확장성 있는 무선 인터넷 프록시 서버를 위한 스케줄링 알고리즘 비교

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

A Comparison of Scheduling Algorithms for Scalable Wireless Internet Proxy Servers

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

요 약

무선 인터넷 프록시 서버는 캐싱과 압축 기능을 이용하여 무선 사용자와 기존 유선 인터넷을 연결해주는 역할을 수행한다. 이때 무선 인터넷 프록시 서버는 대용량 트래픽에 대한 확장성을 가지고 클러스터링으로 구성될 수 있으며 여러 가지 상황을 고려하여 최적의 스케줄링 알고리즘을 사용할 수 있다. 이에 본 논문에서는 확장성을 위해 구성된 클러스터링 기반의 무선 인터넷 프록시 서버에 적용될 수 있는 스케줄링 알고리즘을 비교하여 사용자의 요청 방식에 따른 최적의 스케줄링 알고리즘을 찾고자 한다. 15 개의 컴퓨터를 사용하여 실험을 수행하였고, 실험 결과 사용자가 동일 이미지를 요청한 경우에는 Round-Robin 방식이 가장 좋은 성능을, 다른 이미지를 요청한 경우에는 Least-Connection 방식이 가장 좋은 성능을 보임을 확인하였다.

1. 서론

기존 유선 인터넷과는 달리 무선 인터넷은 사용자의 위치에 무관하게 인터넷을 이용할 수 있는 연속성(Continuity)으로 인해 급속한 성장을 하고 있다. 이러한 무선 인터넷은 언제 어디서나 이용할 수 있다는 장점과 더불어 기술과 환경상의 많은 제약을 가진다. 무선 인터넷 프록시 서버는 무선 사용자와 유선 인터넷을 연결해주며 캐싱과 압축 기능을 이용하여 무선 인터넷이 가지는 제약점을 보완하고 있다. 그림 1 은 무선과 유선을 연결해주는 무선 인터넷 프록시 서버를 나타낸다.

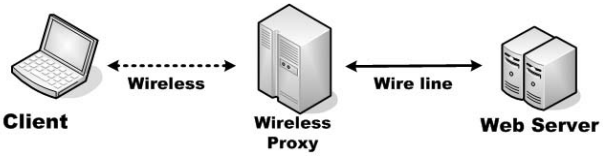

그림 1 무선 인터넷 프록시 서버

무선 인터넷 프록시 서버는 사용자의 급격한 증가에 대응할 수 있는 확장성을 가져야 하며, 이러한 확장성을 위해 클러스터링 방식을 이용한다. 클러스터링 기반의 무선 인터넷 프록시 서버에 대한 설명은 다음과 같다.

1.1 클러스터링 기반의 무선 인터넷 프록시 서버[1, 2]

무선 인터넷 프록시는 캐싱, 압축, 대용량 트래픽에 대한 확장성을 고려하여야 한다. 이러한 관점의 기존 무선 프록시 서버 중 하나인 TranSend[1]는 대용량 트래픽에 대한 확장성을 고려하여 클러스터링으로 구현된 무선 프록시 서버이다. 본 논문에서는 TranSend 를 확장성과 구조적인 관점에서 개선한 CD-A(CD & All-in-one)[2] 구조를 사용하였다. 그림 2 는 CD-A 프록시 시스템의 전체적인 구조를 나타낸다.

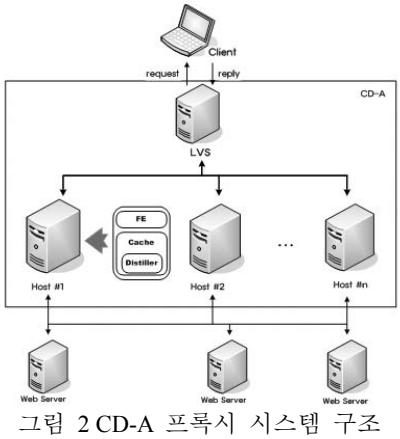

그림 2 CD-A 프록시 시스템 구조

CD-A 구조는 부하 분산을 위한 LVS[3]와 CD-A 호스트들로 구성되며 호스트의 각 모듈로서 FE 와 CD 가 구성된다. LVS 는 클라이언트의 요청을 받아서 각 호스트들에게 전달하는 역할을 담당하며, FE 는 호스트 내에서 클라이언트의 요청에 대한 외부 인터페이스를 담당한다. CD 는 클라이언트의 요청을 처리하는 Cache 와 데이터에 대한 압축을 수행하는 Distiller 의 역할을 함께 수행한다.

LVS 가 스케줄링 알고리즘을 통하여 클라이언트의 요청을 각 호스트에 전달하게 되면, 호스트 내의 FE 는 그 요청을 CD 에 전달한다. CD 는 해당 데이터가 Cache 에 존재하면 FE 에 전달하고, 존재하지 않으면 웹서버로부터 데이터를 요청하여 얻어온 후 압축과정을 거쳐 FE 에 전달하고 FE 는 그 데이터를 클라이언트에 보내는 방법으로 동작한다.

1.2 LVS[3]

LVS(Linux Virtual Server)는 리눅스 기반의 웹 클러스터링을 이용하는 고성능, 고가용성의 서버로서 실제 서버를 클러스터로 구성하여 뛰어난 확장성과 신뢰성을 제공한다. 가상 서버는 독립된 서버들의 클러스터로 구축되어 있어서 확장 가능하고 가용성이 높다. 클러스터 구조는 클러스터 밖의 클라이언트에게는 드러나지 않고(transparent) 마치 클러스터가 하나의 고성능 서버인 것처럼 상호작용한다. 그림 3 에서는 가상 서버의 전체적인 구조를 나타낸다.

실제 서버들은 고속 LAN 또는 지역적으로 분산된 WAN 으로 연결되며 실제 서버들의 외부 연결부는 가상 서버이다. 가상 서버는 클라이언트의 요청을 각기 다른 서버들에게 할당(schedule)하고 클러스터링된 서버들을 IP 주소상의 하나의 가상 서버로 보이도록 만든다. 가상 서버는 사용자(Client)로부터 받은 요청을 NAT, Direct Routing 또는 IP Tunneling 세 가지 방법으로 처리한다.

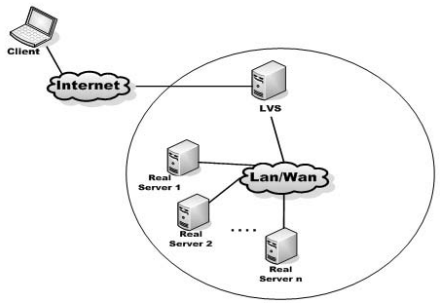

그림 3 가상 서버의 구조

본 논문에서는 LVS 를 이용하여 무선 인터넷 프록시 서버를 클러스터링으로 구성할 때 사용되는 스케줄링 방식을 비교한다. 이러한 비교를 통해 사용자의 요청 방식에 따른 최적의 스케줄링 방식을 찾고자 한다. 본 논문의 구성은 다음과 같다. 2 장에서는 LVS 의 스케줄링 알고리즘들을 비교한다. 3 장에서는 각 스케줄링 알고리즘의 실험 및 결과를 나타내고, 4 장에서는 결론 및 향후 연구 방향을 제시한다.

2. 스케줄링 알고리즘[4]

LVS 는 클라이언트로부터 받은 요청을 스케줄링 알고리즘을 사용하여 분산시킨다. LVS 의 스케줄링 알고리즘은 다음과 같다.

• RR(Round-Robin) Scheduling : 라운드 로빈 방식은 클라이언트로부터 오는 요청에 대해 순차적으로 다음 서버에 연결하는 것이다. 그림 4 는 Round-Robin 알고리즘을 나타낸다.

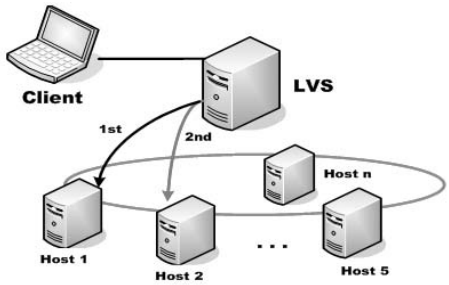

그림 4 Round-Robin Algorithm

• WRR(Weighted Round-Robin) Scheduling : 라운드 로빈 방식의 일종으로 실제 서버에 서로 다른 처리용량(weight)을 지정할 수 있다. 각 서버에 가중치를 부여할 수 있으며, 여기서 지정한 정수 값을 통해 처리 용량을 정한다. 그림 5 는 WRR 알고리즘을 나타낸다.

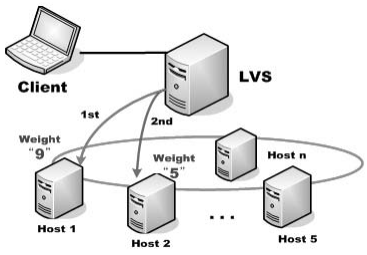

그림 5 Weighted Round-Robin Algorithm

• LC(Least-Connection) Scheduling : LC 방식은 실제서버들의 연결 개수에 따라 요청을 보내는 서버를 결정하며, 가장 연결이 적은 서버에 클라이언트의 요청을 보낸다. 그림 6 은 Least-Connection 알고리즘을 나타낸다.

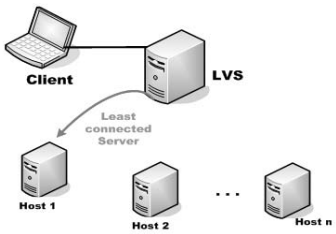

그림 6 Least-Connected Algorithm

• WLC(Weighted Least-Connection) Scheduling : Least-Connection 스케줄링의 일종으로 각각의 실제 서버에 성능 가중치를 부여할 수 있다. 그림 7 은 Weighted Least-Connection 알고리즘을 나타낸다.

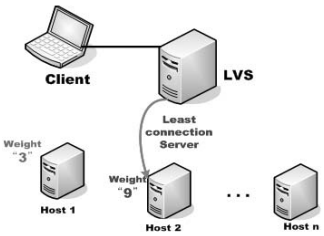

그림 7 Weighted Least-Connection Algorithm

• LBLC(Locality-Based Least-Connection) Scheduling : LBLC 는 일반적으로 클라이언트로부터 요청이 왔을 때 가상서버는 수신하는 목적지 IP 에 해당하는 실제 서버로 요청을 분산한다. 그러나 그 실제 서버에 부하가 많다면 다른 서버 중 부하가 1/2 보다 적은 서버를 찾아 요청을 분산한다.

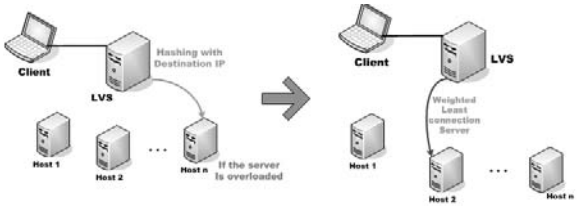

그림 8 Locality-Based Least-Connection Scheduling

• LBLCR(Locality-Based Least-Connection Scheduling with Replication) Scheduling : 목적지 IP 에 대한 서버 그룹(server set)을 할당한다. 이 서버 그룹의 모든 호스트가 과부하 상태가 되면 다른 호스트들 중 가장 적은 연결 개수를 가지는 호스트를 추가하고 서버 그룹내의 가장 높은 부하를 가지는 호스트를 삭제한다.

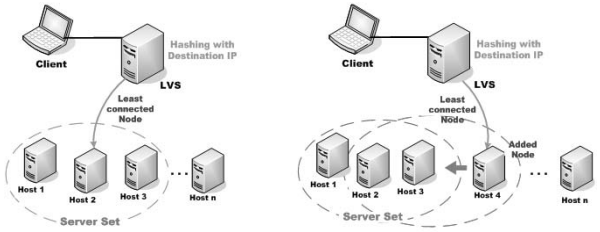

그림 9 LBLCR Algorithm

• DH(Destination Hashing) Scheduling : 가상 서버는 수신 IP 주소를 보고 Hash 테이블을 통해 해당하는 서버로 요청을 분산시킨다. 그림 10 은 Destination Hashing 알고리즘을 나타내고 있다.

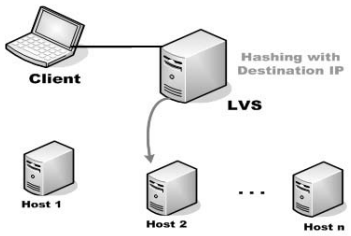

그림 10 Destination Hashing Algorithm

• SH(Source Hashing) Scheduling : 가상 서버가 송신 IP 주소를 보고 Hash 테이블을 통해 해당되는 서버로 요청을 분산시킨다. 그림 11 은 Source Hashing 알고리즘을 나타내고 있다.

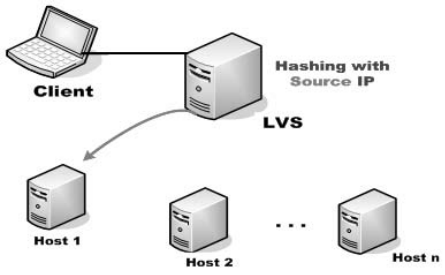

그림 11 Source Hashing Algorithm

3. 실험

3.1 실험 환경

표 1 은 실험에 사용된 하드웨어와 소프트웨어를 나타낸다. 프록시 서버는 PC 15 대로 구성되어 있고 LVS 를 사용하여 부하 분산을 하였다. Apache Bench 라는 프로그램을 Client 에서 수행하여 프록시 서버에 영상(이미지)을 요청하는 방식으로 실험하였다. 표에서 Client 와 LVS 가 Host 보다 하드웨어 성능이 좋은 이유는 확장성 실험을 할 때 Client 와 LVS 에서는 병목이 발생하지 않는 상황에서 프록시 서버 내 호스트들 사이의 부하분산을 확인하고자 했기 때문이다.

표 1 실험용 하드웨어 & 소프트웨어

		하드웨어		소프트웨어	개수
		CPU (Hz)	RAM (MB)		
Client		P-III 700 M	128	AB[5]	1
LVS		P-IV 2.4 G	512	NAT[6]	1
Host	Cache	P-II 400 M	256	Squid[7]	15
	Distiller			JPEG-6b[8]	

3.2 실험 방법

표 2 는 실험에 사용된 변수들을 정리한 것이다. 다른 이미지를 요청하는 실험의 경우 극단적 상황을 고려하여 각각에 호스트에서 처리할 수 있는 이미지 크기를 고정하여 실험하였다. 예를 들어, 1 번 호스트에는 300 bytes 만을 처리하고, 2 번 호스트는 1 Kbytes 만을 처리하도록 구성하였다. 스케줄링 알고리즘의 경우 실험에 사용된 호스트의 성능이 동일하여 WRR, WLC, LBLCR 을 제외하였고, 클라이언트가 하나인 점을 고려하여 SH 도 제외하였다.

표 2 실험에 사용된 변수

사용자의 요청 개수		◦ 약 200 초 동안 프록시가 처리할 수 있는 최대 개수
요청 이미지		◦ JPEG[9]
요청 크기		◦ 300 bytes, 1 K, 10 K, 100 Kbytes, Variation[10]
요청 방식	동일 이미지	◦ 동일 크기의 이미지를 요청
	다른 이미지	◦ 각기 다른 크기의 이미지를 각각의 서버에 분산하고 이를 요청
사용자 정보(Preference)		◦ 이미지 Quality = 중간
웹 서버		◦ Cache 서버 자체에 둠 (프록시내의 성능 평가에 초점을 맞춤)
스케줄링 알고리즘		◦ 4 가지 : RR, LC, LBLC, DH

3.3 실험 결과

(1) 동일 크기 이미지를 요청한 경우

표 3 은 이미지 크기를 동일하게 요청했을 때 호스트 개수에 따른 초당 요청수를 나타낸다.

표 3 호스트 개수에 따른 초당 요청 수

(a) 300 bytes

Algorithms	1	5	10	15
RR	318	1453	2868	4322
LC	304	1378	2684	3343
LBLC	300	1166	1879	2446
DH	305	1367	2675	3479

(b) 1k bytes

Algorithms	1	5	10	15
RR	231	1089	2162	3244
LC	213	1027	2017	2985
LBLC	211	1004	1625	2270
DH	220	1011	1991	2938

(c) 10k bytes

Algorithms	1	5	10	15
RR	36	182	363	545
LC	33	172	345	518
LBLC	34	160	218	282
DH	35	172	342	499

(d) 100k bytes

Algorithms	1	5	10	15
RR	2.30	11.91	23.92	35.99
LC	2.24	11.35	22.81	34.29
LBLC	2.24	10.95	21.34	32.11
DH	2.25	11.35	22.80	34.29

(e) Variation

Algorithms	1	5	10	15
RR	65	321	637	965
LC	62	305	605	880
LBLC	61	267	406	444
DH	63	302	597	895

사용자가 동일 크기의 이미지를 요청한 경우에는 RR 이 가장 좋은 성능을 보이고 LBLC 가 가장 낮은 성능을 보인다. 이를 정리하면 다음과 같다.

- RR : 호스트를 선택할 때 순차적으로 요청을 분배함으로 가장 좋은 성능을 보인다.
- LC : 매 요청마다 호스트를 선택할 때 가장 적은 부하(연결수)를 가지는 호스트를 선택함으로 RR 보다 성능이 떨어지나 다른 두가지 방식에 비해서는 좋은 성능을 가진다.
- DH : 매 요청마다 해쉬를 계산하여 호스트를 선택함으로 RR 이나 LC 에 비해 낮은 성능을 가진다.
- LBLC : 매 요청마다 해쉬를 계산하고 자신의 부하가 과중시에 다른 호스트를 선택함으로 4 가지 방식중에 가장 낮은 성능을 가진다.

(2) 다른 크기의 이미지를 요청한 경우

표 4 는 이미지 크기가 다른 환경에서 호스트별로 다른 크기의 이미지를 같은 이름의 데이터로 만들어 각 호스트에 저장하여 실험하였다. 1 번 호스트부터 300bytes, 1K, 10K, 100K, Variation 으로 설정하고 다시 6 번 호스트부터 1 번 호스트처럼 300bytes 를 할당하는 방식으로 15 번 호스트까지 같은 이름으로 데이터를 할당하였다. 이 실험은 동일한 처리 능력의 호스트들에게서 동시에 동일 이름으로 다른 크기의 데이터를 요청할 때 각 호스트의 부하에 따른 초당 요청 수를 스케줄링 알고리즘별로 비교하는 것이다.

표 4 알고리즘별 호스트 개수에 따른 초당 요청 수

Algorithms	1	5	10	15
RR	12.43	18.88	29.6	41.53
LC	11.49	246	369	409
LBLC	10.29	14.95	37.06	57.64
DH	11.49	18.29	28.83	37.73

사용자가 다른 크기의 이미지를 요청한 경우에는 LC 가 가장 좋은 성능을 보이고 DH 가 가장 낮은 성능을 보인다. 이를 정리하면 다음과 같다.

- LC : 호스트를 선택할 때 가장 적은 연결 개수를 가지는 호스트를 선택함으로 빨리 처리하는 300 bytes 로 요청이 몰릴 것이고 최종 결과는 이에 종속된다.
- LBLC : 해쉬를 계산하는 시간이 있으나 요청이 몰릴 경우 적은 부하를 가지는 다른 호스트를 선택함으로 RR 이나 DH 에 비해 좋은 성능을 가진다.
- RR : 사용자의 요청 크기에 상관없이 호스트에 요청함으로 최종 결과는 가장 적은 성능을 가지는 호스트(100 Kbytes 만을 처리하는 호스트)의 결과에 종속된다.
- DH : RR 과 비슷한 성능을 보이나 호스트를 선택할 때 해쉬 계산 시간이 추가된다.

(3) 각 방식의 비교

실험 결과를 토대로 실험에 사용된 각 스케줄링 방식을 비교하면 표 5 와 같다.

표 5 각 스케줄링 방식의 비교

알고리즘	서버 상황 고려	캐시 간 협동	동일 이미지	다른 이미지	분산 부하	비 고
RR	×	×	1	3	없음	서버의 상황에 무관하게 부하 분산
LC	○	×	2	1	매 분산마다 가장 적은 부하를 선택	서버의 상황을 고려하여 부하 분산
LBLC	○	○	4	2	매 분산마다 해쉬를 계산 + 부하 과중시 다른 서버 선택	해쉬 함수를 사용하여 캐시간 협동성을 보장하고 부하 과중시에 다른 서버 선택
DH	×	○	3	4	매 분산마다 해쉬를 계산	해쉬 함수를 사용하여 캐시간 협동성을 보장

4. 결 론

본 논문에서는 대용량 트래픽을 처리할 수 있는 클러스터링 기반의 무선 인터넷 프록시에서 사용될 수 있는 스케줄링 알고리즘을 사용자의 요청 방식에 따라 비교하였다. 사용자의 요청 이미지가 동일할 경우에는 Round-Robin 방식이, 요청 이미지가 다른 경우에는 Least-Connection 방식이 가장 좋은 성능을 가짐을 확인하였다.

향후 연구 방향을 요약하면 다음과 같다.

- 호스트 부하 정보에 기반 한 스케줄링 적용 : 기존 스케줄링 알고리즘의 경우 호스트의 부하 정보에 무관한 방식으로 인해 최종 결과가 일부 호스트에 종속되는 결과를 가진다. 그러나 호스트의 부하 정보를 고려하여 스케줄링을 하게 되면 사용자의 요청 방식에 무관하게 좋은 성능을 가지게 된다.

참고문헌

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

[2] 곽후근, 한경식, 정규식, “클러스터링 기반의 무선 인터넷 프록시 서버 성능 개선”, 한국정보과학회 춘계학술대회, 2004. 심사중

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

[4] Virtual Server Scheduling Algorithms, http://www.linuxvirtualserver.org/docs/Scheduling.html

[5] AB(Apache Bench), http://www.apache.org

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

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

[8] 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.

[9] 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.

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