 절에서는 TranSend 무선 프록시 서버 및 이의 개선 구조인 All-in-one의 문제점을 정리하고, 3장에서는 이를 해결할 새로운 무선 프록시 구조를 제안한다.

(1) TranSend 구조의 문제점

- 확장성(Scalability) : 그림 2 구조에서 보면 FE, Cache, Distiller는 각각 여러 개의 노드(Node)들로



(a) 구조 (b) 동작 과정

그림 4 CD 프록시 시스템

표 3 CD 동작 과정

과정	설명
1	사용자는 부하 분산기(LB: Load Balancer)에게 데이타를 요청한다.
2	부하 분산기는 FE에게 데이타 요청을 보낸다. (이 FE는 부하 분산기에서 라운드 로빈 방식에 의해 선택된 것이다.)
3	FE는 User Profile DB에게 사용자 정보(Preference)를 요청한다.
4	User Profile DB는 사용자 정보를 FE에게 보낸다.
5	FE는 CD에게 사용자 요청을 보낸다. (이 CD는 사용자 요청 URL의 MD5 Hash 값에 의해 FE가 선택한 것이다.) 6-1. CD 안에 요청 데이타가 없다면, 외부 웹서버에 데이타를 요청한다. 6-2. 웹서버는 데이타를 CD에 보내고, CD는 이를 저장한다.
6	CD는 데이타를 압축한다.
7	CD는 압축된 데이타를 저장한다.
8	CD는 압축된 데이타를 FE에게 보낸다.
9	FE는 부하 분산기로 압축된 데이타를 보낸다.
10	부하 분산기는 사용자 요청에 응답한다.

구성가능하다. 프록시 서버를 확장성 있게 만들려면 노드들을 추가해야하지만 어느 분류(FE 그룹, Cache 그룹, Distiller 그룹)의 노드들을 언제 추가해야 하는지 시스템적인(Systematic) 방법이 없다. 즉, 실험 결과에 의존하여 특정 모듈 그룹이 병목 현상이 발생하면 그룹 모듈을 추가하는 방식으로 하게 된다. 기존 연구[26]에서는 이를 해결하기위해 All-in-one이라는 구조를 제안하였다.

- 복잡성(Complexity) : TranSend는 FE, Cache, Distiller로 구성되어 FE를 중심으로 서로 간에 통신(Communication)을 하도록 구성되어 있다. 이러한 구조는 FE로 모든 통신이 편중되어 있고 Cache와 Distiller가 분리되어 있어 복잡한 통신을 하는 단점을 가진다.

2.4.2 All-in-one 구조의 문제점

- 캐시간 협동성(Cache Cooperation) : All-in-one 구조는 Cache간의 협동(Cooperation)이 없어 다른 호스트에 사용자 요청과 동일한 데이타가 Cache되어 있어도 매번 실제 웹 서버로 요청하고 이를 자신의 로컬 Cache에 두어서 Cache된 데이타의 합이 커지는 문제가 발생한다.
- 복잡성(Complexity) : All-in-one 구조는 TranSend를 구성하는 모듈들을 하나의 호스트에 포함한 구조임으로 TranSend가 가지는 복잡성의 문제를 가진다.

(3) 본 연구의 접근 방식

본 논문에서는 TranSend 및 All-in-one 구조의 단점인 복잡성(Complexity)을 단순화(Simplification)하고 All-in-one의 단점인 캐시간 협동(Cache Cooperation)이 안 되는 문제를 해결하는 새로운 구조를 제안한다.

3. 제안된 프록시 서버

3.1 구조

그림 4(a)는 2.4절에서 분석된 TranSend 및 All-in-one 무선 프록시의 문제점을 기반으로 이를 해결할 수 있도록 제안된 구조이다. 제안된 구조에서는 TranSend에 사용된 기본 모듈에서 Distiller를 없애고 Cache에 압축

(Distillation) 기능을 추가(이하 CD: Cache & Distiller)한 것이다. TranSend에서는 모듈들(FE, Cache, Distiller) 각각이 클러스터링 되어 있고, All-in-one은 All-in-one 호스트가 클러스터링 되어 있는 반면에 제안된 구조에서는 Distiller를 제외한 모듈들(FE, CD) 각각이 클러스터링 되어 있다.

제안된 구조에서 Cache에 압축 기능을 추가하고 이를 FE와 따로 분리한 이유는 다음과 같다. Cache에 압축 기능을 추가한 이유는 전체 구조에서 Distiller를 제거하여 불필요하고 복잡한 통신 구조를 단순화하기 위해서이고 FE를 따로 분리한 이유는 FE에서 Cache를 선택할 때 MD5 Hash를 사용하여 Cache간 협동이 가능하게 하기위해서이다.

(2) 동작 과정

그림 4(b)와 표 3은 CD의 구체적인 동작 과정을 나타낸다.

(3) TranSend vs. All-in-one & CD

표 4는 TranSend와 All-in-one을 제안된 시스템(CD)과 구조적으로 비교한 표이다.

표 4 기존 구조 vs. 제안된 구조(CD)

	TranSend	All-in-one	제안된 구조(CD)
확장성 (Scalability)	No Systematic	Systematic	No Systematic
단순화 (Simplification)	X	X	O
캐시간 협동 (Cache Cooperation)	O	X	O

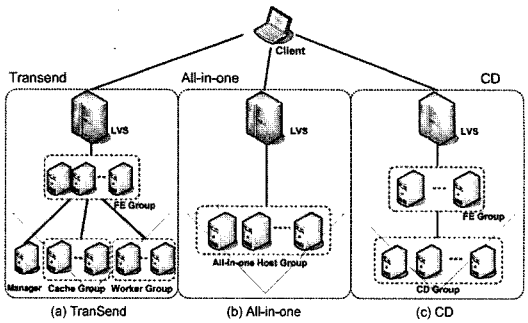


그림 5 실험에 사용된 구성도

4. 실험 및 토론

4.1 실험 환경

표 5는 실험에 사용된 하드웨어와 소프트웨어를 나타낸다. 프록시 서버는 PC 16대로 구성되어 있고 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 상황에서 프록시 서버내 호스트들 사이의 확장성을 확인하고자 했기 때문이다.

그림 5는 실험에 사용된 구조들의 구성도를 나타낸다. 그림 5(a)와 5(c)는 All-in-one과 확장성 측면에서 비교하기위해, TranSend 및 CD 기본 구조에 FE를 클러스

표 5 실험용 하드웨어 & 소프트웨어

		하드웨어		소프트웨어	개수
		CPU (Hz)	RAM (MB)		
Client		P-III 700 M	128	AB[26]	1
LVS		P-IV 2.4 G	512	NAT[27]	1
Host	Cache	P-II 400 M	256	Squid[28]	16
	Distiller			JPEG-6b[29]	

표 6 실험에 사용된 변수

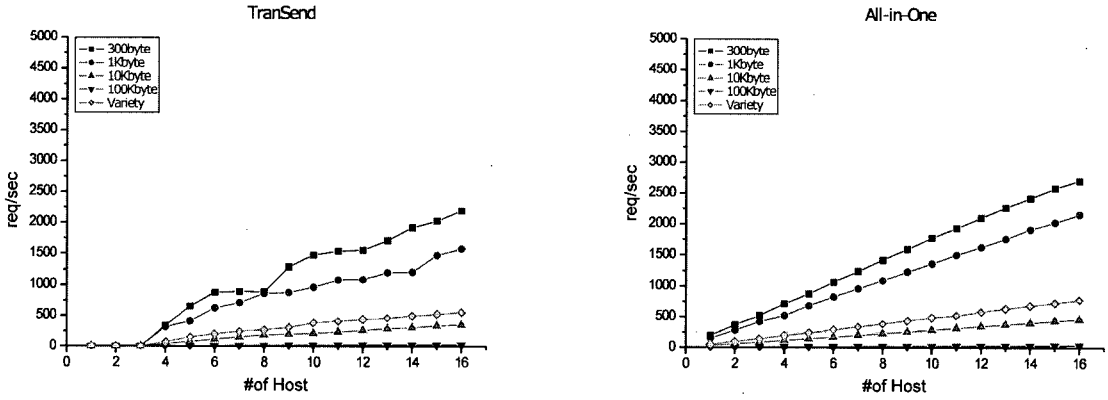
사용자의 요청 개수	• 약 200초 동안 프록시가 처리할 수 있는 최대 개수
요청 이미지	• JPEG[30]
요청 크기	• 300 bytes, 1 K, 10 K, 100 Kbytes, Variation[31]
요청 방식	• 같은 크기 : 다른 이름으로 랜덤(Random)하게 요청 (Cache들 사이에서 MD5 Hash를 적용하기 위해) • Variation : 1K-10K 사이의 이미지를 랜덤하게 요청
사용자 정보(Preference)	• 이미지 Quality = 중간
웹 서버	• Cache 서버 자체에 둠 (프록시내의 성능 평가에 초점을 맞춤)
병목(bottleneck)	• 호스트의 CPU 점유율 중 가장 높은 호스트 (본 실험에서는 ethernet이나 system bus 병목은 발생하지 않는다.)

표 7 각 구조의 실험 방법

	TranSend	All-in-one	CD
1	기본 TranSend 시스템을 구성한다. (Host 1 : Manager/Monitor, Host 2 : FE1, Host 3 : Cache1, Host 4 : Distiller1)	기본 All-in-one 시스템을 구성한다. (Host 1)	기본 CD 시스템을 구성한다. (Host 1 : FE, Host 2 : CD)
2	AB(Apache Bench)를 이용하여 TranSend로 JPEG 이미지를 약 200초 동안 프록시가 처리할 수 있는 최대 개수로 요청한다.	TranSend Step 2와 동일	TranSend Step 2와 동일
3	초당 요청 개수 및 각 Host의 CPU 점유율을 측정하여 병목 Host를 알아낸다.	초당 요청 개수를 측정한다.	TranSend Step 3과 동일
4	병목 Host를 추가한다.	All-in-one Host를 추가한다.	TranSend Step 4와 동일
5	실험에 사용된 Host 수(16대)만큼 2)-4)를 반복한다.	실험에 사용된 Host 수(16대)만큼 2)-4)을 반복한다.	TranSend Step 5와 동일

표 8 호스트 개수에 따른 초당 요청수 (CD)

# of Hosts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	0	487	622	960	1235	1430	1776	1823	2141	2356	2647	2673	3078	3110	3139	3163
1 Kbytes	0	371	483	648	909	952	1129	1378	1560	1609	1515	1784	1880	2290	2460	2356
10 Kbytes	0	40	79	110	153	187	195	241	275	238	295	287	316	289	333	370
100 Kbytes	0	2.43	5.24	7.3	10	11.74	14.22	15.86	17.34	19.86	20.8	18.43	23.71	25.48	26.5	26.38
Variation	0	74	130	201	253	330	366	396	388	385	410	553	570	608	399	664



(a) TranSend (b) All-in-one

그림 6 호스트 개수에 따른 초당 요청수 (TranSend & All-in-one)

터링하도록 LVS를 추가하였다.

4.2 실험 방법

표 6은 실험에 사용된 변수들을 정리한 것이고 각 구조의 실험 방법을 정리하면 표 7과 같다. Cache 클러스터링은 요청된 이미지의 서로 다른 이름으로 MD5 Hash 값을 얻어 실험을 수행하였다.

4.3 실험 결과

(1) TranSend & All-in-one

그림 6(a)는 TranSend 구조에서 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타내고 그림 6(b)는 All-in-one 구조에서 이미지 크기를 다르게 요청 했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다.

(2) CD

표 8과 그림 7은 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다. 예를 들면, 표 8에서 이미지 크기가 300 bytes이고 호스트의 개수가 5개일 때, 5개의 구성은 표 9(b)에 나와 있고 (FE : 3대, CD : 2대), 초당 요청수는 1235 requests/sec가 나왔다.

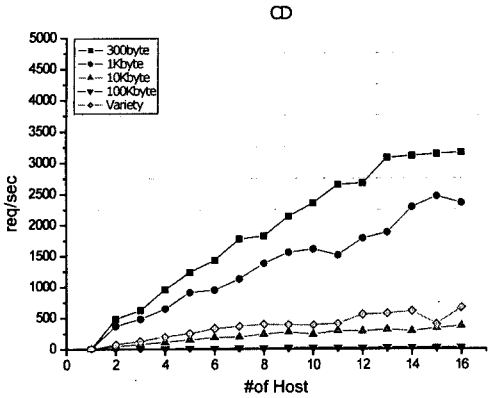


그림 7 호스트 개수에 따른 초당 요청수 (CD)

그림 7을 보면 이미지 크기가 클 경우 호스트 숫자에 비례하여 성능이 향상되나 이미지 크기가 작을 경우 호스트 숫자에 비례하여 성능이 향상되지 않음을 알 수 있다. 이는 표 9에서 보여지듯이 이미지 크기가 클 경우 CD에 집중되어 병목이 발생함으로 CD만 추가하면 성능이 향상된다. 그러나 이미지 크기가 작으면 병목이 모듈간(FE, CD)에 골고루 분포하게 되어 이를 모두 해결해야만 성능 향상이 발생함을 알 수 있다.

또한 호스트를 추가하여도 초당 요청수가 증가하지 않고 감소되는 경우를 볼 수 있다. 이는 캐시(CD)를 클러스터링하기위해 FE에서 사용된 MD5 Hash의 side-effect 이다. 즉, 요청된 이미지의 이름에 따라 MD5 Hash를 사용하여 캐시를 선택하게 될 때 요청이 일부의 CD로 집중되는 경우가 발생하고 전체의 성능(초당 요청수)은 이러한 일부 CD에 종속됨으로 호스트를 추가하여도 초당 요청수가 떨어지는 현상이 발생하였다. 그러나 전체적으로 호스트를 계속 추가하면 초당 요청수가 계속 증가함으로 확장성 관점에서는 문제가 없음을 확인하였다.

표 9은 호스트 개수에 따른 CPU 점유율 및 모듈들의 분포를 나타낸다. 표 9(a)에서 보면 호스트 개수가 2개일 경우 FE 1대, CD 1대로 배정되어 수행하였다. CPU 점유율이 가장 높은 호스트를 다음번 호스트 추가시 배정하는 원칙을 적용하였다. 호스트가 4대일 경우 CD2의 CPU 사용률이 98.5%이었음으로 5번째 호스트는 CD로 배정하였다. 표 9(a)를 기준으로 # of Hosts = 4일 때 병목난에 CD2가 적혀있고 추가에 CD(CD3)가 적혀있다. 가장 높은 호스트가 2개 존재하면 사용자 요청 처리

표 9 호스트 개수에 따른 CPU 점유율 및 모듈들의 분포 (CD)

(a) CPU 점유율 (300 bytes)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
# of Host	FE1	CD1	FE2	CD2	FE3	CD3	FE4	FE5	CD4	CD5	FE6	FE7	CD6	FE8	CD7	CD8	병목	추가
2	100	83.4															FE1	FE2
3	85	100	87.4														CD1	CD2
4	99.5	72.2	98.8	88.2													FE1	FE3
5	95.8	97.1	96.1	98.5	96.3												CD2	CD3
6	99.5	78.5	99.5	74.7	99.8	78.3											FE3	FE4
7	98.4	97.1	97.9	88.6	98.2	92.3	96										FE1	FE5
8	91.2	100	91.5	95.1	90.6	97.9	87.9	89.5									CD1	CD4
9	97.7	72	97.8	100	98	94.5	95.8	98.3	70.6								CD2	CD5
10	99.6	77.8	99.7	84.2	99.7	61.5	98.5	99.4	59.7	87.6							FE2	FE6
11	98.1	88.6	98.6	93.1	98.4	66.1	96.2	100	67.8	96.7	98.2						FE5	FE7
12	94.4	89.2	95.1	98.7	93.8	71	93	94.1	62.3	94.1	94	92.1					CD2	CD6
13	97.4	67.6	98.5	76.1	98.2	91.6	96.8	98	94.9	78.3	99.9	99.5	67.4				FE6	FE8
14	95.1	73	94.2	77.3	94.8	96.9	91.9	94.2	100	78.5	99.2	98.3	70.2	99.4			CD4	CD7
15	94.6	91.2	95	100	94.4	81.7	93.1	99.5	77.7	67	98.6	97.7	55.7	98.8	99.5		CD2	CD8
16	95.2	60.9	94.7	79.3	93.4	58.4	93.2	98.9	58.9	70.8	98.1	97.2	66.7	98.3	100	97.9	CD7	

(b) 모듈들의 분포

# of Hosts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	FE #	CD #
300 bytes	FE1	CD1	FE2	CD2	FE3	CD3	FE4	FE5	CD4	CD5	FE6	FE7	CD6	FE8	CD7	FE9	9	7
1 Kbytes	FE1	CD1	CD2	FE2	CD3	FE3	CD4	CD5	FE4	CD6	CD7	CD8	CD9	FE5	FE6	CD10	6	10
10 Kbytes	FE1	CD1	CD2	CD3	CD4	CD5	CD6	CD7	CD8	CD9	CD10	CD11	CD12	CD13	CD14	CD15	1	15
100 Kbytes	FE1	CD1	CD2	CD3	CD4	CD5	CD6	CD7	CD8	CD9	CD10	CD11	CD12	CD13	CD14	CD15	1	15
Variation	FE1	CD1	CD2	CD3	CD4	CD5	CD6	CD7	CD8	CD9	CD10	FE2	CD11	CD12	CD13	CD14	2	14

표 10 평균 성능 향상률 (CD)

%	300 bytes	1 Kbytes	10 Kbytes	100 Kbytes	Variation	Average
TranSend vs. CD	81.03	70.21	37.98	46.51	38.58	54.86
All-in-one vs. CD	32.05	17.71	-8.02	-10.42	-7.84	4.70

표 11 사용자 요청(10,000개의 서로 다른 요청)에 따른 캐시의 합
(여기서, K = 캐시 데이타의 합, N = 프록시 서버 대수)

	캐시 합	비고
TranSend	K 개	FE에서 캐시로 부하 분산 시에 MD5 해쉬를 사용하여 각 클라이언트 요청마다 해당 캐시를 찾도록 함
All-in-one	N*K 개 (N 대 x K 개)	LVS에서 FE로 부하 분산 시에 라운드 로빈을 사용함으로 다른 캐시에 요청 데이타가 있어도 이를 사용하지 않고 자신의 캐시를 사용함으로 캐시의 합을 증가시킴
CD	K 개	FE에서 캐시로 부하 분산 시에 MD5 해쉬를 사용하여 각 클라이언트 요청마다 해당 캐시를 찾도록 함

표 12 캐시간 협동성과 구조적 복잡성에 따른 분류

Cache Cooperation	Complexity	TranSend	All-in-one
X (Round Robin)	Complex	TranSend-RR	All-in-one
O (MD5 Hash)	Complex	TranSend	N/A
X (Round Robin)	Simple	CD-RR	CD-A
O (MD5 Hash)	Simple	CD	N/A

* N/A : Not Available

순서에 우선하는 호스트를 추가하였다(FE > CD).

표 9(a)는 이미지 크기가 300 bytes일 경우 실험이고 이미지 크기가 1K, 10K, 100K, Variation이 될 경우 같은 방식으로 호스트 숫자를 증가하면서 실험을 수행하였다. 표 11(b)의 맨 오른쪽 옆 FE #, CD #를 보면 이미지 크기에 따라 사용한 FE, CD 숫자가 각각 다르다는 것을 알 수 있다. 표에서 보여지듯이 이미지 크기가 작으면 FE와 CD의 호스트 개수가 증가하고, 이미지 크기가 크면 CD의 호스트 개수가 증가함을 알 수 있다.

(3) TranSend, All-in-one vs. CD

표 10는 TranSend와 All-in-one에 대한 CD의 평균 성능 향상률을 나타낸 것이다. 제안된 시스템(CD)은 TranSend와 All-in-one에 비해 각각 평균 54.86%와 4.70%의 성능 향상을 가진다. 작은 크기의 이미지는 큰 크기의 이미지보다 상대적으로 모듈간의 통신에서 많은 시간이 소요됨으로 제안된 시스템에서 높은 성능 향상률을 가짐을 알 수 있다. 그리고 TranSend에 비해 All-in-one의 성능 향상률이 낮은 이유는 All-in-one의 CPU 활용도가 다른 시스템에 비해 높아 초당 요청수가 상대적으로 높기 때문이다.

표 11은 각 구조에서 필요한 캐시의 합을 나타낸다. 사용자가 K개의 서로 다른 데이타를 계속 요청하였을 때, TranSend와 CD 구조를 가지는 N대의 프록시는 서로의 캐시를 공유함으로 총 K개의 데이타에 대한 캐시가 필요하다. 그러나 All-in-one 구조를 가지는 N대의 프록시는 서로의 캐시를 공유하지 않음으로 각 프록시마다 K개의 캐시를 가지게 되어 총 N*K개의 캐시를 가지게 됨을 알 수 있다. 즉, 사용자의 요청이 다양해질수록 All-in-one 구조가 가지는 캐시의 합이 커짐을 알 수 있다.

(4) CD-RR & CD-A

TranSend와 All-in-one을 캐시간 협동성과 구조의 복잡성 관점에서 분류하면 표 12와 같다.

TranSend-RR은 TranSend에서 캐시를 선택할 때 MD5 Hash를 사용하지 않고 RR(Round Robin)을 사용하여 캐시간 협동을 하지 않은 경우이다. 이 구조에서는 Distiller와 Cache가 분리되어 있어 사용된 캐시의 수가 적음을 알 수 있다. 실험을 통해서 캐시의 수가 적을 때에는 MD5 Hash와 RR이 크게 차이가 없음을 확인하였고 이에 TranSend-RR에 대한 실험을 수행하지 않았다.

CD-RR은 CD에서 캐시를 선택할 때 MD5 Hash를 선택하지 않고 RR을 사용하여 캐시간 협동을 하지 않은 경우이다. 이 경우에서는 Distiller가 따로 존재하지 않고 Cache에 통합되어 있어 사용된 캐시의 수가 많음을 알 수 있다. 실험을 통해서 캐시의 수가 많을 때에는 MD5 Hash와 RR이 크게 차이가 남을 확인하였고 그림 8과 표 13은 CD-RR에서 초당 요청수 및 평균 성능 향

표 13 평균 성능 향상률 (CD-RR)

%	300 bytes	1 Kbytes	10 Kbytes	100 Kbytes	Variation	Average
TranSend vs. CD-RR	93.44	105.29	74.42	64.55	81.46	83.83
All-in-one vs. CD-RR	38.47	39.76	13.89	-0.63	18.33	21.96

표 14 평균 성능 향상률 (CD-A)

%	300 bytes	1 Kbytes	10 Kbytes	100 Kbytes	Variation	Average
TranSend vs. CD-A	125.02	130.84	88.76	87.30	96.53	105.69
All-in-one vs. CD-A	64.51	58.96	27.30	16.59	32.29	39.93

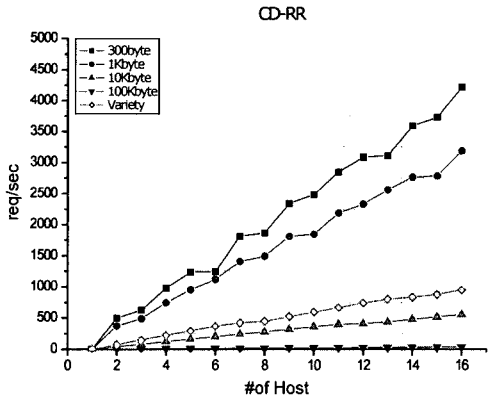


그림 8 호스트 개수에 따른 초당 요청수 (CD-RR)

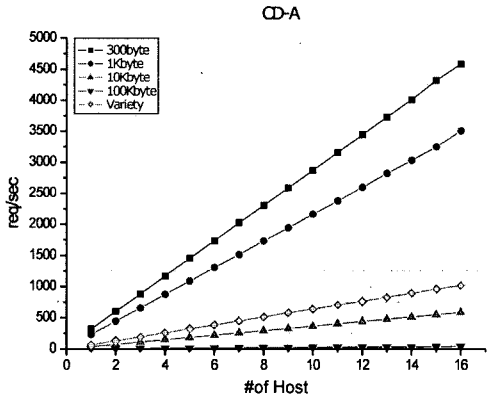


그림 9 호스트 개수에 따른 초당 요청수 (CD-A)

상률을 나타낸다. 캐시간 협동을 위해 MD5를 사용할 경우 일부 캐시로 요청이 몰려 전체 성능이 일부 캐시에 의존하는 현상을 이 표를 통해 설명할 수 있다. RR (라운드 로빈)의 경우는 매 요청에 대해 어떤 캐시가 자신의 요청을 가졌는가에 상관없이 부하 분산을 함으로 동일한 성능의 프록시들은 동일한 개수의 요청을 받아 일을 처리하게 된다. 그러나 MD5의 경우는 각각의 요청에 대해 기존에 사용했던 캐시로 매칭함으로 프록시들이 처리하는 요청 개수는 동일하지 않고 하나의 프록시로 요청이 몰리는 현상이 발생한다. 이때, 전체 성능 측정(초당 요청수)은 맨 마지막에 요청을 끝낸 프록시를 기준으로 함으로 요청이 동일하게 분포하는 RR 방식이 MD5에 비해 성능이 좋아짐을 알 수 있다. 이 실험은 CD 방식에서 캐시간 협동을 하지 않고(MD5) All-in-one에서 사용하는 부하 분산 방식(RR)을 사용할 경우 얼마나 더 성능이 향상될 수 있는지를 보여준다.

CD-A는 All-in-one과 마찬가지로 CD에 사용된 모듈(FE, CD)을 하나의 호스트에 집어넣고 LVS를 이용하여 클러스터링한 구조이다. 캐시간 협동성을 지원하지는 않지만 모든 구조 중에서 가장 성능 좋은 구조임을 확인하였고 그림 9와 표 14는 이러한 결과를 나타낸다. CD-A가 가장 좋은 성능을 보이는 이유는 CPU 활용도 측면에서 설명할 수 있다. 예를 들면, CD 시스템은 표 9(a)에서 보여지듯이 호스트의 개수가 3개일 때 CD1을 제외하고 나머지 모듈들의 CPU 활용도가 낮음을 알 수 있다(FE1 = 85%, CD1 = 100%, FE2 = 87.4%). 반면 CA-A 시스템은 모든 모듈이 하나의 호스트에 들어있고, 요청된 이미지의 크기에 따라 필요한 모듈의 CPU 점유율이 상대적으로 높게 배정되는 방식으로 운영된다. 그래서 항상 CPU 점유율이 100%가 됨으로 구조적 관점에서 CPU 활용도가 높다. 이 실험은 CD-A 구조가 All-in-one 구조를 유지하면서 내부적으로 Cache와 Distiller의 통신 구조를 단순화 함으로써 성능 관점에서 가장 좋은 구조임을 보여준다.

4.4 토론

제안된 시스템이 기존 시스템(TranSend, All-in-one)에 비해 성능이 좋아진 이유는 구조적 관점에서 복잡한 구조를 단순한 구조로 바꾼 결과로 설명할 수 있다. 단순화 과정은 다음 2가지로 요약할 수 있다. 제안된 단순화된 구조는 모듈(FE, Cache, Distiller)간의 통신이 상대적으로 빈번한 작은 크기의 이미지 요청 실험에서 더 좋은 성능을 보인다.

- 단순화 1 : Cache에서 FE, FE에서 Distiller로 통신하는 구조를 Cache와 Distiller가 직접 통신하는 구조로 단순화
- 단순화 2 : Cache와 Distiller가 소켓 통신을 이용하

지 않고 Cache에서 압축을 직접 수행하는 구조로 단순화

또한 제안된 구조가 All-in-one 구조에 비해서 가지는 장점은 프록시를 구성하는 캐시의 협동성을 들 수 있다. 즉, 사용자가 임의의 요청을 했을 때 제안된 구조는 전체 프록시가 가지는 캐시의 합이 사용자 요청 개수와 같지만 All-in-one 구조는 각각의 프록시가 모든 사용자 요청 데이타를 가지고 있어야 한다. 그러므로 All-in-one이 가지는 전체 캐시의 합은 사용자 요청 데이타 수와 사용된 프록시의 수의 곱으로 나타난다.

제안된 구조의 단점은 다음과 같다.

- 확장성 : 제안된 구조는 TranSend와 마찬가지로 확장시에 시스템적인 방법이 없는 단점을 가진다. 이를 위해 CD 구조를 All-in-one의 호스트 내부 구조에 적용하고, LVS에서 부하 분산시에 MD5를 적용할 수 있다.

5. 결 론

본 논문에서는 무선 인터넷의 근본적인 문제점 중 일부를 해결 할 수 있도록 제안된 TranSend 및 이의 개선 구조인 All-in-one 프록시 서버의 문제점을 지적하고, 구조 및 성능 개선을 제안하였다. TranSend 및 All-in-one 프록시 서버의 문제점을 확장성, 복잡성, 캐시간 협동 관점에서 분석하였고, 이를 해결하기 위해 새로운 구조를 제안하였다. 실험을 통해 제안된 구조가 성능 향상에 기여했음을 확인하였다.

향후 연구 방향을 요약하면 다음과 같다.

- LVS의 NAT(Network Address Translation) 방식을 DR(Direct Routing) 방식으로 변환 : NAT 방식은 외부와 내부 패킷이 모두 LVS를 통과함으로 호스트를 증가하는데 한계를 가진다.
- 부하 분산 시에 라운드 로빈 대신에 호스트의 처리 능력에 따른 부하 분산 방식으로 변환 : 부하 분산에서 라운드 로빈을 적용할 경우 부하가 동일하게 분산됨으로 최종 결과는 처리 능력이 가장 적은 호스트에 종속되는 단점을 가진다.
- MD5 Hash의 side-effect를 최소화 : MD5 Hash를 사용하면 캐시간 협동성을 보장하나 하나의 캐시로 요청이 집중되어 전체 성능이 집중된 일부 캐시에 종속되는 단점을 가진다. 이를 해결하기 위해 2D 또는 3D MD5 Hash의 적용을 고려해볼 수 있다.

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] C. Perkins, "Mobile IP," Communications Magazine, IEEE, Vol. 35, No. 5, pp. 84-99, 1997.

[4] F. Sultan, K. Srinivasan, D. Iyer and L. Iftode, "Migratory TCP: connection migration for service continuity in the Internet," Proceedings of 22nd International Conference on Distributed Computing System, IEEE, pp. 469-470, 2002.

[5] N. Eshak and M. Baba, "Design a new transport protocol (wireless TCP) to support mobility for mobile ad hoc networks," NCTT 2003 Proceedings of 4th National Conference on Telecommunication Technology, IEEE, pp. 144-147, 2003.

[6] S. Ross, J. Hill, M. Chen, A. Joseph, D. Culler and E. Brewer, "A security architecture for the post-PC world," U. C. Berkeley Technical Report, to appear.

[7] B. Zenel and D. Duchamp, "A general purpose proxy filtering mechanism applied to the mobile environment," Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking, pp. 248-259, 1997.

[8] Z. Sahinoglu and P. Orlik, "Power efficient transmission of layered video through wireless proxy servers," IEEE Electronics Letters, Vol. 39, Issue 8, pp. 698 - 699, 2003.

[9] P. Yong and J. Modestino, "Interactive video coding and transmission over wired-to-wireless IP networks using an edge proxy," IEEE Internation Conference on Acoustics, Speech, and Signal Processing, Vol. 4, pp. 281-284, 2003.

[10] P. Mckinley, T. Chiping and A. Mani, "A study of adaptive forward error correction for wireless collaborative computing," IEEE Transactions on Parallel and Distributed Systems, Vol. 13, Issue 9, pp. 936-947, 2002.

[11] Z. Jiang, K. Leung, B. Kim and P. Henry, "Seamless mobility management based on proxy server," Wireless Communications and Networking Conference, IEEE, Vol. 2, pp. 563-568, 2002.

[12] J. Rendon, F. Casadevall and J. Carrasco, "Wireless TCP proposals with proxy servers in the GPRS network," The 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications," Vol. 3, pp. 1156-1160, 2002.

[13] B. Yao and W. Fuchs, "Recovery proxy for wireless applications," Proceedings of 12th International Symposium on Software Reliability Engineering, IEEE, pp. 112-119, 2001.

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

2.
[15] A. Fox, "A Framework For Separating Server Scalability and Availability From Internet Application Functionality," Ph. D. dissertation, U. C. Berkeley, 1998.
[16] M. Liljeberg, H. Helin, M. Kojo and K. Raatikainen, "Enhanced Services for World Wide Web in Mobile WAN Environment," Department of Computer Science, University of Helsinki, Report C-1996-28, 1996.
[17] B. Housel, G. Samaras and D. Lindquist, "WebExpress: A client/intercept based system for optimizing web browsing in a wireless environment," Mobile Networks and Applications, ACM, pp. 419-431, 1998.
[18] J. Lee, M. Kim, H. Youn, Y. Hahm and D. Lee, "Class-based proxy server for mobile computers," Proceedings of International Workshops on Parallell Processing, IEEE, pp. 559-566, 2000.
[19] K. Ham, S. Jung, S. Yang, H. Lee and K. Chung, "An Enhanced Proxy Architecture for Efficient Web Browsing over Cellular Networks," Proceedings of the 14th International Conference on Information Networking, pp. 5A 4.1-4.5, 2000.
[20] K. Kim, H. Lee and K. Chung, "A Distributed Proxy Server System for Mobile Web Service," Proceedings of the 15th International Conference on Information Networking, IEEE, pp. 8A 749-754, 2001.
[21] A. Maheshwari, A. Sharma, K. Ramamritham and P. Shenoy, "TranSquid: transcoding and caching proxy for heterrogeneous e-commerce environments," Proceedings of 12th International Workshop on RIDE-2EC, IEEE, pp. 50-59, 2002.
[22] B. Knutsson, H. Lu, J. Mogul, "Architecture and pragmatics of server-directed transcoding," Proceedings of 7th International Workshop on Web Content Caching and Distribution, 2002.
[23] C. Bowman, P. Danzig, M. Scwartz and et al, "Harvest: A Scalable, Customizable Discovery and Access System," Technical Report, CU-CS-732-94, University of Colorado, 1994.
[24] D. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, 1992.
[25] LVS(Linux Vitual Server), http://www.linuxvirtualserver.-org
[26] AB(Apache Bench), http://httpd.apache.org/docs--2.0/programs/ab.html
[27] Virtual Server via NAT, http://www.linuxvirtualserver.org/VS-NAT.html
[28] Squid Web Proxy Cache, http://www.squid--cache.org
[29] 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
[30] 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.
[31] 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

1996년 호서대학교 전자공학과 졸업(학사). 1998년 숭실대학교 전자공학과 대학원 졸업(석사). 1998년~1998년 7월 숭실대학교 정보통신전자공학부 박사 과정 1998년 8월~2000년 7월 (주) 3R 부설 연구소 주임 연구원. 2000년 8월~현재 숭실대학교 정보통신전자공학부 박사 과정. 관심분야는 mobile computing (network, system & application)

정 규 식

1979년 서울대학교 전자공학과(공학사) 1981년 한국과학기술원 전산학과(이학석사). 1981년~1984년 금성사(현 LG전자) 중앙연구소 선임연구원. 1986년 미국 University of Southern California(컴퓨터공학석사). 1990년 미국 University of Southern California(컴퓨터공학박사). 1990년~1993년 (주) 코리아써키드 기술자문. 1993년 12월~1994년 3월 미국 IBM Wastson 연구소 방문 연구원. 1998년 2월~1999년 2월 미국 IBM Almaden 연구소 방문 연구원. 1990년 9월~현재 숭실대학교 정보통신전자공학부, 교수. 관심분야는 internet infrastructure, high performance internet computing & wireless internet