무선 인터넷 프록시 서버 클러스터 환경에서 호스트 부하 정보에 기반한 동적 스케줄링

박홍주° 곽후근 정규식
송실대학교 정보통신전자공학부
{phj0520°, gobarian, kchung}@q.ssu.ac.kr

Dynamic Scheduling based on Host Load Information in a Wireless Internet Proxy Server Cluster Environment

Hongjoo Park° Hukeun Kwak Kyusik Chung
School of Electronics Engineering, Soongsil University

요 약

우선 인터넷 프록시 서버 클러스터에서 부하 분산기는 사용자의 요청을 각 서버(호스트)로 분산시키는 역할을 한다. 리눅스 가상 서버(LVS: Linux Virtual Server)는 소프트웨어적으로 사용되는 부하 분산기로서 여러 가지 스케줄링 방식들을 가지고 있다. 그러나 부하 분산시에 서버(호스트)의 유동적인 부하 정보를 반영하지 못하는 단점이 있다. 이에 개선된 방식으로 서버의 동시 연결 개수에 따라 상한계(Upper Bound)와 하한계(Lower Bound)를 설정하고, 요청을 분산하는 동적 스케줄링(Dynamic Scheduling)이 존재한다. 그러나 사용자의 요청 컨텐츠에 따라 상한계와 하한계가 바뀔 수 있음에도 불구하고 이 값들이 고정되어 있다는 단점을 가진다.

본 논문에서는 호스트 부하 정보에 기반한 스케줄링 방식을 제안한다. 제안된 방식은 호스트의 부하 정보를 바탕으로 사용자의 요청을 분산하였으며, 사용자의 요청에 따라 상한계와 하한계가 바뀔 수 있음을 고려하여 상한계와 하한계를 설정하지 않고 사용자 요청 컨텐츠에 따라 적절하게 요청이 분배되도록 하였다. 16대의 컴퓨터를 사용하여 실험을 수행하였으며, 실험 결과 사용자가 요청하는 컨텐츠가 동일한 경우에는 기존 스케줄링 방식과 13% 성능 감소를, 다른 경우에는 기존 스케줄링 방식보다 102%의 성능 향상을 보임을 확인하였다.

1. 서 론

우선 인터넷 서비스는 기존의 정보검색 위주의 간단한 서비스에서 전자 상거래나 멀티미디어 서비스 등의 복잡한 서비스로 사용자들의 욕구가 상승하고 있다. 이에 따라 우선 인터넷 대역폭의 효율적인 사용과 빠른 이용 시간을 위하여 우선 인터넷 프록시 서버의 필요성이 증대되고 있다. 우선 인터넷 프록시 서버는 우선 사용자를 유선 인터넷 서버에 연결 시켜주는 역할을 한다. 그림 1은 우선 인터넷에 사용되는 우선 인터넷 프록시 서버를 나타내고 있다.

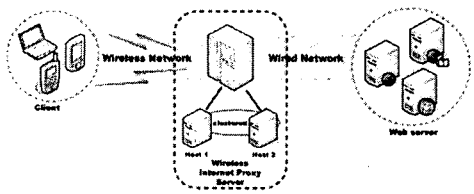


그림 1 우선 인터넷 프록시 서버

우선 인터넷 프록시 서버는 급증하는 사용자의 요청에 대한 확장성(Scalability)을 보장하기 위해 클러스터링 구조를 가진다. 우선 인터넷 프록시 서버 클러스터는 고성능 및 고가용성의 효과를 가진다.

1.1 우선 인터넷 프록시 서버 클러스터[2]

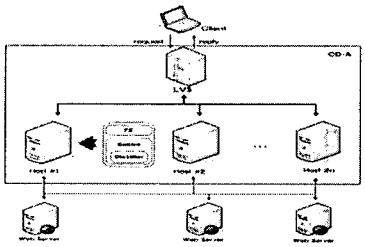


그림 2 CD-A 프록시 시스템 구조

우선 인터넷 프록시 서버는 우선 인터넷의 낮은 대역폭 해결하기 위해 캐싱(Caching)[3]과 압축(Distillation)[4]을 사용하며, 대용량 트래픽에 대한 확장성(Scalability)이 고려되어야 한다. TranSend[5]는 대용량 트래픽에 대한 확장성을 고려하여 클러스터링으로 구현된 무선 프록시 서버이다. 본 논문에서는 TranSend를 확장성과 구조적인 관점에서 개선한 CD-A(CD & All-in-one)구조를 사용하였다. 그림 2는 CD-A 프록시 시스템의 전체적인 구조를 나타낸다.

1.2 리눅스 가상 서버(LVS: Linux Virtual Server)[1]

리눅스 가상 서버(LVS: Linux Virtual Server)는 리눅스를 기반으로 독립된 여러 서버들을 하나의 클러스터로 구성하여 뛰어난 확장성과 가용성을 제공한다. 그림 3은 리눅스 가상 서버의 전체적인 구조를 나타낸다.

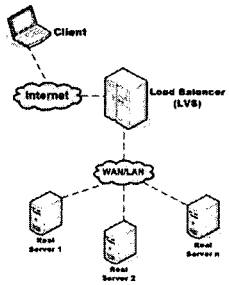


그림 3 리눅스 가상 서버의 구조

2. 기존 연구

2.1 LVS 스케줄링[6]

LVS는 클라이언트로부터 받은 요청을 8가지 스케줄링 방식을 이용하여 분산시킨다. 각각의 스케줄링 방식은 다음과 같다.

(1) RR(Round-Robin) & WRR(Weighted Round-Robin)

RR은 클라이언트로부터 오는 요청을 순서대로 서로 다른 서버로 보내고, WRR은 서버의 처리 용량이 다를 때, 각 서버의 처리 용량에 비례하는 가중치를 두어 요청을 분산한다.

(2) LC(Least Connection) & WLC(Weight Least Connection)

LC는 연결된 개수를 측정하여 가장 작은 연결 개수를 가지는 서버로 요청을 분산하고, WLC는 각 서버의 처리 용량에 따라 가중치(Weight)를 할당하고 이 가중치에 따라 요청을 할당한다. 가중치가 큰 서버로 더 많은 요청을 할당 하게 된다.

(3) LBLC(Locality-Based Least Connection)

LBLC 알고리즘은 일단 요청이 들어오면 부하가 적은 서버로 요청을 할당한다. 그러나 만약 이 서버가 과부하가 되면, 다른 서버들 중에서 자신의 가중치보다 1/2보다 적은 부하를 가진 서버를 선택하고 요청을 이 서버에 할당한다.

(4) LBLCR(Locality-Based Least Connection-Replicatio)

LBLCR은 서버 그룹(Set)을 할당한다. 서버 그룹 내에서는 가장 연결 개수가 작은 서버로 요청이 할당된다. 서버 그룹 내의 모든 서버가 과

부하라면, 다른 서버들 중 가장 연결 개수가 작은 서버를 그룹에 포함한다.

(5) DH(Destination Hashing) & SH(Source Hashing)

DH는 수신(Destination) IP 주소를 가지고 Static Hashing 테이블을 통해 매핑된 서버로 요청을 보내고, SH는 송신(Source) IP 주소를 가지고 Static Hashing 테이블을 통해 매핑된 서버로 요청을 보낸다.

2.2 동적 스케줄링(Dynamic Scheduling)[7]

동적 스케줄링은 기존 LVS 스케줄링의 단점을 보완하기 위해 제안된 스케줄링 알고리즘이다. 서버(호스트)의 동시 연결 개수를 이용하여 부하를 분산하는 방식으로 Throttling Mechanism[8]을 스케줄링에 적용하였다. Throttling Mechanism은 상한계(Upper Bound)와 하한계(Lower Bound)를 두고 상한계를 넘어선 서버에게는 더 이상 작업을 할당하지 않고 하한계로 떨어지면 다시 작업을 할당하는 방식으로 동작하게 된다.

2.3 접근 방식

(1) LVS 스케줄링의 단점

LVS 스케줄링 알고리즘이 가지는 단점을 정리하면 표 1과 같다.

표 1 LVS 스케줄링 알고리즘의 단점

스케줄링	단 점
RR	서버의 처리 능력이나 요청 컨텐츠가 다른 경우 이를 반영하지 못한다.
WRR	가중치는 사용자의 요청 컨텐츠에 따라서 유동적으로 바뀔 수 있고 WRR은 이를 반영하지 못한다.
LC	사용자의 요청 컨텐츠가 다른 경우 이를 반영하지 못한다.
WLC	가중치는 사용자의 요청 컨텐츠에 따라서 유동적으로 바뀔 수 있고 WLC는 이를 반영하지 못한다.
LBLC	DH와 WLC를 결합한 방식으로 동작함으로 이들의 단점을 그대로 가진다.
LBLCR	서버들이 그룹화 되어 있다는 점을 제외하고 DH와 WLC를 결합한 방식으로 동작함으로 이들의 단점을 그대로 가진다.
DH	서버의 처리 능력 혹은 사용자의 요청이 다른 경우 이를 반영하지 못한다.
SH	서버의 처리 능력 혹은 사용자의 요청이 다른 경우 이를 반영하지 못한다.

(2) 동적 스케줄링의 단점

•사전 실험을 통하여 상한계와 하한계를 설정해주어야 한다.

•상한계와 하한계는 사전 실험으로 고정되어 있어 서버의 처리 능력이 바뀔 때마다 이를 다시 수정해주어야 한다.

•사용자가 서로 다른 컨텐츠를 요청하는 경우 상한계와 하한계가 고정되어 있어 이를 반영하지 못한다.

(3) 본 연구의 접근 방식

제안된 알고리즘은 호스트의 부하 정보를 이용하여 사전 실험 없이 서버의 처리 능력과 사용자의 요청 컨텐츠가 다른 경우를 능동적으로 반영하는 방식이다.

3. 호스트 부하 정보에 기반한 동적 스케줄 (DS-HLI)

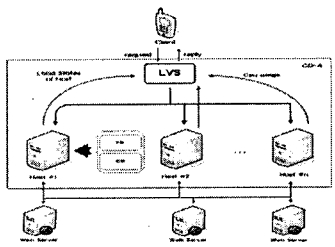


그림 4 무선 인터넷 프록시 서버 클러스터(CD-A)와 제안된 스케줄링 알고리즘

그림 4는 2.3절에서 언급한 기존 LVS 스케줄링과 동적 스케줄링의 문제점을 기반으로 이를 해결할 수 있도록 제안된 알고리즘을 나타낸다. 제안된 알고리즘은 기존의 무선 인터넷 프록시 서버 클러스터인 TranSend를 개선한 CD-A 구조에 적용하였고 이에 대해 자세히 기술하면 표 2와 같다.

표 2 호스트 부하 정보에 기반한 동적 스케줄링

단계	설명
1	서버(호스트)들은 LVS에게 주기적으로 자신의 부하 정보를 보낸다.
2	LVS는 서버(호스트)들로부터 받은 부하 정보를 기반으로 각각의 서버들에 대한 가중치(Weight) 테이블을 갱신한다.
3	사용자(Client)가 무선 인터넷 프록시 서버 클러스터(CD-A)로 컨텐츠를 요청한다.
4	사용자 요청을 받은 LVS(외부 인터페이스)는 현재 각각의 서버들에 대한 가중치 테이블을 검사하여 가장 많은 가중치를 가지는 서버로 요청을 분산한다.

그림 5은 전체적인 사용자 요청 처리 순서를 나타내며 각 단계를 요약하면 표 3와 같다.

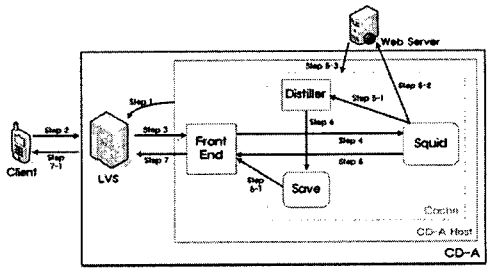


그림 5 사용자 요청 처리 순서

표 3 사용자 요청 처리 순서

단계	설명
1	서버(호스트)들은 LVS에게 주기적으로 자신의 부하 정보를 보내고 LVS는 서버(호스트)들로부터 받은 부하 정보를 기반으로 각각의 서버들에 대한 가중치(Weight) 테이블을 갱신한다.
2	사용자(Client)가 무선 인터넷 프록시 서버 클러스터(CD-A)로 컨텐츠를 요청한다.
3	사용자 요청을 받은 LVS(외부 인터페이스)는 현재 각각의 서버들에 대한 가중치 테이블을 검사하여 가장 많은 가중치를 가지는 서버로 요청을 분산한다.
4	사용자 요청을 받은 서버 내 FE는 요청한 컨텐츠를 캐시에 요청한다.
5	캐시에 요청된 컨텐츠가 있고 압축된 형태라면 바로 FE로 응답하고, 압축된 형태가 아니라면 압축기로 보낸다. 만약, 요청한 데이터가 없다면 외부 웹 서버로 컨텐츠를 요청하고 이를 캐시에 저장 후 압축기로 보낸다.
6	캐시는 압축된 데이터를 저장하고 이를 FE로 보낸다.
7	FE는 압축된 데이터를 LVS로 보내고, LVS는 FE로부터 받은 데이터를 사용자에게 요청에 대한 응답으로 보낸다.

4. 실험 및 토론

4.1 실험 환경

표 4는 실험에 사용된 하드웨어와 소프트웨어를 나타낸다.

표 4 실험용 하드웨어 & 소프트웨어

		하드웨어		소프트웨어	개수
		CPU (Hz)	RAM (MB)		
사용자		P-IV 2.26 G	256	AB[9]	1
LVS		P-IV 2.4 G	512	NAT[10]	1
서버	캐시	P-II 400 M	256	Squid[11]	16
	압축기			JPEG-6b[12]	

. 4.2 실험 방법

실험은 아래의 6 단계를 9개의 스케줄링 알고리즘에 반복 적용하였다.

1. 무선 인터넷 프록시 서버(CD-A) 1대를 구성한다.
2. 실험에 사용할 스케줄링 알고리즘을 세팅한다.
3. AB(Apache Bench)를 이용하여 서버로 JPEG 이미지를 약 200초 동안 요청한다.
4. 초당 요청 개수를 측정한다.
5. 서버를 1대 추가한다.
6. 실험에 사용된 서버의 수(16대)만큼 3 ~5 과정을 반복한다.

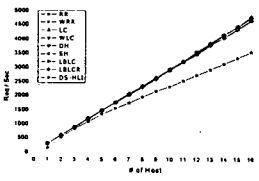
4.3 실험 결과

(1) 사용자가 동일 컨텐츠를 요청한 경우

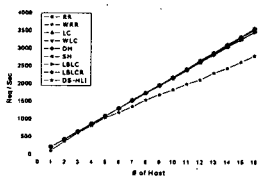
그림 6은 스케줄링별 호스트 개수에 따른 초당 요청수를 나타낸다. 동일 컨텐츠의 요청은 같은 크기의 이미지를 요청하는 방식으로 실험하였다.

•OS-HLI : 가중치를 업데이트 할 때는 요청을 분산할 수 없음으로 다른 스케줄링 알고리즘에 비해 성능 감소가 나타난다.

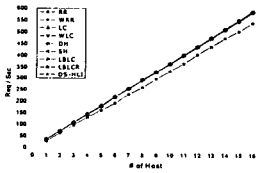
•RR, WRR, LC, WLC, LBLC, LBLCR, DH, SH : 기존 알고리즘은 호스트가 요청을 처리할 수 있는 최대치를 처리함을 알 수 있다.



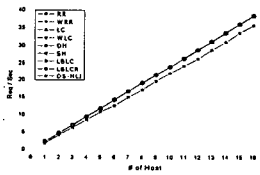
(a) 300 Bytes(



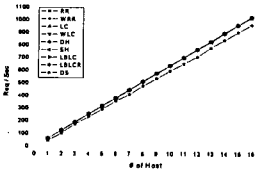
b) 1 Kbytes



(c) 10 Kbytes



(d) 100 Kbytes



(e) Variation

그림 6 호스트 개수에 따른 초당 요청 수 (동일 컨텐츠)

(2) 사용자가 다른 컨텐츠를 요청한 경우

1번 호스트에는 300 bytes를, 2-5번 호스트에는 1K, 10K, 100K, Variation bytes의 이미지를 넣었다. 나머지 호스트들도 같은 작업을 반복하였다. 그림 7은 호스트 개수에 따른 초당 요청수를 나타낸다.

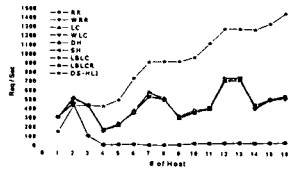


그림 7 호스트 개수에 따른 초당 요청 수 (다른 컨텐츠)

•RR, WRR, DH, SH : 최종 초당 요청수는 100 Kbytes를 처리하는 서버에 종속됨을 알 수 있다. 100 Kbytes를 처리하는 서버가 사용자 요청에 응답해야 모든 요청이 끝나기 때문이다.

•LC, WLC, LBLC, LBLCR : 100 Kbytes를 처리하는 서버로 최종 초당 요청수가 종속되지만, 상대적으로 다른 서버를 선택할 기회가 많음으로 RR, WRR, DH, SH 방법보다 좋은 성능을 보인다.

•OS-HLI : 사용자의 서로 다른 컨텐츠 요청에 따른 호스트 부하 정보를 이용하여 스케줄링을 함으로 100 Kbytes를 처리하는 서버에 최종 초당 요청수가 종속되지 않고 모든 방식 중에서 가장 좋은 성능을 가짐을 알 수 있다.

4.4 토론

제안된 방식의 단점은 사용자가 동일 컨텐츠를 요청할 때 기존 방식보다 성능이 떨어진다는 점이다. 이는 제안된 방식이 각 호스트의 부하 정보를 받아 가중치를 업데이트 할 때 요청을 처리하지 못하기 때문이다. 그러나 사용자의 실제 컨텐츠 요청 패턴은 서로 다른 컨텐츠를 요청하는 경우가 많음으로 이러한 성능 감소는 크게 문제가 되지 않는다고 가정할 수 있다.

제안된 방식의 장점은 사용자가 서로 다른 컨텐츠를 요청할 때 기존 방식에 비해 성능이 향상된다는 점이다. 이는 제안된 방식이 스케줄링을 할 때 호스트의 부하 정보를 이용한 것으로 설명할 수 있다. 제안된 방식은 서버의 CPU 부하 정보를 이용하여 스케줄링 함으로써 서버의 처리 능력 혹은 사용자의 요청 컨텐츠가 변하더라도 이를 능동적으로(사전 실험 없이) 반영하도록 하였다.

5. 결 론

기존 방법이 서버의 처리 능력 혹은 사용자의 요청 컨텐츠에 무관한 방법인 반면 제안된 방법은 서버의 부하 정보를 이용하여 서버의 처리 능력 및 사용자의 요청 컨텐츠의 정보를 반영하도록 하였다. 또한 실험을 통해 제안된 방법이 기존 스케줄링 방법에 비해 높은 성능 향상을 가짐을 확인하였다.

향후 연구 방향을 요약하면 다음과 같다.

•다양한 호스트 부하 정보의 적용 : 메모리, 네트웍 I/O, 시스템 인터럽트 등을 스케줄링 시에 고려한다.

•사용자의 다양한 컨텐츠 요청 반영 : 정적 컨텐츠(이미지, HTML 등)와 동적 컨텐츠(스크립트, CGI 등)를 요청하는 경우를 고려한다.

참고문헌

[1] LVS(Linux Vitual Server), http://www.linuxvirtualserver.org.
[2] 곽후근, 정규식, "무선 인터넷 프록시 서버 클러스터 성능 개선", 한국정보과학회:정보통신, 게재 예정, 2005.
[3] 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.
[4] A. Savant, N. Memon and T. Suel, "On the Scalability of an Image Transcoding Proxy Server", In IEEE International Conference on Image Processing, Barcelona, Spain, 2003.
[5] A. Fox, "A Framework for Separating Server Scalability and Availability from Internet Application Functionality", Ph. D. Dissertation, U. C. Berkeley, 1998.
[6] LVS Scheduling Algorithms, http://www.linuxvirtualserver.org/docs/scheduling.html.
[7] S. Hwang and N. Jung, "Dynamic Scheduling of Web Server Cluster", Proceedings of the 9th International Conference on Parallel and Distributed Systems, IEEE, 2002.
[8] C. A. Ruggiero and J. Sargeant, "Control of Parallelism in the Manchester Dataflow Machine", In Functional Programming Language and Computer Architecture, LNCS 274, Springer-Verlag, pp. 1-15, 1987.
[9] AB(Apache Bench), http://www.apache.org.
[10] Virtual Server via NAT, http://www.linuxvirtualserver.org/VS-NAT.html.
[11] Squid Web Proxy Cache, http://www.squid-cache.org.
[12] 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.