무선 인터넷 프록시 서버 클러스터

곽후근, 한경식, 황재훈, 정규식
숭실대학교 정보통신전자공학부
e-mail : gobarian@q.ssu.ac.kr

A Wireless Internet Proxy Server Cluster

Hu-Keun Kwak, Kyung-Sik Han, Jae-Hoon Hwang, Kyu-Sik Chung
School of Electronics Engineering, Soongsil University

요 약

TranSend 는 클러스터링 기반의 무선 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 구조와 모듈간의 간단한 통신 구조를 가지는 CD 라는 구조를 제안하였다. 그리고 이 두 가지의 장점을 결합하는 CD-A 라는 구조를 제안하였으나 캐시 간 협동성이 없는 단점을 가진다.

이에 본 논문에서는 시스템적으로 확장성을 보장하고, 모듈간의 단순한 통신 구조를 가지며 캐시 간 협동을 보장하는 클러스터링 기반의 무선 인터넷 프록시 서버를 제안한다. 16 대의 컴퓨터를 사용하여 실험을 수행하였고 실험 결과 TranSend, All-in-one, CD 및 CD-A 구조에 비해 각각 91.16%, 30.52%, 28.31%, -6.54%의 성능 향상을 보였다.

1. 서론

현대 정보화 사회에서 무선 인터넷의 사용이 급증하고 있으며, 이에 따라 핸드폰, 노트북, PDA 에서의 무선 인터넷 사용이 점차 보편화되어 가고 있다. 사람들이 이동하면서 언제든지 인터넷에 접속하여 정보를 얻고, 전송할 수 있게 됨으로 무선 인터넷에 대한 사회적 관심이 증가되고 있다.

무선 인터넷의 사용이 증가하고 있는 상황에서 이들이 가지는 근본적인 문제들도 무시할 수 없는 요소로 부각되고 있다: 낮은 대역폭, 빈번하게 연결이 끊기는 현상, 단말기내의 낮은 컴퓨팅 파워 및 작은 화면, 단말기 사용자의 이동성, 네트워크 프로토콜, 보안 등.

이에 본 논문에서는 위의 문제를 캐싱(Caching)과 압축(Distillation)으로 해결하는 방법으로 무선 프록시를 사용한다. 기본적으로 무선 프록시는 캐싱, 압축 및 대용량 트래픽에 대한 확장성(Scalability)을 고려하여야 한다.

1.1 TranSend [1]

그림 1 은 TranSend 프록시 시스템의 전체적인 구조를 나타낸다.

각 모듈로써 Front End(FE), User Profile DB, Cache($), Worker, Manager, Graphical Monitor 가 구성된다. FE 는 Client 요청에 대한 외부 인터페이스를 담당하며, User Profile DB 는 사용자와 관련된 정보(Preference)를 저장한다. Cache 는 Client 의 요청을 처리하며, Worker(Datatype-Specific Distiller) 는 데이터에 대한 압축을 수행한다. Manager 는 Distiller 를 관리하고, Graphical Monitor 는 시스템 전체의 상태를 볼 수 있게 해준다.

Client 요청을 FE 가 받고 Cache 서버에 요청하여 존재하면 해당 데이터를 받고, 존재하지 않으면 Cache 서버가 웹 서버로부터 요청하여 받아온다. FE 는 그 데이터를 Worker 에게 보내 압축을 요청하며 압축된 데이터를 Client 에게 보내는 방법으로 동작한다.

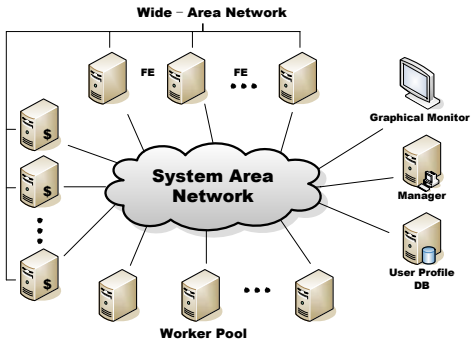

그림 1 TranSend

1.2 All-in-one [2]

All-in-one 은 TranSend 에 사용된 모듈들을 모두 하나의 호스트에 넣고 이 호스트들을 LVS(Linux Virtual Server)[3]를 사용하여 부하 분산을 하는 것이다. 그림 2 는 이를 나타낸

다. TranSend 에서는 모듈들(FE, Cache, Distiller) 각각이 클러스터링 되어 있는 반면에 All-in-one 에서는 각 모듈들을 하나의 호스트에 포함하고 이러한 호스트를 클러스터링하는 구조로 되어 있다.

All-in-one 에서 모듈들을 모두 하나의 호스트에 넣은 이유는 시스템적으로 확장하는 구조를 만들기 위해서이다. 즉, TranSend 는 새로운 모듈을 추가시에 동작과정중의 병목을 찾아 그 모듈을 추가해야하는(No Systematic) 반면에, All-in-one 은 병목에 상관없이 새로운 호스트를 추가하면 (Systematic) 그 호스트내의 모듈 중에서 필요한 모듈이 상대적으로 많이 사용되는 방식을 가지고 있다.

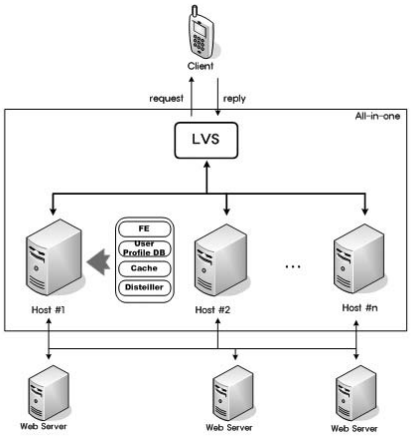

그림 2 All-in-one

1.3 CD (Cache & Distiller) [4]

CD 구조는 TranSend 및 All-in-one 무선 프록시의 문제점을 보완하기 위해 TranSend 에 사용된 기본 모듈에서 Distiller 를 없애고 Cache 에 압축 기능을 추가한 것이다. TranSend 에서는 모듈들(FE, Cache, Distiller) 각각이 클러스터링 되어 있고 All-in-one 은 All-in-one 호스트가 클러스터링 되어 있는 반면에 CD 구조에서는 Distiller 를 제외한 모듈들(FE, CD) 각각이 클러스터링 되어 있다.

CD 구조에서 Cache 에 압축 기능을 추가하고 이를 FE 와 따로 분리하여 전체 구조에서 Distiller 를 없애 불필요하고 복잡한 통신 구조를 단순화하였다. FE 에서 Cache 를 선택할 때 MD5 Hash[5]를 사용하기 때문에 Cache 간 협동이 가능하다. 그림 3 은 CD 구조를 나타내고 있다.

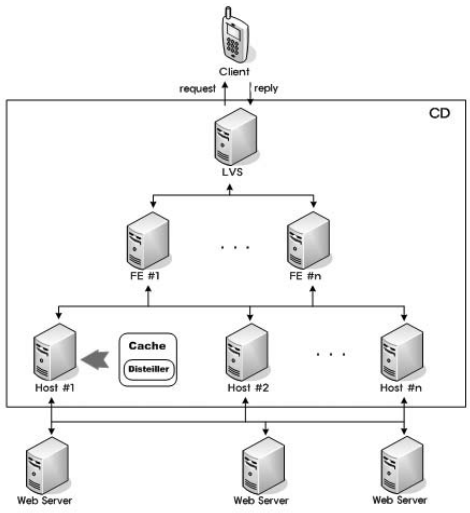

그림 3 CD

1.4 CD-A (CD & All-in-one) [6]

CD-A 구조는 TranSend 에 사용된 기본 모듈에서 Distiller 를 없애고 Cache 에 압축(Distillation) 기능을 추가하여 이 모듈들(FE, Cache)을 하나의 호스트에 넣고 LVS 를 사용하여 부하 분산을 한 것(CD-A: CD & All-in-one)이다.

Cache 에 압축 기능을 추가한 이유는 전체 구조에서 Distiller 를 제거하여 불필요하고 복잡한 통신 구조를 단순화하기 위해서이다. 그리고 모듈들을 하나의 호스트에 넣은 이유는 시스템적으로 확장하는 구조를 만들기 위해서이다. 즉, TranSend 는 새로운 모듈을 추가 시에 동작과정중의 병목을 찾아 그 모듈을 추가해야하는(No Systematic) 반면에, CD-A 는 병목에 상관없이 새로운 호스트를 추가하면 (Systematic) 그 호스트 내에 모듈 중에 필요한 모듈이 상대적으로 많이 사용된다. 그림 4 에서는 CD-A 의 기본 구조에 대해서 나타내고 있다.

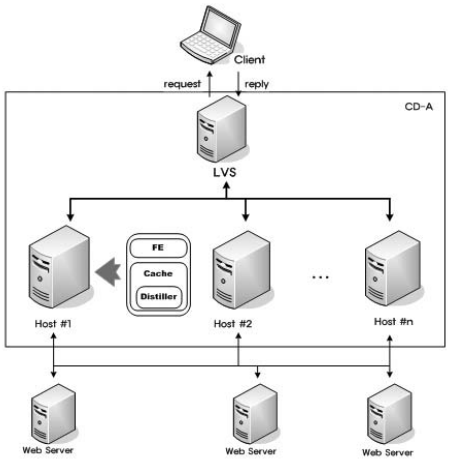

그림 4 CD-A

1.5 접근 방식

TranSend 및 이의 개선 구조인 All-in-one, CD, CD-A 의 문제점을 정리하면 표 1 과 같다.

표 1 기존 구조의 문제점

기존 구조	문제점
TranSend	◦ 확장성(Scalability) : FE, Cache, Distiller 모듈은 각각 여러 개의 노드(Node)들로 구성가능하다. 프록시 서버의 확장성을 위해 노드들을 추가할 때 병목이 발생하는 특정 모듈을 실험 결과에 의존하여 추가 하는 시스템적이지 못한(No Systematic) 방법을 가진다. ◦ 복잡성(Complexity) : FE, Cache, Distiller 의 구성에서 FE 를 중심으로 서로 간에 통신을 하도록 구성되어 있어서 FE 로 모든 통신이 편중되어 있고 Cache 와 Distiller 가 분리되어 있어 불필요한 통신을 하는 단점을 가진다.
All-in-one	◦ 복잡성(Complexity) : TranSend 를 구성하는 모듈을 하나의 호스트에 넣은 구조임으로 TranSend 처럼 모듈간의 통신이 복잡하다는 단점을 가진다. ◦ 캐쉬간 협동성(Cache Cooperation) : LVS 를 통하여 Round Robin 방식으로 각 호스트들에게 요청을 분산시킴으로써 캐쉬 간 협동이 불가능하다.
CD	◦ 확장성(Scalability) : TranSend 구조와 마찬가지로 프록시 서버의 확장성을 위해 노드들을 추가할 때 병목이 발생하는 특정 모듈을 실험 결과에 의존하여 추가 하는 시스템적이지 못한(No Systematic) 방법을 가진다.
CD-A	◦ 캐쉬간 협동성(Cache Cooperation) : LVS 를 통하여 Round

	Robin 방식으로 각 호스트들에게 요청을 분산시킴으로써 캐쉬 간 협동이 불가능하다.

본 논문에서는 기존 구조의 문제점을 토대로 시스템적으로 확장성을 보장하고, 복잡한 통신 구조를 단순화하며 캐시간 협동성을 가지는 새로운 구조를 제안한다.

본 논문의 구성은 다음과 같다. 2 장에서는 기존 무선 프록시가 가지는 문제점을 해결하는 새로운 구조를 설명한다. 3 장에서는 실험 및 토론을, 4 장에서는 결론 및 향후 연구 방향을 제시한다.

2. 제안된 무선 인터넷 프록시 서버 클러스터

그림 5 는 1.5 절에서 분석된 TranSend, All-in-one 및 CD, CD-A 무선 프록시의 문제점을 기반으로 이를 해결할 수 있도록 제안된 구조(이하 SSC: Systematic Scalability, Simple Structure, Cache Cooperation)이다. 제안된 구조에서는 All-in-one 에 사용된 기본 구조에서 그 모듈들 중에 Distiller 를 없애고 Cache 에 압축 기능을 추가한 CD-A 구조를 사용하였으며, LVS 를 사용하여 부하 분산을 하였다. 그래서 기존 무선 프록시들의 확장성 문제를 시스템적인(Systematic) 방법으로 해결하였고, 여러 모듈들 간의 복잡한 구성과 통신 구조도 Cache 에 압축 기능을 추가하여 단순화(Simplification)시켰다. 제안된 구조에서는 부하 분산 방식에서도 기존의 CD-A 에서 라운드 로빈(Round Robin) 방식을 이용하여 호스트간의 협동성이 없었던 점을 개선하기 위해 LVS 가 호스트를 선택할 때 Destination Hashing Scheduling 방식을 사용하여 선택하므로 캐시간 협동(Cache Cooperation)을 가능하게 하였다.

제안된 구조의 동작원리는 다음과 같다.

◦ 사용자(Client)가 제안된 프록시 구조(SSC)에 데이터를 요청한다.

◦ LVS 는 사용자의 요청한 수신주소(Destination IP)에 대해 Hash 를 이용하여 계산된 값에 따라 Host 를 선택한다.

◦ 선택된 호스트는 실제 웹 서버에 데이터를 요청한다.

◦ 호스트는 웹 서버로부터 받은 데이터를 LVS 를 통해 사용자에게 전송한다.

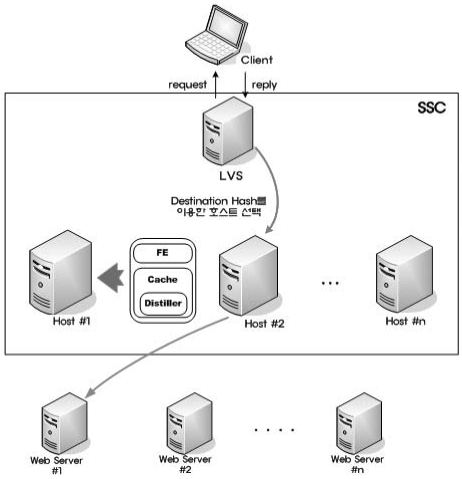

그림 5 제안된 구조 (SSC)

그림 6 는 제안된 구조의 사용자 요청 처리 순서를 나타낸 것이고 이를 요약하면 다음과 같다.

Step 1: 사용자(Client)가 LVS 에 데이터를 요청한다.

Step 2: LVS 가 사용자의 요청한 수신 주소(Destination IP)를 Hash 함수를 이용하여 고유한 호스트를 선택한다.

Step 3: FE 는 사용자가 요청한 데이터를 Cache 에 요청한다.

Step 4: Cache 는 요청된 데이터를 압축한다.

1) Cache 에 요청한 데이터가 없으면 외부의 웹서버에 요청한다.

2) 웹 서버는 Cache 에 데이터를 보내고, Cache 는 데이터를 압축한다.

Step 5: Cache 는 압축된 데이터를 저장한다.

Step 6: Cache 는 요청된 데이터를 FE 에 보낸다.

Step 7: FE 는 압축된 데이터를 LVS 에 보낸다.

Step 8: LVS 는 FE 로부터 받은 데이터를 사용자에게 요청에 대한 응답으로 보낸다.

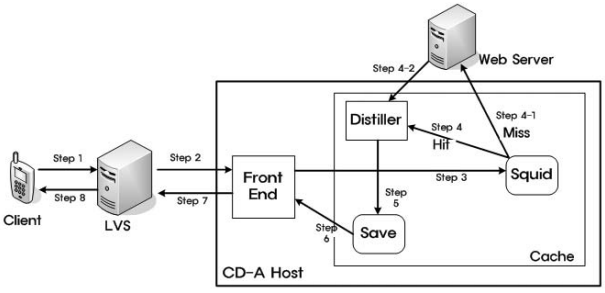

그림 6 제안된 구조(SSC)의 사용자 요청 처리 순서

표 2 는 기존 구조와 제안된 구조를 비교한 표이다.

표 2 기존 구조 vs. 제안된 구조 (SSC)

	TranSend	All-in-one	CD	CD-A	SSC
Scalability	No Systematic	Systematic	No Systematic	Systematic	Systematic
Simplification	X	X	O	O	O
Cache Cooperation	X	X	O	X	O

3. 실험 및 결과

3.1 실험 환경

표 3 은 실험에 사용된 하드웨어와 소프트웨어를 나타내고, 표 4 는 실험에 사용된 변수를 정리한 것이다.

표 3 실험용 하드웨어 & 소프트웨어

		Hardware		Software	#
		CPU (Hz)	RAM (MB)		
Client	TranSend, All-in-one, SSC	P-III 700 M	128	Apache Bench	1
	CD, CD-A	P-IV 1.7 G	512		
LVS		P-IV 2.4 G	512	NAT	1
Host	Cache	P-II 400 M	256	Squid	16
	Distiller			JPEG-6b	

표 4 실험에 사용된 변수

사용자의 요청 개수	◦ 약 200 초 동안 프록시가 처리할 수 있는 최대 개수
요청 이미지	◦ JPEG
요청 크기	◦ 300 bytes, 1 K, 10 K, 100 Kbytes, Variation
사용자 정보(Preference)	◦ 이미지 Quality = 중간
웹 서버	◦ Cache 서버 자체에 둠 (프록시내의 성능 평가에 초점을 맞춤)

3.2 실험 결과

(1) TranSend & All-in-one

그림 7 은 TranSend 와 All-in-one 에서 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다.

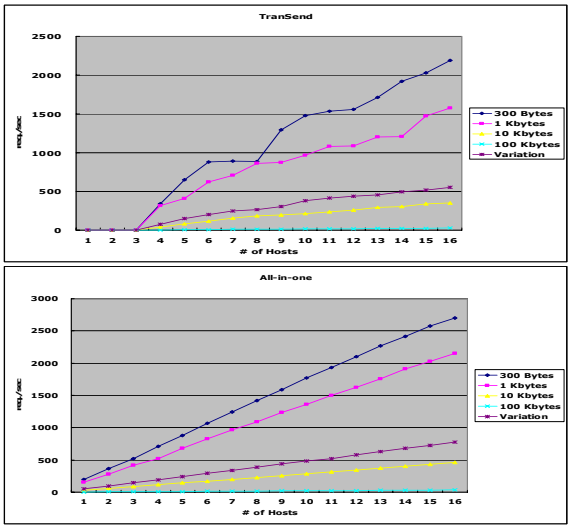

그림 7 TranSend & All-in-one

(2) CD & CD-A

그림 8 은 CD 와 CD-A 에서 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다.

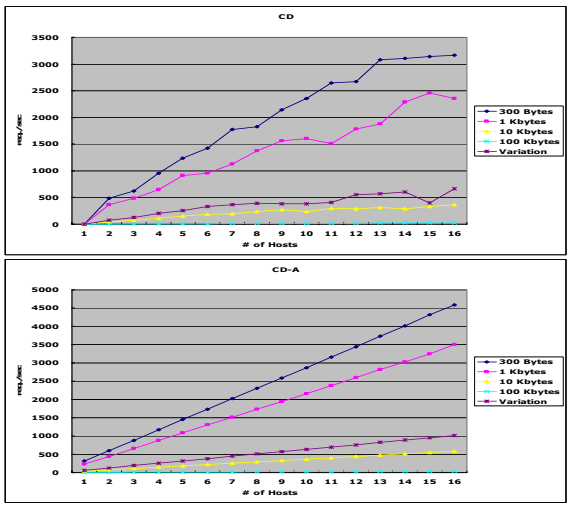

그림 8 CD & CD-A

(3) SSC

그림 9 는 제안된 구조(SSC)에서 이미지 크기를 다르게 요청했을 경우 호스트 개수에 따른 초당 요청수를 나타낸다.

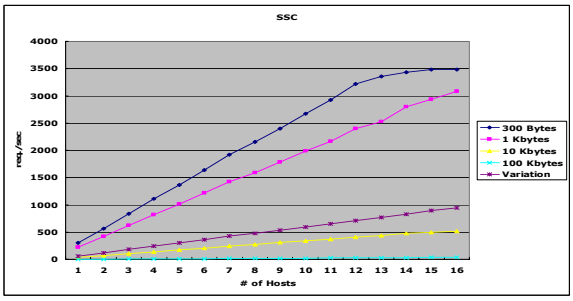

그림 9 제안된 구조 (SSC)

(4) 기존 구조 vs. 제안된 구조 (SSC)

표 5 는 기존 구조에 대한 SSC 의 평균 성능 향상률을 나타낸 것이다. 제안된 시스템은 TranSend, All-in-one, CD 에 비해 성능이 향상되었으나 CD-A 에 비해서는 성능이 감소되었다. 이는 프록시들을 스케줄링하는 방식의 차이로 설명할 수 있다. 즉, CD-A 는 프록시의 캐시간 협동성을 보장하지 않는 Round Robin 방식으로 스케줄링한 반면에 SSC 에서는 캐시간 협동성을 보장하기 위해 매 사용자 요청마다 Hash 를 계산하는 Destination Hash 스케줄링 방식을 사용함으로 이에 따른 처리 속도 지연을 가진다.

표 5 평균 성능 향상률 (SSC)

%	300 Bytes	1 Kbytes	10 Kbytes	100 Kbytes	Vari.	Avg.
TranSend	104.20	112.05	76.51	78.45	84.59	91.16
All-in-one	50.37	46.95	19.35	11.16	24.75	30.52
CD	14.43	24.79	33.70	27.70	40.91	28.31
CD-A	-8.54	-7.54	-6.25	-4.66	-5.70	-6.54

4. 결론

본 논문에서는 무선 인터넷의 근본적인 문제점 중 일부를 해결할 수 있도록 제안된 TranSend 와 이의 개선 구조인 All-in-one, CD 및 CD-A 프록시 서버의 문제점을 확장성, 복잡성, 캐시간 협동성 관점에서 분석하였다. 그리고 시스템적인 확장성을 보장하고, 단순화된 통신구조 및 캐시간 협동성을 가지는 새로운 구조를 제안하였다. 실험을 통해 제안된 구조가 성능 향상에 기여했음을 확인하였다.

제안된 구조의 단점 및 향후 연구 방향을 요약하면 다음과 같다.

◦ Hash Scheduling 의 side-effect 최소화: Hash 를 사용하면 캐시간 협동성을 보장하나 하나의 프록시(캐시)로 요청이 집중되어 전체 성능이 집중된 일부 프록시(캐시)에 종속되는 단점을 가진다. 이를 해결하기 위해 Spatial hash-joins[7]의 적용을 고려해 볼 수 있다.

◦ LVS 부하 최소화: Hash Scheduling 방식은 사용자의 매 요청마다 Hash 를 계산하여 프록시를 선택하는 방식임으로 다른 스케줄링 방식에 비해 LVS 에 더 많은 부하를 주게 된다. 이를 해결하기 위해 각 요청에 대한 Destination Hash 값을 저장하여 동일 Destination 에 대한 요청이 올 경우 기존에 저장된 Hash 값을 사용하면 LVS 의 계산 부하를 최소화할 수 있다.

참고문헌

[1] A. Fox, "A Framework for Separating Server Scalability and Availability from Internet Application Functionality," Ph. D. dissertation, U. C. Berkeley, 1998.

[2] H. Kwak, J. Woo, Y. Jung, D. Kim, and K. Chung, "A Clustering based Wireless Internet Proxy Server," Journal of KISS : Information Networking, Vol. 31, No. 1, Feb. 2004.

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

[4] H. Kwak, K. Han, and K. Chung, "The Structure Improvement of a Clustering based Wireless Internet Proxy Server," The 14th Joint Conference on Communications and Information, under review.

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

[6] H. Kwak, K. Han, and K. Chung, "The Performance Improvement of a Clustering based Wireless Internet Proxy Server," The 31th Spring Conference on KISS, under review.

[7] M.-L. Lo and C. V. Ravishankar, "Spatial hash-joins," In SIGMOD, Montreal, Canada, 1996.