클러스터링 기반의 무선 인터넷 프록시 서버
(A Clustering based Wireless Internet Proxy Server)

곽 후 근 † 우 재 용 †† 정 윤 재 †† 김 동 승 ††† 정 규 식 ††††
(Hukeun Kwak) (Jaeyong Woo) (Yunjae Jung) (Dongseung Kim) (Kyusik Chung)

요 약 기존 유선 인터넷과 달리 무선 인터넷은 낮은 대역폭, 빈번하게 접속이 끊기는 현상, 단말기내의 낮은 컴퓨팅 파워 및 작은 화면, 사용자의 이동성 등의 특성에 따른 많은 제약점들을 갖고 있다. 또한 무선 인터넷 서버는 급증하는 사용자에 따른 대용량 트래픽을 처리할 수 있도록 확장성이 있어야 한다. 이에 위의 문제를 캐싱(Caching)과 압축(Transcoding, Distillation)으로 해결하는 방법으로 무선 프록시 서버를 사용한다.

TranSend는 클러스터링 기반의 무선 프록시 서버로 제안된 것이나 시스템적인(Systematic) 방법으로 확장성을 보장하지 못하는 단점을 가진다. 이에 본 논문에서는 시스템적인 방법으로 확장성을 보장하는 클러스터링 기반의 무선 인터넷 프록시 서버를 제안한다. 16대의 컴퓨터를 사용하여 실험을 수행하였고 실험 결과 TranSend 시스템에 비해 32.17%의 성능 향상을 보였다.

키워드 : 무선 인터넷, 프록시 서버, 클러스터링

Abstract As different from wired internet, wireless internet has limitations due to the following characteristics; low bandwidth, frequent disconnection, low computing power, small screen in user terminal, and user mobility. Also, wireless internet server should be scalable to handle a large scale traffic due to rapidly growing users. Wireless proxy servers are used for the wireless internet because their caching and transcoding functions are helpful to overcome the above limitation.

TranSend was proposed as a clustering based wireless proxy server but its scalability is difficult to achieve because there is no systematic way to do it. In this paper, we proposed a clustering based wireless internet proxy server which can be scalable in a systematic way. We performed experiments using 16 PCs and experimental results show 32.17% performance improvement of the proposed system compared to TranSend system.

Key words : Wireless internet, Proxy server, Clustering

1. 서 론

현재 정보화 사회는 인터넷과 무선 이동 통신이라는 커다란 두개의 축에 의해 선도되고 있으며 인터넷과 무선 이동 통신은 서로 조화를 이루면서 급속도로 성장하고 있다. 무선 인터넷이란 언제 어디서나 인터넷에 접속하여 다양한 정보검색과 전자상거래 등을 하는 것으로 기존 인터넷 환경의 공간적 제약을 극복하여 서비스하는 방식이며, 좀더 넓은 의미로는 무선 랜등 고정 무선 인터넷 서비스를 포함하여 무선을 통해 인터넷에 접속하는 것을 뜻한다. 급격한 인터넷 사용의 증가와 핸드폰, 노트북, PDA 등 휴대용 단말기의 사용 급증으로 사람들이 이동하면서 언제든지 인터넷망에 접속하여 정보를 얻고, 통신할 수 있는 무선 인터넷은 더욱 더 새로운 사회적 관심거리가 되고 있다.

무선 인터넷의 사용이 증가하고 있는 현실에서 무선 인터넷의 본질적인 문제 역시 무시할 수 없는 요소로 부각되고 있다. 현재까지 나와 있는 무선 인터넷의 근본적인 문제점과 고려해 볼 수 있는 해결책은 다음과 같다.

- 낮은 대역폭 : 압축(Transcoding, Distillation)[1], 캐싱(Caching)

· 이 연구는 한국과학재단 특정기초(KOSEF R01-2001-000-00341-0)의 연구비 지원에 의해 수행되었음
† 학생회원 : 숭실대학교 정보통신전자공학부
gobarian@q.ssu.ac.kr
†† 비 회 원 : 숭실대학교 정보통신전자공학부
tad572@q.ssu.ac.kr
jgyver@q.ssu.ac.kr
††† 종신회원 : 고려대학교 전기공학과 교수
dkim@classic.korea.ac.kr
†††† 종신회원 : 숭실대학교 정보통신전자공학부 교수
kchung@q.ssu.ac.kr
논문접수 : 2003년 9월 17일
심사완료 : 2003년 11월 6일

- 빈번하게 연결이 끊기는 현상 : 쿠키(Cookie), 캐싱(Caching)[2]
- 단말기내의 컴퓨팅 파워 및 작은 화면 : 압축(Transcoding, Distillation)
- 단말기 사용자의 이동성 : Mobile IP[3], TCP Migration[4]

이에 위의 문제를 캐싱(Caching)과 압축(Transcoding, Distillation)으로 해결하는 방법으로 무선 프록시를 사용한다. 그림 1은 무선 인터넷에서 사용되고 있는 무선 프록시를 나타내고 이의 기능을 정리하면 다음과 같다.

- 압축(Transcoding, Distillation)[1]
- 캐싱(Caching)[2]
- FEC(Forward Error Correction)[5]
- 보안(Security)[6]
- 핸드오프(Handoff)[7]
- 멀티미디어 전송[8]
- 프로토콜 변환(Protocol Translation)[9]
- 오류 복구(Failure Recovery)[10]

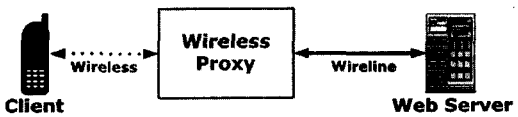


그림 1 무선 프록시

본 논문에서는 기존 무선 프록시가 가지는 문제점을 분석하고 이를 해결할 새로운 구조를 제안한다. 본 논문의 구성은 다음과 같다. 제2장에서는 기존 무선 프록시 및 이들이 가지는 문제점을 소개한다. 3장에서는 기존 무선 프록시가 가지는 문제점을 해결하는 새로운 구조를 설명하고, 4장에서는 실험 및 토론을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 연구 배경

2.1 기존 무선 프록시 서버

기본적으로 무선 프록시는 캐싱(Caching), 압축(Distillation), 대용량 트래픽에 대한 확장성(Scalability)을 고려하여야 한다. 이러한 관점에서 기존 무선 프록시 서버[11-17]를 정리하면 표 1과 같다.

기존 무선 프록시들은 캐싱(Caching) 및 압축(Distillation) 기능을 제공하고 있다. 확장성 관점에서는 일부는 확장성을 고려하고 있지만 대부분은 고려하고 있지 않다.

Mowgil(Mobile Office Workstations using GSM Links)[11]는 프로토콜과 관련된 부하를 최소화하기 위해 TCP 대신 MDCP(Mowgli Data Channel Protocol)를 HTTP 대신 MHTTP(Mowgli HTTP)를 사용하였다. 또한 Mowgli는 갑작스럽게 연결이 끊기는 것을 보완하기 위해 캐싱(Caching)을 이용하였다.

WebExpress[12]는 사용자 요청에 대한 HTML 응답이 많은 부분 비슷하다는 것에 착안하여 차이나는 부분만을 전송하는 방식(Differencing)을 사용하였다. 또한 WebExpress는 프로토콜 부하를 줄이기 위해 TCP/IP 연결 부하 및 HTTP 헤더를 줄였다.

TranSend[13]는 대용량 트래픽에 대한 확장성을 고려하여 클러스터링으로 구현된 무선 프록시로서 본 논문에서는 이를 기반으로 새로운 구조를 제안한다.

Class-based Proxy[14]는 사용자 이동 기기의 화면 크기에 따라 요청 이미지의 압축 정도를 조절하는 방식을 사용하였다. 또한 압축된 이미지를 공유함으로써 같은 화면 크기를 가지는 다른 사용자가 이를 이용하도록 하였다.

KWU Proxy[15]는 HTML에 포함된 이미지 등이 새로운 연결이 필요하다는 점에 착안하여 프록시가 미리 사용자를 대신하여 이미지 등을 받아오고 이를 사용자에게 전송함으로써(Prefetching) HTTP와 관련된 처리를 최적화하였다.

Distributed Proxy[16]은 기존 무선 프록시가 가지는

표 1 기존 무선 프록시 비교

Wireless Proxy	caching	distillation	scalability	etc
Mowgli[11]	O	O	X	◦ Minimizing protocol overhead ◦ Disconnected-mode support
WebExpress[12]	O	O	X	◦ Differencing ◦ Protocol reduction
TranSend[13]	O	O	O	◦ Clustering
Class-based Proxy[14]	O	O	X	◦ According to display class ◦ Sharing distilled files
KWU Proxy[15]	O	O	X	◦ Optimizing HTTP processing
Distributed Proxy[16]	O	O	O	◦ Handoff protocol
TranSquid[17]	O	O	X	◦ Server-directed transcoding[18]

단점인 단일장애점(a single point of failure)이나 대용량 트래픽에 대한 서비스를 고려하여 여러대의 프록시를 지역적으로 분산시키는 방식을 사용하였다. 또한 분산 프록시간의 핸드오프를 위해 새로운 핸드오프 프로토콜을 제안하였다.

TranSquid[17]은 데이타가 가지는 문맥(Semantics)을 무시하고 종류에 따라 무조건 압축(Distillation, Transcoding)을 수행하는 기존 무선 프록시의 단점을 보안하기 위해 Server-directed transcoding[18]을 사용하였다. Server-directed transcoding은 데이타를 가지고 있던 서버가 문맥에 따른 정보를 제공하여 이를 압축시에 이용하는 방식이다.

2.2 클러스터링 무선 프록시 서버[13]

그림 2는 TranSend 프록시 시스템의 전체적인 구조를 나타낸다.

각 모듈로써 Front End(FE, MS:Manager Stub), User Profile DB, Cache($), Worker(W, Worker API, WS:Worker Stub), Manager, Graphical Monitor가 구성된다. 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 데이타에 대한 압축을 수행한다. Manager는 Distiller를 관리하고, Graphical Monitor는 시스템 전체의 상태를 볼 수 있게 해준다.

Client 요청을 FE가 받고 Cache 서버에 요청하여 존재하면 해당 데이타를 받고, 존재하지 않으면 Cache 서버가 웹 서버로부터 요청하여 받아온다. FE는 그 데이타를 Worker(Distiller)에게 보내 압축을 요청하며 압축된 데이타를 Client에게 보내는 방법으로 동작한다.

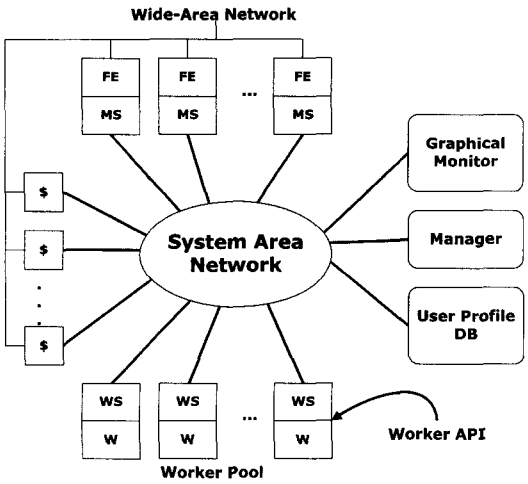


그림 2 TranSend 프록시 시스템 구조

각 모듈별 구체적인 기능은 다음과 같다.

2.2.1 Front End(FE)

FE가 하는 일은 다음과 같다.

- 외부 HTTP 인터페이스 제공 및 사용자 요청에 대한 전체적인 관리
- 400개의 쓰레드(Thread)로 구성되며 사용자 요청을 병렬적으로 처리
- User Profile DB를 통한 사용자 관리(Customization)
- 오류 복구(fault tolerance) : Manager로부터 주기적으로 신호(beacon)가 오지 않을 경우 새로운 Manager를 실행한다.

2.2.2 User Profile DB

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.

2.2.3 Cache

TranSend 시스템은 4개의 Harvest Cache[19] 노드(Node)로 구성되어 있다. 4개의 Cache 노드를 효율적으로 관리하기 위하여 Manager Stub에서는 사용자가 요청한 주소(URL)에 대해 MD5 Hash[20]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 있는 구조로 되어 있다.

2.2.4 Worker(Datatype-Specific Distiller)

Worker API는 두가지 기능을 수행하는데, 하나는 압축(Distillation) 지연 시간(Latency)를 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.

TranSend 시스템은 다음과 같은 3가지의 Distiller로 구성된다.

- JPEG Distiller : 크기(Scaling)와 양화자(Quantization) 값을 조절하여 압축을 수행한다. 압축을 수행하는 함수는 기존 코드인(off-the-shelf) JPEG-6a library[21]를 사용한다.
- GIF Distiller : GIF는 JPEG으로 변환하고 JPEG 압

축을 수행한다. GIF를 JPEG로 변환하는 이유는 JPEG으로 작업을 수행하는 것이 간단하고 좋은 성능을 가지기 때문이다.

- HTML Distiller : 압축된 이미지 옆에 원 이미지를 볼 수 있는 링크와 사용자 정보(Preference)를 조절할 수 있는 툴바(Toolbar)를 추가한다.

2.2.5 Manager

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 단일장애점(a single point of failure)이 될 가능성이 존재한다는 점이다.

Manager가 하는 일은 다음과 같다.

- 멀티캐스팅 : Manager는 주기적으로 신호(beacon)를 멀티 캐스팅한다. FE와 Distiller는 이 신호로부터 Manager의 위치를 파악한다. 또한 이 신호에 Distiller들의 위치와 부하 정보를 실어서(piggyback) 보내고 이를 FE가 이용한다(매 사용자 요청마다 FE가 Manager와 통신하는 것을 피하기 위함).
- 새로운 Distiller 추가(Spawn) : Distiller들로부터 주기적으로 얻은 부하(Distiller 큐(Queue)에 쌓인 일의 양)가 임계값을 넘을 경우 새로운 Distiller를 추가한다.
- Distiller들의 부하 분산 : Distiller들로부터 주기적으로 얻은 부하 정보와 로터리 스케줄링(Lottery scheduling)[22]을 이용하여 부하 분산을 한다.
- 오류 복구(fault tolerance)와 시스템 튜닝(tuning) : Distiller가 작업 수행 도중 다운(Fail)되거나 Distiller로부터 주기적인 상태 정보가 오지 않으면 그 Distiller를 다시 추가(Spawn)한다.

2.2.6 Graphical Monitor

Tcl/Tk 기반의 Monitor는 시스템 전체의 상태를 볼 수 있게 만들어 졌다. 시스템상의 모듈들이 멀티캐스트 그룹을 통하여 Monitor에게 자신의 정보를 알려 준다. Monitor와의 통신에서 멀티캐스트 그룹을 사용하는 이유는 지역적으로 떨어진 곳에서 원격 관리를 위해 다수의 Monitor를 사용하기 위해서이다. 또한 시스템내의 심각한 에러가 발생하였을 때, 예를 들어 어떤 모듈로부터 정보가 오지 않을 때, 시스템 관리자에게 메일이나 호출을 보낼 수 있다.

2.3 접근 방식

본 논문에서는 TranSend 무선 프록시 서버가 가지는 문제점을 확장성(Scalability), 부하 분산(Load Balancing), 오류 복구(Failure Tolerance) 측면에서 정리하고, 3장에서는 이를 해결할 새로운 무선 프록시 구조를 제안한다.

2.3.1 확장성(Scalability)

그림 2 구조에서 보면 FE, Cache, Distiller는 각각 여러개의 노드(Node) 들로 구성가능하다. 프록시 서버를 확장성있게 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.

표 2는 총 16대의 호스트로 TranSend 시스템을 구성했을 때, 실험 결과에 의존하여 확장성 있게 시스템을 확장한 경우로 요청된 이미지 크기에 따른 모듈의 수를 나타낸다(4.3절 (1) TranSend 실험 결과 참조). 표에서 보여지듯이 이미지 크기가 작을수록 FE와 Cache가 더 필요하고 이미지가 클수록 Distiller를 많이 필요로 함을 알 수 있다. 이렇게 성능을 높이기 위해 모듈의 수를 늘릴 수 있으나 스케일(Scale)하게 만드는데 시스템적인 방법이 없다.

표 2 이미지 크기에 따른 모듈들의 수(총 16대)

이미지 크기	FE	Cache	Distiller	Manager & Monitor	Total
300 bytes	8	4	3	1	16
1 Kbytes	6	3	6	1	16
10 Kbytes	2	1	12	1	16
100 Kbytes	1	1	13	1	16
Variation	3	1	11	1	16

2.3.2 부하 분산(Load Balancing)

표 3은 TranSend를 구성하는 모듈들이 사용하고 있는 부하 분산 방식을 나타낸다. 표에서 보여지듯이 FE의 경우 여러개의 FE를 위한 부하 분산 방식이 고려되어 있지 않고, Cache의 경우는 MD5 Hash를 사용하여 동일 주소(URL)에 대해 동일 Cache를 할당할 수는 있으나 동일 주소로 요청이 몰릴 경우 부하 분산이 제대로 되지 않는 문제점이 존재한다.

표 3 TranSend 모듈들의 부하 분산 방식

모듈	부하 분산
FE	없음
Distiller	Lottery Scheduling
Cache	MD5 Hash

2.3.3 오류 복구(Fault Tolerance)

표 4는 TranSend를 구성하고 있는 모듈들의 오류 복구를 나타낸다. 표에서 보여지듯이 Manager와 Distiller를 제외하고 FE와 Cache는 오류 복구가 되지 않는 문제점이 존재한다. FE의 경우 모든 사용자 요청의 통로

역할을 수행함으로 오류가 발생했을 경우 복구가 되지 않으면 TranSend 시스템 전체를 사용할 수 없는 경우(a single point of failure)가 발생한다. Cache의 경우 4개의 노드를 사용하나 오류가 발생해도 그 Cache를 요청 대상에서 제외하지 않는 문제를 가진다.

표 4 TranSend 모듈들의 오류 복구

모듈	오류 복구	방식
FE	X	◦ 없음
Manager	O	◦ FE가 Manager로부터 주기적으로 신호(beacon)가 오지 않을 경우 그 Manager를 다시 실행
Distiller	O	◦ Distiller가 작업 수행 도중 죽거나 Distiller로부터 주기적인 상태 정보가 오지 않으면 그 Distiller를 다시 추가(Manager에 의해 수행)
Cache	X	◦ 없음

2.3.4 본 연구의 접근 방식

본 논문에서는 시스템적으로 확장하지 못하는 TranSend 프록시 구조의 단점을 해결하는 새로운 구조를 제안한다. 제안된 구조는 부하 분산이나 오류 복구 측면에서도 기존 구조에 비해 장점을 가진다.

3. 제안된 프록시 서버

그림 3은 2.3절에서 분석된 TranSend 무선 프록시의 문제점을 기반으로 이를 해결할 수 있도록 제안된 구조이다. 제안된 구조는 TranSend에 사용된 모듈들을 모두 하나의 호스트에 넣고(이하 All-in-one) 이 호스트들을 LVS(Linux Virtual Server)[23]를 사용하여 부하 분산을 하는 것이다. TranSend에서는 각각의 모듈들(FE, Cache, Distiller) 각각이 클러스터링 되어 있는 반면에 본 논문에서는 각 모듈들을 하나의 호스트들에 포함하고 이러한 호스트를 클러스터링하는 구조로 되어 있다.

제안된 구조의 동작원리는 다음과 같다.

- 사용자(Client)가 제안된 프록시 구조(All-in-one)에 데이타를 요청한다.
- 요청을 받은 LVS가 라운드 로빈(Round Robin) 방식으로 임의의 Host를 선택한다.
- 선택된 호스트는 실제 웹 서버에 데이타를 요청한다.
- 호스트는 웹 서버로부터 받은 데이타를 LVS를 통해 사용자에게 전송한다.

그림 4는 Host안에서 Client 요청 처리 순서를 나타내고 각 단계를 요약하면 다음과 같다.

Step 1 : 주기적으로 Distiller는 부하 정보를 Manager로 보내고, Manager는 이 부하 정보의 평균값을 FE로 보낸다.

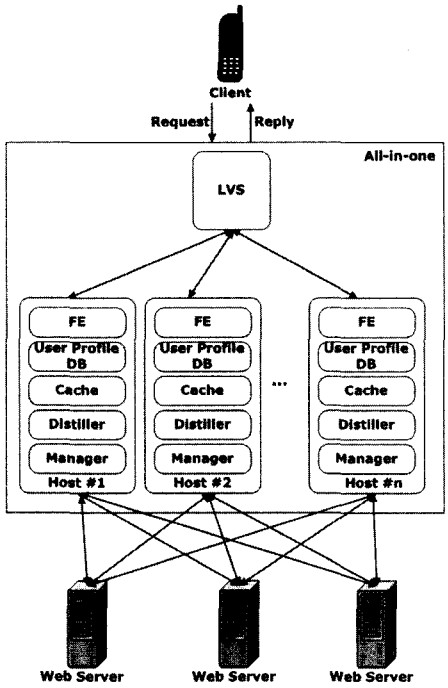


그림 3 제안된 구조(All-in-one)

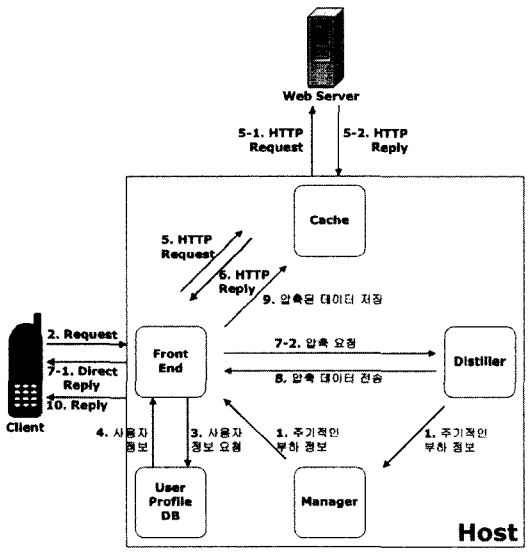


그림 4 Client 요청 처리 순서

Step 2 : 사용자(Client)가 FE에 데이타를 요청한다.

Step 3 : FE는 User Profile DB에 사용자 정보(Preference)를 요청한다.

Step 4 : User Profile DB는 사용자 정보(Preference)를 FE로 보낸다.

Step 5 : FE는 사용자가 요청한 데이타를 Cache에게 요청한다.
1) Cache에 요청된 데이타가 없으면 외부의 웹 서버로 요청한다.
2) 웹 서버는 Cache로 데이타를 보낸다.

Step 6 : Cache는 FE에게 요청된 데이타를 보낸다.

Step 7 : Cache로부터 받은 데이타를 확인한다.
1) 데이타가 압축(Distillation)된 형태라면 바로 사용자에게 보낸다.
2) 데이타가 압축(Distillation)된 형태가 아니라면 FE는 Distiller에게 Cache로부터 받은 데이타와 사용자 정보(Preference)를 보내서 압축을 요청한다.

Step 8 : Distiller는 데이타를 압축(Distillation)한 다음 FE로 보낸다.

Step 9 : FE는 압축(Distillation)된 데이타를 Cache에 저장한다.

Step 10 : FE는 사용자에게 데이타 요청에 대한 응답을 한다.

이 동작과정은 TranSend과 거의 동일하다. 차이점은 다음과 같다. TranSend에서는 FE에서 Distiller로 요청한다. Distiller가 여러개 있으면 그 중 하나를 선택하는 과정이 추가된다. 또한 FE에서 Cache로 요청할 때 Cache가 여러개 있으면 그 중 하나를 선택하는 과정이 추가된다.

표 5는 TranSend 시스템과 제안된 시스템(All-in-one)을 구조적으로 비교한 표이다.

4. 실험 및 토론

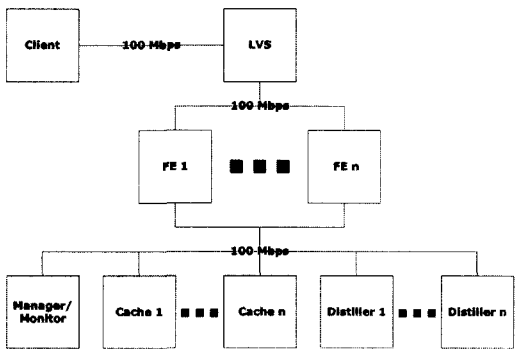
4.1 실험 환경

4.1.1 하드웨어 & 소프트웨어

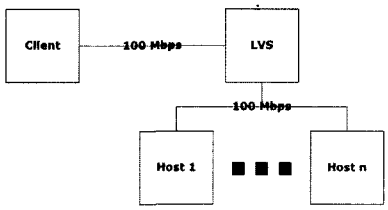
표 6은 실험에 사용된 하드웨어와 소프트웨어를 나타낸다. 프록시 서버는 PC 16대로 구성되어 있고 TranSend 시스템에서 FE의 부하 분산과 제안된 시스템의 부하 분산을 위하여 LVS라는 Load Balancer를 사용하였다. Apache Bench라는 프로그램을 Client에서 수행하여 프록시 서버에 영상(이미지)를 요청하는 방식으로 실험하였다. 표에서 Client와 LVS가 Host보다 하드웨어 성능이 좋은 이유는 TranSend와 All-in-one에서 확장성 실험을 할 때 Client와 LVS에서는 병목이 발생하지 않는 상황에서 프록시 서버내 호스트들 사이의 확장성을 확인하고자 했기 때문이다.

4.1.2 구성도

그림 5는 실험에 사용된 TranSend와 All-in-one의 구성도를 나타낸다. 그림 5(a)는 All-in-one과 Tran-



(a) TranSend



(b) All-in-one

그림 5 실험에 사용된 구성도

표 5 TranSend 시스템 vs. 제안된 구조(All-in-one)

문제점	TranSend 시스템	제안된 구조(All-in-one)
확장성	◦No systematic	◦Systematic
부하 분산	◦FE : 부하 분산을 하지 않음 ◦Cache : 일부 Cache로 부하가 몰릴 가능성 존재	◦라운드 로빈(Round Robin) 방식으로 고르게 부하를 분산
오류 복구	◦FE : 단일장애점(a single point of failure) ◦Cache : 오류가 발생해도 요청 대상에서 제외되지 않음	◦LVS : 단일장애점(a single point of failure) ◦오류가 발생하면 요청 대상에서 제외

표 6 실험용 하드웨어 & 소프트웨어

		하드웨어		소프트웨어	개수
		CPU (Hz)	RAM (MB)		
Client		P-III 700 M	128	AB[24]	1
LVS		P-IV 2.4 G	512	NAT[25]	1
Host	Cache	P-II 400 M	256	Squid[26]	16
	Distiller			JPEG-6b[27]	

Send를 확장성 측면에서 비교하기 위해 TranSend 기본 구조에 FE를 클러스터링하기 위해 LVS를 추가하였다.

4.2 실험 방법

4.2.1 실험 변수

표 7은 실험에 사용된 변수들을 정리한 것이다.

4.2.2 실험 방법

TranSend의 실험 방법을 정리하면 다음과 같다.

1) 기본 TranSend 시스템을 구성한다. (Host 1 : Manager/Monitor, Host 2 : FE1, Host 3 : Cache1, Host 4 : Distiller1)

2) AB(Apache Bench)를 이용하여 TranSend로 JPEG 이미지를 약 200초 동안 프록시가 처리할 수 있는 최대 개수로 요청한다.

3) 초당 요청 개수 및 각 Host의 CPU 점유율을 측정하여 병목 Host를 알아낸다.

4) 병목 Host의 모듈을 추가한다.

5) 실험에 사용된 Host 수(16대)만큼 2)-4)를 반복한다.

All-in-one의 실험 방법을 정리하면 다음과 같다.

1) AB(Apache Bench)를 이용하여 All-in-one Host로 JPEG 이미지를 약 200초 동안 프록시가 처리할 수 있는 최대 개수로 요청한다.

2) 초당 요청 개수 및 Host 안의 각 모듈의 CPU 점유율을 측정한다.

3) All-in-one Host를 추가한다.

4) 실험에 사용된 Host 수(16대)만큼 1)-3)을 반복한다.

4.3 실험 결과

4.3.1 TranSend

표 8과 그림 6은 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 (Manager/Monitor : 1대, FE : 2대, Cache : 1대, Distiller : 1대), 초당 요청수는 650 requests/sec가 나왔다. 그림 6을 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 Distiller에 집중되어 병목이 발생함으로 Distiller만 추가하면 성능이 향상된다. 그러나 이미지 크기가 작으면 병목이 모듈간(FE, Cache, Distiller)에 골고루 분포하게 되어 이를 모두 해결해야만 성능 향상이 발생함을 알 수 있다.

표 9는 호스트 개수에 따른 CPU 점유율 및 모듈들의 분포를 나타낸다. 표 9(a)에서 보면 호스트 개수가 4개일 경우 Manager & Monitor 1대, FE 1대, Cache 1대, Distiller 1대로 배정되어 수행하였다. CPU 점유율

표 7 실험에 사용된 변수

사용자의 요청 개수	◦약 200초 동안 프록시가 처리할 수 있는 최대 개수
요청 이미지	◦JPEG[28]
요청 크기	◦300 bytes, 1 K, 10 K, 100 Kbytes, Variation[29]
요청 방식	◦같은 크기 : 10개의 다른 이름으로 랜덤(Random)하게 요청 (Cache들 사이에서 MD5 Hash를 적용하기 위해) ◦Variation : 1K-10K 사이의 이미지를 랜덤하게 요청
요청 분포	◦1(100), 2(106), 3(100), 4(95), 5(119), 6(88), 7(98), 8(85), 9(100), 10(108) : 1000개를 기준으로 랜덤 함수로 얻은 분포
사용자 정보(Preference)	◦이미지 Quality = 중간
웹 서버	◦Cache 서버 자체에 둠 (프록시내의 성능 평가에 초점을 맞춤)
이미지 랜덤 요청	◦매 1000개의 요청마다 랜덤 seed를 반복 (Variation의 경우 요청 개수에 따라 이미지 크기의 분포가 틀려짐으로 이를 일정하게 유지하기 위해)
병목(bottleneck)	◦호스트의 CPU 점유율 중 가장 높은 호스트 (본 실험에서는 ethernet이나 system bus 병목은 발생하지 않는다.)

표 8 호스트 개수에 따른 초당 요청수(TranSend)

# of Hosts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	0	0	0	337	650	878	892	887	1290	1476	1537	1555	1711	1920	2027	2190
1 Kbytes	0	0	0	315	410	621	708	860	875	964	1079	1088	1199	1207	1472	1576
10 Kbytes	0	0	0	39	78	115	153	184	198	212	233	259	291	303	336	350
100 Kbytes	0	0	0	2.25	4.64	7.07	9.27	11.34	13.65	15.42	17.61	19.23	20.87	22.7	24.38	26.1
Variation	0	0	0	74	149	201	246	267	306	379	411	435	456	492	520	554

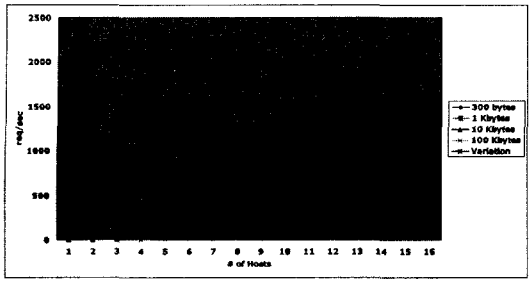


그림 6 호스트 개수에 따른 초당 요청수(TranSend)

이 가장 높은 호스트를 다음번 호스트 추가시 배정하는 원칙을 적용하였다. 호스트가 4대일 경우 FE1의 CPU 사용률이 100%이었음으로 5번째 호스트는 FE로 배정하였다. 표 9(a)를 기준으로 # of Hosts = 4일때 병목난에 FE1이 적혀있고 추가에 FE(FE2)가 적혀있다. 가장 높은 호스트가 2개 이상 존재하면 사용자 요청 처리 순서에 우선하는 호스트를 추가하였다(FE > Cache > Distiller).

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 9(b)의 맨 오른쪽 옆 FE #, C. #, D. #를 보면 이미지 크기에 따라 사용한 FE, Cache, Distiller 숫자가 각각 다르다는 것을 알 수 있다. 표에서 보여지듯이 이미지 크기가 작으면 FE와 Cache의 호스트 개수가 증가하고, 이미지 크기가 크면 Distiller의 호스트 개수가 증가함을 알 수 있다.

4.3.2 All-in-one

표 10과 그림 7은 이미지 크기를 다르게 요청 했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다.

그림 8은 호스트 내 모듈들의 CPU 점유율을 나타낸다. 그림 8(a)는 이미지 크기가 300 bytes인 경우에 호스트의 개수에 따른 호스트 내 모듈들의 CPU 점유율을 나타내고 8(b)는 이들을 평균하여 이미지 크기별로 정리해놓은 그림이다. 이 점유율은 호스트가 증가함에 따라 특정 호스트(여기서 호스트 1번)에서 측정한 결과로서 이미지 크기가 작을수록 FE와 Cache의 점유율이 높고, 이미지 크기가 클수록 Distiller의 점유율이 높음을 알 수 있다. 호스트의 하드웨어 사양이 동일하고 Client로부터 들어온 요청을 라운드 로빈(Round-Robin)으로 각 호스트로 분배함으로 호스트별 측정 데이타를 동일하다.

표 9 호스트 개수에 따른 CPU 점유율 및 모듈들의 분포(TranSend)
(단, MM = Manager & Monitor, C. = Cache D. = Distiller)
(a) CPU 점유율 (300 bytes)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
# of Hosts	MM	FE1	C.1	D.1	FE2	FE3	FE4	C.2	D.2	FE5	FE6	FE7	D.3	C.3	FE8	C.4	병목	추가
4	7.3	100	39.8	36.7													FE1	FE2
5	20	99.8	75.1	68.6	100												FE2	FE3
6	24.3	98	97.3	86	98.1	98.8											FE3	FE4
7	8.5	87.5	100	71.6	88.1	88.4	83.1										C.1	C.2
8	35.4	92.2	57.6	99.9	93.9	93.1	94	50.9									D.1	D.2
9	16.8	99.7	77.1	81.2	100	100	98.9	68.8	43.6								FE2	FE5
10	68.4	100	84.4	90	99.6	99.3	98.8	83.4	91	99.6							FE1	FE6
11	51.4	93.1	86.8	33.2	95	95.2	92.1	75.5	85.7	96.5	94.3						FE5	FE7
12	51.7	90.7	93.9	100	93.8	92.4	90.4	86	100	90	87.3	89.6					D.1	D.3
13	41.1	92.7	100	57.5	94.5	93.1	90.1	90.1	59.1	92.7	91.9	90.4	52.1				C.1	C.3
14	27.6	96.8	47.6	66.6	97.4	97.5	96.6	89.8	75.1	97.8	97.5	96.4	41.2	95.3			FE5	FE8
15	33.9	93.7	52.5	75	95.2	94.1	93	92.3	89.6	93.7	93.6	91.5	45.2	95.9	93.7		C.3	C.4
16	28.9	96.1	79.9	73.3	97.4	96.6	95.7	52.1	72	98.2	97.6	96.1	68.6	62.7	94.8	67	FE5	

(b) 모듈들의 분포

# of Hosts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	FE #	C. #	D. #
300 bytes	MM	FE1	C.1	D.1	FE2	FE3	FE4	C.2	D.2	FE5	FE6	FE7	D.3	C.3	FE8	C.4	8	4	3
1 Kbytes	MM	FE1	C.1	D.1	FE2	D.2	FE3	D.3	FE4	C.2	D.4	FE5	C.3	FE6	D.5	D.6	6	3	6
10 Kbytes	MM	FE1	C.1	D.1	D.2	D.3	D.4	D.5	D.6	FE2	D.7	D.8	D.9	D.10	D.11	D.12	2	1	12
100 Kbytes	MM	FE1	C.1	D.1	D.2	D.3	D.4	D.5	D.6	D.7	D.8	D.9	D.10	D.11	D.12	D.13	1	1	13
Variation	MM	FE1	C.1	D.1	D.2	D.3	D.4	FE2	D.5	D.6	D.7	D.8	FE3	D.9	D.10	D.11	3	1	11

표 10 호스트 개수에 따른 초당 요청수(All-in-one)

# of Hosts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	195	368	520	711	875	1068	1243	1419	1591	1772	1931	2098	2265	2417	2578	2700
1 Kbytes	152	281	420	518	683	826	963	1092	1236	1361	1498	1625	1758	1907	2025	2149
10 Kbytes	28	56	84	114	143	171	200	228	258	288	316	345	374	401	432	458
100 Kbytes	1.89	3.80	5.79	7.90	10.04	12.19	14.36	16.58	18.75	20.94	23.28	25.39	27.57	29.70	31.89	34.08
Variation	49	96	144	193	241	290	338	385	436	485	519	579	632	678	727	774

표 11 All-in-one의 성능 향상률(16개의 호스트 기준)

	300 bytes	1 Kbytes	10 Kbytes	100 Kbytes	Variation	Average
TranSend 1 (req/sec)	2190.27	1575.90	350.29	26.10	553.61	
TranSend 2 (req/sec)	2217.78	1634.77	360.86	28.50	581.98	
All-in-one (req/sec)	2699.92	2148.53	458.43	34.08	773.90	
Improvement 1 (%)	23.27	36.34	30.87	30.58	39.79	32.17
Improvement 2 (%)	21.74	31.43	27.04	19.58	32.98	26.55

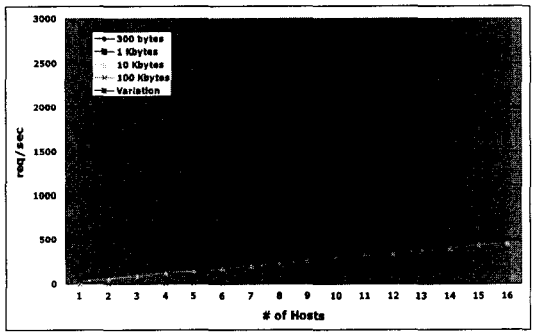
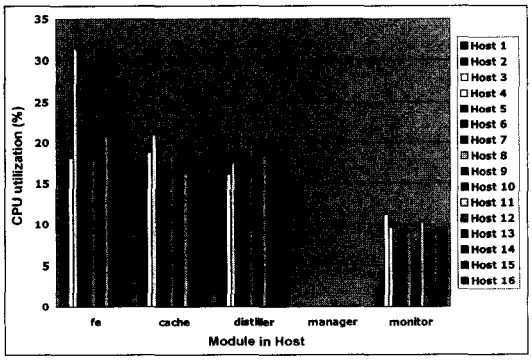
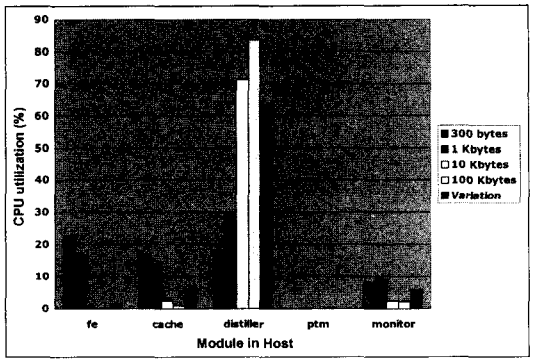


그림 7 호스트 개수에 따른 초당 요청수(All-in-one)



(a) 300 bytes



(b) 평균 CPU 점유율

그림 8 호스트 내 모듈들의 CPU 점유율

4.3.3 TranSend vs. All-in-one

표 11은 16개의 호스트를 기준으로 TranSend에 대한 All-in-one의 성능 향상률을 나타낸 것이다. 평균 향상률(TranSend 1 기준)은 32.17%이고, Distiller의 추가(Spawn) 시간을 고려한 평균 향상률(TranSend 2 기준)은 26.55%이다. 여기서 Distiller의 추가(Spawn) 시간을 고려하였다는 것은 TranSend 시스템의 경우 여러 개의 Distiller를 사용한다고 하더라도 처음에 모두 한꺼번에 사용하는 것이 아니라 기존 Distiller의 부하가 임계값을 넘을 경우 새로운 Distiller를 추가하는 구조로 동작한다. 따라서 모든 Distiller가 동작하려면 어느 정도의 시간이 소요됨을 알 수 있다. TranSend 2의 실험은 이러한 상황을 고려하여 사용자 요청 시간(약 200초)을 2배로 하여 실험을 수행한 것이다.

4.4 토론

제안된 시스템이 기존 시스템에 비해 성능이 좋아진 이유는 구조적 관점에서 호스트들의 CPU 활용도로 설명할 수 있다. 예를 들면, TranSend 시스템은 표 9(a)에서 보여지듯이 호스트의 개수가 4개일 때 FE를 제외하고 나머지 모듈들의 CPU 활용도가 낮음을 알 수 있다 (Manager/Monitor = 7.3%, FE1 = 100%, Cache1 = 39.8%, Distiller1 = 36.7%). 반면 제안된 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.

제안된 구조의 단점은 다음과 같다.

- LVS : 제안된 구조는 FE와 Cache의 오류 복구 문제를 해결하였으나 이의 해결을 위해 새롭게 추가된 LVS가 새로운 단일장애점(a single point of failure)이 되는 문제를 가지고 있다. 이의 해결을 위해서는 백업(backup) LVS를 두어 오류 복구를 하거나 L4/L7 switch[30]를 사용하는 방법이 있다.
- Cache : 제안된 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. 이의 문제를 해결하기 위해서는 Cache 라우팅 테이블(Cache Routing Table)[31]이나 CARP(Cache Array Routing Protocol)[32]의 적용을 고려해 볼 수 있다.

5. 결 론

본 논문에서는 무선 인터넷의 근본적인 문제점 중 일부를 해결 할 수 있도록 제안된 TranSend 프록시 서버의 문제점을 지적하고, 구조 및 성능 개선을 제안하였다. TranSend 프록시 서버의 문제점을 확장성, 부하 분산, 오류 복구 관점에서 분석하였고, 이를 해결하기 위해 새로운 구조를 제안하였다. 실험을 통해 제안된 구조가 성능 향상에 기여했음을 확인하였다.

향후 연구 방향을 요약하면 다음과 같다.

(1) 대용량 이미지나 동영상에 대한 Distiller의 수행 시간 단축

(2) 클라이언트의 환경을 고려한 QoS(Quality of Service)

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] 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.

[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] 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.

[8] J. Seitz, K. Cheverst, N. Davies, M. Ebner and A. Friday, "Management of proxy objects providing multimedia applications in the mobile environment," Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management, pp. 915~928, 1999.

[9] 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.

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

[11] 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.

[12] B. Housel, G. Samaras and D. Lindquist, "Web-Express: A client/intercept based system for optimizing web browsing in a wireless environment," Mobile Networks and Applications, ACM, pp. 419~431, 1998.

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

[14] 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.

[15] 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.

[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] D. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, 1992.
[21] Independent JPEG Group, http://www.ijg.org
[22] C. Waldspurger and W. Weihl, "Lottery Scheduling: Flexible Proportional-Share Resource Management," Proceedings of symposium on Operating System Design and Implementation, 1994.
[23] LVS(Linux Vitual Server), http://www.linuxvirtualserver.org
[24] AB(Apache Bench), http://www.apache.org
[25] Virtual Server via NAT, http://www.linuxvirtualserver.org/VS-NAT.html
[26] Squid Web Proxy Cache, http://www.squid-cache.org
[27] 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.
[28] 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.
[29] S. Chandra, A. Gehani, C. Ellis and A. Vahdat, "Transcoding Characteristics of Web Images," Proceedings of the SPIE Multimedia Computing and Networking Conference, 2001.
[30] T. Schroeder, S. Goddard and B. Ramamurthy, "Scalable Web Server Clustering Technologies," IEEE Network, Vol 14, No. 3, pp. 38~45, 2000.
[31] R. Malpani, J. Lorch and D. Berger, "Making WWW caching servers cooperate," 4th Internatinal WWW confererence, 1995.
[32] V. Valloppillil and K. Ross, "Cache array routing protocol v1.0," 1998.

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)

우 재 용
2001년 숭실대학교 전자공학과 졸업(학사). 2003년 숭실대학교 전자공학과 대학원 졸업(석사). 2003년 2월~현재 이처닷컴(eture.com) 연구원. 관심분야는 mobile commerce, unix system

정 윤 재
2002년 숭실대학교 전자공학과 졸업(학사). 2002년 3월~현재 숭실대학교 정보통신전자공학부 석사 과정. 관심분야는 system, operating system & network

김 동 승
1978년 서울대학교 전자공학과 졸업(학사). 1980년 한국과학기술원 전자전기공학과 졸업(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 교수. 관심분야는 highly available scalable WWW server, dynamic load balancing in WWW servers

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