무선인터넷 프록시 서버 클러스터 환경에서 프록시 캐싱을 위한 향상된 부하 분산에 대한 연구

황재훈° 곽후근 정규식
숭실대학교 정보통신전자공학부
{dublur°, gobarian}@q.ssu.ac.kr, kchung@ssu.ac.kr

An Improved Load Balancing for Proxy Caching in a Wireless Internet Proxy Server Cluster Environment

Jaehoon Hwang° Hukeun Kwak, Kyusik Chung
School of Electronics Engineering, Soongsil University

요 약

현대 사회에서 무선 인터넷의 활용 분야가 날로 확대되어 가고 있으나 낮은 대역폭, 단말기 해상도의 다양성 등의 여러 가지 근본적인 문제들을 가지고 있다. 이러한 문제를 해결하기 위하여 무선인터넷 프록시 서버를 사용하며, 데이터에 대한 압축과 캐싱 방법을 사용한다. 캐싱 관점에서 해시의 사용은 캐시의 히트율(Hit ratio)을 높이며, 캐시간 협동성(Cooperative Caching)도 가지게 한다. 그러나 기존의 해시기반 스케줄링 알고리즘에서 사용자의 요청이 균등하게 분배되지 못하고 일부 서버로 몰리는 문제점을 가지고 있다.

본 논문에서는 해시 테이블을 이용한 라운드로빈 방식과 캐시 이용률을 이용한 스케줄링 방식을 제안하여, 캐시간 협동성과 사용자의 요청을 고르게 분포 시키도록 하였다. 16대의 컴퓨터를 이용하여 실험을 수행하였고, 실험결과를 통해 제안된 방식이 기존 방식보다 클라이언트의 요청을 캐시 서버들 사이로 균일하게 분포시키고, 이에 따라 전체 무선 인터넷 프록시 서버의 성능이 향상됨을 확인하였다.

1. 서 론

무선 인터넷에 대한 관심이 증가하는 가운데 무선 인터넷 단말기의 사용이 보편화 되고, 서비스도 검색 위주의 간단한 것에서 멀티미디어 서비스 등의 복잡한 서비스로 바뀌어져 가고 있다. 그러나 무선 인터넷에는 근본적인 문제점들을 가지고 있어서 이러한 문제점을 해결하기 위하여 무선 인터넷 프록시 서버를 사용하며 무선 사용자를 유선의 인터넷 서버에 연결 시켜주는 역할을 한다. 그림 1은 무선 인터넷에 사용되는 무선 인터넷 프록시 서버[5]를 나타내고 있다.

무선 인터넷 프록시 서버는 제한적인 대역폭을 효율적으로 사용하기 위해 캐싱(Caching)을 하며, 캐시에 저장된 데이터의 히트율(Hit Ratio)을 높이기 위해 해싱을 사용한다. 해싱은 그 특성상 캐시간의 협동성(Cooperative Caching)[1~4]을 보장한다. 이는 사용자가 요청한 URL의 해싱값을 이용하여 요청을 캐시 서버에 할당하는 것으로 클러스터간 중복되는 공간을 줄이고, 캐시서버 수에 무관하게 캐시 전체 크기를 일정하게 할 수 있는 것을 의미한다.

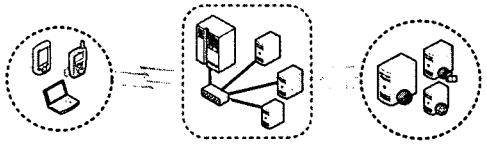


그림 1 무선 인터넷 프록시 서버

본 논문에서는 기존의 해싱을 이용한 스케줄링 방식이 가지는 문제점을 분석하고 이를 해결할 새로운 해시 스케줄링 방식을 제안한다. 본 논문은 2장에서는 기존의 해싱을 이용한 스케줄링 방식과 그 문제점을 소개한다. 3장에서는 기존의 문제점을 해결하는 새로운 해시 스케줄링 방식을 제안하고, 4장에서는 실험을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 기존 연구

캐시 선택 시에 사용할 수 있는 해싱 방법 중에는 정적 해싱(Static Hashing)[6]과 MD5 해싱(Message Digest ver.5 Hashing)[7] 방법 등이 있다.

2.1 정적 해싱(Static Hashing)[6]

정적 해싱은 임의개의 버킷(Bucket)을 생성하고 클러스터의 개수로 이를 분할하여 매핑하고 사용자 요청 URL을 기준으로 캐시를 선택하는 방식이다. 정적 해싱은 초기 버킷생성 후에는 일정 이상의 요청부터 클러스터 분배를 위한 추가적인 계산 시간이 소용되지 않는 장점을 가지나, 한번 정해진 버킷은 유동적이지 않기 때문에 캐시의 추가나 삭제시에 유동적이지 못하다는 단점을 가진다. 그림 2는 LVS(Linux Virtual Server)[6]에서 사용하는 정적 해싱을 통한 캐시 선택 방법을 나타낸다.

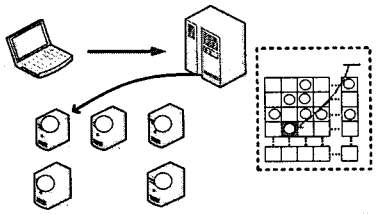


그림 2 LVS에서 사용하는 정적 해싱을 통한 캐시 선택

2.2 MD5 해싱(MD5 Hashing) [7]

MD5 해싱은 사용자 요청 URL에 대해 고정 길이의 해시값이 나오고, 이를 캐시의 수에 매핑하는 방식으로 동작한다. 요청이 들어올 때 마다 Message Digest를 생성, 클러스터에 할당하기 때문에 버킷을 위한 별도의 메모리 공간 소요가 없고, 캐시의 추가 및 삭제 시에 영향을 받지 않는 유동적인 특징을 가진다. 그러나, 매 요청마다 해시값을 계산하여 클러스터로 분배하기 때문에 사용자의 요청 수 증가에 따른 각 요청의 해싱값 계산 시간이 커진다는 단점을 가진다. 그림 3은 무선 인터넷 프록시 서버인 TranSend에서 사용하는 MD5 해싱을 통한 캐시 클러스터의 선택을 나타내고 있다.

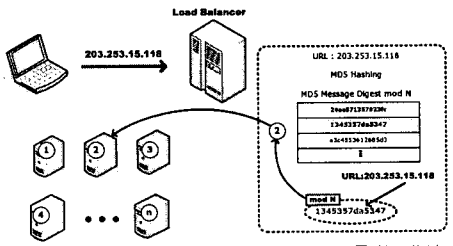


그림 3 Transend에서 사용하는 MD5 해싱을 통한 캐시 선택

2.3 접근 방식

정적 해싱은 고정적인 해싱을 하기 때문에 캐시의 추가나 삭제 등의 유동적인 상황을 반영하지 못하며, MD5해싱은 해시의 특성상 전체 사용자의 요청이 일부 서버로 집중 될 수 있는 단점을 가진다. 3장에서는 사용자의 요청을 정적 해싱처럼 균등하게 분포시키며, MD5처럼 유동적인 상황을 반영하는 새로운 스케줄링 알고리즘을 제안한다.

3. 제안된 알고리즘

3.1 해시 테이블을 이용한 라운드 로빈

해시 테이블을 이용한 라운드 로빈 방식은 캐시가 협동성을 가지도록 하기 위해 해시를 이용하되 이들의 특성으로 인해 사용자 전체의 요청이 일부 캐시 서버로 집중되는 것을 막는 것이다. 기존 방식이 사용자 URL의 해시값을 사용된 캐시 서버 수로 나누어 사용자 요청을 할당했다면, 이 방식은 사용자 URL의 해시값을 해시 테이블로 관리하여 사용자의 요청을 캐시 서버로 균일하게 할당하였다.

그림 4는 MD5 및 해시 테이블을 적용하여 캐시를 선택하는 과정을 나타내고 있으며 이를 정리하면 다음과 같다.

- 부하 분산기가 사용자의 요청 URL에 대해 MD5 해싱을 적용하여 Message Digest(해시값)를 생성한다.
- 해시 테이블을 검색하여 Message Digest(해시값)에 해당하는 곳에 캐시 서버가 할당되어 있는지 확인하고, 할당되어 있다면 해당 캐시 서버로 사용자 요청을 보낸다.
- 캐시 서버가 할당되어 있지 않다면, 새로운 캐시 서버를 할당한다. 캐시 서버가 N개라면 처음 캐시 서버 할당을 요청할 때는 캐시 서버 1을 할당한다.
- 두 번째 요청에는 캐시 서버 2를 할당하며, 라운드 로빈 방식으로 동작하여, N+1번째 캐시서버 할당을 요청할 때는 캐시 서버 1을 다시 할당한다.

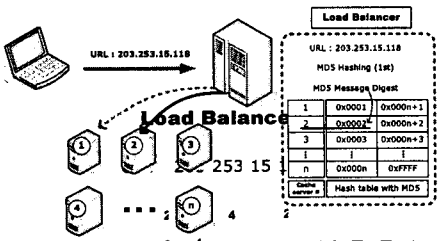


그림 4 해시 테이블을 이용한 라운드 로빈

3.2 캐시 사용률을 이용한 스케줄링

캐시 사용률을 이용한 스케줄링 방식은 해시값으로 캐시 서버를 할당할 때 캐시 사용률을 이용하고 이들을 해시 테이블로 관리하여 사용자의 요청에 대하여 사용률이 낮은 캐시 서버로 먼저 할당하여 요청이 균일하게 할당되도록 하였다.

그림 5는 MD5 및 캐시 사용률을 적용하여 캐시를 선택하는 과정을 나타내며 이를 정리하면 다음과 같다.

- 부하 분산기가 사용자의 요청 URL에 대해 MD5 해싱을 적용하고 계산된 해시값에 캐시서버가 할당되어 있는지 확인하고 할당되어 있다면, 해당 캐시 서버로 요청을 보낸다.
- 캐시 서버가 할당되어 있지 않다면 새로운 캐시 서버를 할당하고, 사용자의 요청을 새로운 캐시 서버로 보낸다.
- 캐시 서버가 할당되어 있지 않다면, 각 캐시 서버들의 현재 사용 횟수를 비교하고, 가장 적게 사용되는 캐시에 요청을 할당하고, 할당된 캐시의 사용횟수를 증가 시킨다.

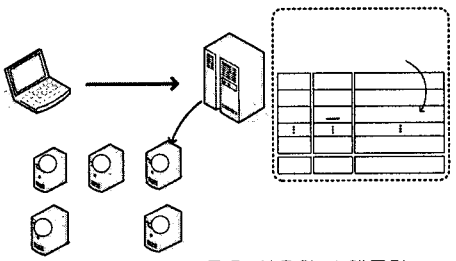


그림 5 캐시 사용률을 이용한 스케줄링

4. 실험 및 토론

4.1 실험환경

표 1은 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.

표 1 실험용 하드웨어 & 소프트웨어

		하드웨어		소프트웨어	개수
		CPU (Hz)	RAM (MB)		
Client		P-IV 2.26 G	256	AB[6]	1
LVS		P-IV 2.4 G	512	NAT[6]	1
Host	Cache	P-II 400 M	256	Squid[10]	16
	Distiller			JPEG-6b[13]	

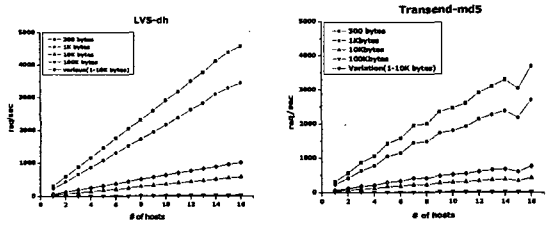
표 2는 실험에 사용된 변수들을 정리한 것이다.

표 2 실험에 사용된 변수

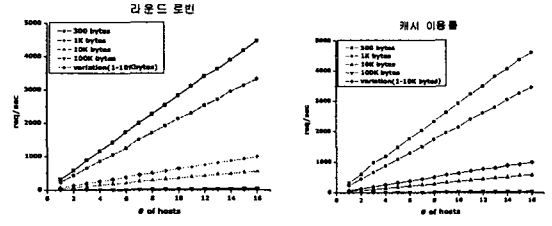
사용자의 요청 개수	◦ 약 200초 동안 프록시가 처리할 수 있는 최대 개수
요청 이미지	◦ JPEG[11]
요청 크기	◦ 300 bytes, 1 K, 10 K, 100 Kbytes, Variation[13]
사용자 정보 (Preference)	◦ 이미지 Quality = 중간
웹 서버	◦ Cache 서버 자체에 둠 (프록시내의 성능 평가에 초점을 맞춤)
병 목 (bottleneck)	◦ 호스트의 CPU 점유율 중 가장 높은 호스트 (본 실험에서는 ethernet이나 system bus 병목 은 발생하지 않는다.)

4.2 실험 결과

그림 6은 각 알고리즘 별 이미지 크기를 달리 요청했을 때 호스트 개수에 따른 초당 요청수를 나타낸 것이다.



(a)LVS-DH (b)Transend_md5



(c)해시테이블을 이용한 라운드로빈 (d)캐시사용률 이용한 스케줄링

그림 6 호스트 개수에 따른 초당 요청 수

그림 6(a)는 정적해싱인 lvs-dh 알고리즘을 사용한 것이다. 그리고 그림 6(b)는 MD5해싱 스케줄링 방식을 사용한 것으로 호스트 개수에 따라 성능이 비례적으로 증가하지 않는 것을 볼수 있는데, 이는 MD5해싱 특성에 따라 요청이 일부 서버로 몰려서 전체 성능이 이에 종속된 것을 이유 때문이다.

표 3은 이미지 크기별 초당 요청수를 비교한 것이다. 일반적인 LVS-RR을 기준으로 하여 4개의 해시 알고리즘을 비교하였다. 표의 결과와 같이 평균적으로 md5가 최저의 성능을 나타낸 것을 볼 수 있으며, 제안된 2가지 해시 스케줄링 방식이 일반적 라운드 로빈의 값에 근접함을 볼 수 있다. 결과에서 정적해싱인 LVS-dh가 좋은 성능을 가지는 것을 볼 수 있는데, 이는 다양한 IP의 실험을 하지 않은 것과, 유동적인 상황을 실험에 반영하지 않았기 때문에 고정적인 상황에서 얻어진 좋은 결과라고 설명할 수 있다.

표 3 이미지 크기별 초당 요청 수 비교

Algorithm	300bt	1K	10K	100K	Variation	Average
LVS-RR	100	100	100	100	100	100
LVS-DH	100.54	100.72	100.54	100.93	98.91	100.33
Transend -md5	88.42	87.03	84.92	95.05	83.25	87.73
라운드로빈	97.97	98.16	99.22	99.67	98.03	98.61
캐시이용률	101.63	100.39	100.41	107.71	97.95	99.95

5. 결 론

제안된 해시테이블을 라운드 로빈 방식과 캐시 사용률을 이용한 스케줄링 방법이 기존 해싱을 이용한 스케줄링 방법에 비해 성능이 좋아진 이유는 사용자의 요청을 일부 서버로 몰리지 않게 한 것으로 설명할 수 있다. 기존 방식은 사용자의 목적지 URL을 이용하여 해싱을 적용하여 부하가 일부 서버로 몰리는 반면, 제안된 방식은 사용자의 목적지 URL을 이용하여 해싱을 적용하고 이를 해시 테이블로 관리함으로써 전체 사용자의 요청을 캐시 서버 사이로 균일하게 분산하였다.

향후 연구 방향을 요약하면 다음과 같다.

- 기존의 실험에서는 일정한 크기의 단순한 이미지와 일정 범위의 IP주소, 일정 수의 요청 및 요청 시간으로 실험을 하였다. 이는 실제 인터넷 환경의 다양한 요청들을 반영하지 못하는 단점을 가지고 있었다. 제안된 알고리즘들을 사용자의 다양한 패턴을 반영한 요청을 처리하는 것을 실험하여 기존 알고리즘들과 비교하여 본다.
- 캐시 서버의 추가나 삭제의 경우에도 크게 영향을 받지 않고 해쉬 테이블을 생성하여 이에 따라 스케줄링 할 수 있는 알고리즘으로 보완한다.

6. 참고 문헌

[1] F. Baboescu, "Proxy Caching with Hash Functions", Technical Report CS2001-0674, 2001.
[2] Micorsoft Corp., "Cache Array routing protocol and microsoft proxy server 2.0", White Paper, 1999.
[3] David Karger and al., "Web caching with consistent hashing", The 8th Internation WWW Conference, 1999.
[4] K. Ross, "Hash-routing for collections of shared web caches", IEEE Network Magazine, 1997.
[5] 곽후근, 정규식, "무선 인터넷 프록시 서버 클러스터 성능 개선", 한국정보과학회:정보통신, 2005.6
[6] LVS(Linux Vitual Server), http://www.linuxvirtualserver.org
[7] D. Rivest, "The MD5 Message Digest Algorithm", RFC 1321, 1992.
[8] AB(Apache Bench), http://www.apache.org.
[9] Virtual Server via NAT, http://www.linuxvirtualserver.org/VS-NAT.html.
[10] Squid Web Proxy Cache, http://www.squid-cache.org.
[11] 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.
[12] 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.
[13] S. Chandra, A. Gehani, C. Ellis and A. Vahdat, "Transcoding Characteristics of Web Images", Proceedings of the SPIE Multimedia Computing and Networking Conference, 2001.