확장성 있는 캐시 서버 클러스터에서의 부하 분산을 위한 동적 서버 정보 기반의 해싱 기법

곽 후 근[†] · 정 규 식[††]

요 약

캐시 서버 클러스터에서의 캐싱은 인터넷 트래픽 및 웹 유저의 요청 및 응답 시간을 줄여주는 효과를 가진다. 이때, 캐시의 히트율(Hit ratio)을 증가시키는 한 가지 방법은 해시 함수를 이용하여 캐시가 협동성(Cooperative Caching)을 가지도록 하는 것이다. 캐시가 협동성을 가진다는 것은 캐시 서버 수와 무관하게 캐시 메모리 전체 크기를 일정하게 할 수 있다는 것을 의미한다. 반면에 캐시가 협동성을 가지지 않는다면 각 캐시 서버가 모든 캐시 데이터를 가져야 하므로 캐시 메모리 전체 크기가 캐시 서버 수에 비례하여 증가한다.

해싱을 이용한 방법의 문제점은 해시의 특성으로 인해 클라이언트의 요청이 일부 캐시 서버로 집중되고 전체 캐시 서버 클러스터의 성능이 일부 캐시 서버에 종속된다는 점이다. 이에 본 논문에서는 동적 서버 정보를 이용하여 클라이언트의 요청을 일부 캐시 서버가 아닌 전체 캐시 서버에 균일하게 분포시키는 방법을 제안한다. 16대의 컴퓨터를 이용하여 실험을 수행하였고 실험 결과는 기존 방법에 비해 클라이언트의 요청을 캐시 서버들 사이로 균일하게 분포시키고 이에 따라 전체 캐시 서버 클러스터의 성능이 향상됨을 확인하였다.

키워드 : 캐시 서버, 클러스터, 스케줄링, 해싱, 동적 서버 정보

Hashing Method with Dynamic Server Information for Load Balancing on a Scalable Cluster of Cache Servers

Hukeun Kwak[†] · Kyusik Chung[††]

ABSTRACT

Caching in a cache server cluster environment has an advantage that minimizes the request and response time of internet traffic and web user. Then, one of the methods that increases the hit ratio of cache is using the hash function with cooperative caching. It is keeping a fixed size of the total cache memory regardless of the number of cache servers. On the contrary, if there is no cooperative caching, the total size of cache memory increases proportional to the number of cache servers since each cache server should keep all the cache data.

The disadvantage of hashing method is that clients' requests stress a few servers in all the cache servers due to the characteristics of hashing and the overall performance of a cache server cluster depends on a few servers. In this paper, we propose the method that distributes uniformly client requests between cache servers using dynamic server information. We performed experiments using 16 PCs. Experimental results show the uniform distribution o

Keyword : Cache server, Cluster, Scheduling, Hashing, Dynamic Server Information

1. 서 론

웹 사이트를 운영할 때 캐싱을 사용하지 않는다면, 사용자가 매번 웹 사이트를 방문해야 함으로 요청에 대한 응답 시간이 길어지고, 웹 사이트에 요청이 집중되거나(Hot-Spot) 다운되면 요청에 대한 응답을 할 수 없다[1, 2]. 웹 사이트를 운영할 때 캐싱을 사용한다면, 사용자는 요청에 대한 응답을 캐시 서버로부터 받을 수 있음으로 요청에 대한 응답 시간을 단축할 수 있고, 웹 사이트에 요청이 집중되거나 다운되어도 일시적으로는 요청에 대한 응답을 받을 수 있다. 캐싱을 운영하는 방법[3]을 위치에 따라 분류하면, 캐시 서버의 위치에 따라 전방, 중간, 후방 캐싱이 존재한다. 전방 캐시 서버는 클라이언트 쪽에 위치하여 캐싱을 수행하고, 후방 캐시 서버는 웹 서버 쪽에 위치하여 캐싱을 수행한다. 중간 캐시 서버는 클라이언트와 웹 서버의 중간에 위치하여 클라이언트가 캐시 서버의 존재 여부를 모르게 캐싱을 수행한다.

캐시간 협동성(Cache Cooperation 또는 Cooperative

※본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음
† 준 회 원 : 숭실대학교 IT 대학 정보통신전자공학부 박사
†† 정 회 원 : 숭실대학교 IT 대학 정보통신전자공학부 교수
논문접수 : 2007년 3월 10일, 심사완료 : 2007년 7월 18일

Caching)[4-7]을 사용하지 않는다면, 웹 사이트를 구성하는 서버들이 모든 데이터를 중복해서 저장해야 한다. 데이터의 크기가 늘어나면 전체 데이터의 합은 (서버의 수 x 데이터의 크기)가 된다. 캐시간 협동성을 사용한다면, 웹 사이트를 구성하는 서버들이 모든 데이터를 나누어서 저장한다. 데이터의 크기가 늘어나도 전체 데이터의 합은 (1 x 데이터의 크기)로 일정하게 유지된다. 캐시간 협동성을 위한 방법에는 프로토콜을 이용하는 방법과 해싱을 사용하는 2가지 방법이 존재한다. 프로토콜을 이용하는 방법[8-11]은 요청을 받은 캐시 서버가 데이터를 가지고 있지 않다면, 주변의 캐시 서버에게 해당 데이터를 가지고 있는지 물어본다. 만약, 주변 캐시 서버가 클라이언트의 요청 데이터를 가지고 있다면 그 데이터를 사용하고, 가지고 있지 않다면 웹 서버에게 데이터를 요청한다. 해싱을 이용하는 방법[12-18]은 사용자가 요청한 데이터(URL)를 해싱한 후 그 해시 값을 이용하여 캐시 서버를 선택하는 방법이다. 해시을 이용하게 되면 동일한 URL에 동일한 해시 값이 생성됨으로, 동일 URL에 대해 동일 캐시 서버를 선택할 수 있다. 이러한 해싱 및 캐시 서버의 선택은 캐시 서버로 요청을 분산하는 부하 분산기가 수행하게 된다.

기존 방법들의 단점은 요청을 캐시 서버에 균일하게 분포시키는 문제, 확장성 문제, 요청 집중 처리 문제로 나누어서 생각할 수 있다. 균일 분포의 문제에서는 클라이언트가 요청한 URL과 캐시 서버를 연결할 때 해시를 사용하게 되면 클라이언트가 요청한 URL이 캐시 서버에 균일하게 할당되지 않는다. 이는 해싱을 통해 해시 값을 얻을 때 이 값이 균일하게 분포하지 않음을 의미한다. 확장성(Scalability)의 문제에서는 클라이언트가 요청한 URL이 캐시 서버로 고르게 분포되지 않으면, 특정 캐시 서버들로 요청이 집중된다. 특정 캐시 서버들로만 요청이 집중되면 캐시 서버의 전체적인 성능은 특정캐시 서버에 의존하게 됨으로 전체적인 확장성은 떨어지게 된다. 요청 집중 처리의 문제에서는 클라이언트가 요청한 동일한 URL은 동일한 캐시 서버에게 할당이 된다. 이때, 특정 URL로 요청이 집중되는 경우 캐시 서버의 전체적인 성능은 특정 URL을 처리하는 캐시 서버에 의존하게 된다.

본 논문에서 제안한 방법은 동적 서버 정보에 기반하여 해싱을 수행하는 방법이다. 균일 분포와 확장성을 위해 해시 값으로 캐시 서버를 선택할 때 동적 서버 정보를 이용하고, 요청 집중 처리를 위해 라운드 로빈(Round-Robin)을 이용하였다. 본 논문의 구조는 다음과 같다. 2장에서는 캐시 서버 클러스터를 위한 기존 해싱 방법이 가지는 문제점을 소개한다. 3장에서는 기존 해싱 방법들이 가지는 문제점을 해결하는 새로운 동적 서버 정보 기반의 해싱 방법을 설명하고, 4장에서는 실험 결과를, 5장에서는 성능을 분석한다. 마지막으로 6장에서는 결론 및 향후 연구 방향을 제시한다.

2. 연구 배경

2.1 MD5

MD5(Message Digest ver.5)[12]는 1992년에 MIT의 Ronald Rivest가 개발하였다. 입력되는 메시지는 임의의 길이를 가질 수 있고, 이는 264 bit 이하여야 한다. 메시지는 512 비트 단위 블록들로 분할되며, 이것은 각각 16개의 32 비트 서브 블록으로 분할된다. 이 메시지 블록들은 512 비트의 배수 길이가 되도록 0으로 패딩되고, 마지막 블록에서 64 비트 구간은 길이 필드로 사용된다. 메시지 축약의 결과는 128 비트(=32 비트 X 4 서브 블록)이다.

MD5 해싱은 사용자 요청 URL에 대해 고정 길이의 해시 값이 나오고, 이를 캐시의 수에 매핑하는 방식으로 동작한다. 요청이 들어올 때 마다 메시지 축약을 생성, 클러스터에 할당하기 때문에 버킷을 위한 별도의 메모리 공간 소요가 없고, 캐시의 추가 및 삭제 시에 영향을 받지 않는 유동적인 구조를 가진다. 그러나 매 요청마다 해시 값을 계산하여 클러스터로 분해하기 때문에 사용자의 요청 수 증가에 따른 각 요청의 메시지 축약 계산 시간이 커진다는 단점을 가진다.

2.2 MIT-CH (Consistent Hashing)

MIT-CH[13]는 1997년 MIT에서 개발한 대표적인 해시 기반 부하 분산 방식으로서 현재까지 이를 응용한 여러 해싱 방식들이 제안되고 있다. MIT-CH는 MD5 등의 랜덤 함수를 사용하여 요청된 URL에 대해서 해시 값을 계산한다. 계산된 해시 값은 (0,1]의 영역을 가지는 단위원에 할당하며, 캐시 서버에 대해서도 같은 방식으로 할당한다. 요청URL에 대해 할당된 값은 원형의 구간에서 시계방향으로 가장 가까운 캐시 서버에 할당되는 방식이다. 또한 별도의 부하 분산기의 사용 없이 DNS(Domain Name System)에서 요청 URL에 대한 해싱 기능을 제공하여 사용자가 캐시 서버로 접속하도록 하였다.

MIT-CH의 캐시 포인트는 이진 트리 형식으로 저장될 수 있고, 캐싱된 URL에 대한 검색은 이진트리 내에서 단일 검색 방식이 이용된다. N개의 캐시 서버에서 검색을 위한 시간이 $O(\log n)$ 내에서 소요되기 때문에 캐시 서버가 증가하여도 캐싱된 URL에 대한 검색 시간이 별로 걸리지 않는 장점을 가진다. 그러나 요청 URL과 캐시 서버 이름에 대해서 MD5를 통하여 할당하기 때문에 해시 특성상 요청이 균일하게 분포되지 못하는 단점을 가지고 있고, 요청이 균일하게 분포되지 못하므로 서버 확장에 따른 성능의 선형적 증가를 보장하지 못한다. 또한 캐시 서버의 부하 상황을 부하 분산에 반영하지 못하기 때문에 순간적인 요청 집중 현상에 대처할 수 없는 단점을 가진다.

2.3 CARP (Cache Array Routing Protocol)

CARP[14]는 1997년 Microsoft 사에서 개발한 해시 기반 부하 분산 방식으로서 현재 Microsoft 사의 제품에서 사용되고 있는 부하 분산 방식이다. CARP는 요청 URL과 캐시 서버의 이름에 대하여 각각의 해시 값을 계산하고, 서로 조합하여 최고의 조합 값을 가지는 캐시 서버로 할당하는 방식을 사용한다. 구조로는 전체적으로 평등한 병렬 구조를 가지는 것과 다운 스트림 프록시 및 업 스트림 프록시의 계

층적인 구조를 가지는 것이 있다.

CARP는 일반적인 해시 기반 부하 분산 방식과 같이 ICP(Inter Cache Protocol) 요청 메시지 없이 해시에 의하여 데이터를 캐싱하고 있는 서버로 요청을 할당하며, 데이터의 중복 저장을 방지하는 효과를 가진다. 또한 캐시 서버 이름과 요청 URL의 해시 값에 대한 할당이 비교적 균일하게 분포되고 캐시 서버를 추가 할 때 전체적인 클러스터의 성능이 선형적으로 증가하므로 서버의 추가에 따른 확장성을 보장할 수 있다. 그러나 사용자의 요청에 대해서 해시 값을 계산하고 캐시 서버들의 해시 값과 조합하여 캐시 서버를 선택하는 방법을 사용하기 때문에 요청 처리에 대한 절차가 다소 복잡하고, 캐시 서버의 부하 상황을 반영하지 못하기 때문에 순간적인 요청 집중 현상에 대해서 대처할 수 없는 단점을 가진다.

2.4 MOD-CH (Modified Consistent Hashing)

MOD-CH[15]는 기존의 MIT-CH 방법의 단점인 클러스터 내에서 요청을 균일하게 할당하지 못하는 것을 보완하고자 제안된 방식이다. 이는 기존의 MIT-CH에서 요청 URL에 대한 캐시 서버의 할당과 관련하여 해시 포인트가 단위원에서 시계방향으로 가까운 서버로 할당되는 방식을 수정하여 해시 포인트가 방향과 상관없이 가까운 캐시 서버에 할당되게 하여 캐시 서버간의 요청 할당에 대한 표준 편차를 줄이는 역할을 하도록 제안되었다.

MOD-CH는 기존 MIT-CH의 특징을 그대로 가지면서 각 서버의 요청 할당에 대한 표준 편차를 줄여 전체적으로 요청이 균일하게 분포되도록 하였다. 그러나 캐시 서버의 할당과 관련하여 여전히 랜덤 함수를 사용함으로 완전한 균일 분포를 이루지 못하는 단점을 가진다. 이로 인해 서버의 확장에 따른 성능의 선형적인 증가를 완벽하게 보장하지 못하며, 캐시 서버의 부하 상황을 반영하지 못하기 때문에 순간적인 요청 집중 현상에 대해서 대처할 수 없는 단점을 가진다.

2.5 Adaptive-LB (Adaptive Load Balancing)

Adaptive-LB[16]는 2004년에 제안된 방식으로서 기존의 방식과 달리 해시 기반 부하 분산과 함께 라운드 로빈 부하 분산 방식을 동시에 사용하여 부하 분산한다. 이 방식은 기본적으로 해시 기반 부하 분산 방식을 사용하지만, 요청 집중 상황의 처리에 초점을 둔다. 즉, 가장 자주 요청되는 데이터에 대해서 목록으로 관리하고, 해당요청이 발생하면 전체 캐시 서버에 라운드 로빈으로 분산시키는 방법을 사용한다.

Adaptive-LB는 다른 해시 기반 부하 분산 방식과 달리 요청 집중 현상에 대해서 적응적으로 처리할 수 있다는 장점을 가지지만, 그 이외의 사항인 캐시 서버의 추가/삭제 문제, 요청의 균일 분포 및 서버의 확장성에 대해서는 고려를 하지 않고 있다.

2.6 EXT-CH (Extended Consistent Hashing)

EXT-CH[17]는 기존의 MIT-CH 방식의 변형으로 요청 컨텐츠를 캐시 서버에 할당할 때 컨텐츠의 부하 정보를 반영하였다. 이러한 방식을 이용하여 요청에 대해 캐시 서버를 할당하게 되면 요청 집중 현상을 처리할 수 있다.

EXT-CH는 요청 컨텐츠의 요청 빈도 및 이들의 평균 부하에 따라 캐시 서버 할당 영역을 설정하여 요청 집중 현상에 대해서 효과적으로 처리할 수 있다. 그리고 MIT-CH에 비해 요청의 균일한 분포를 나타낼 수 있는 장점을 가진다. 그러나 MIT-CH의 특성을 가지므로 전체적으로 균일한 요청의 분포를 나타내지 못하며, 이에 따라 캐시 서버의 확장에 따른 성능의 선형적 증가도 가지지 못하는 단점을 가지고 있다. 또한, 캐시 서버의 할당과 컨텐츠의 부하 정보를 DNS 서버에서 관리함으로 DNS 서버에 무리한 부하를 가중시키는 단점을 가진다.

2.7 CC-DH (Cache Clouds Dynamic Hashing)

CC-DH[18]는 기존의 일반적으로 작은 캐시 클러스터 그룹 환경에서 운영되는 해시 기반 부하 분산 방식과 달리 네트웍 끝단 부분에서 내부 네트워크와 캐시 서버의 전체적인 성능 향상을 위하여 캐시 서버 간의 협동성을 가지게 하는 것이다. 이 방식은 Cache Clouds라는 개념을 소개하고, Cache Clouds 내에서 캐시 서버의 부하 정보를 고려하여 캐시 서버를 할당하는 동적 해싱 방식을 제안하였다.

대규모 캐시 서버 그룹에 적용되는 CC-DH는 기존의 프록시 캐싱과 응용 환경이 다른 것을 볼 수 있다. 그리고 기본적인 개념에서도 기존의 해시 기반 부하 분산에서 사용하

〈표 1〉 해시 기반 부하 분산의 단점

해시 기반 부하 분산	단 점
MD5	단순히 해시 값만 사용하는 것으로서 요청에 대한 해시 값이 일정하지 않기 때문에 서버 추가에 대한 성능 확장성이 없고 요청 집중 현상을 처리할 수 있다.
MIT-CH	서버의 추가/삭제의 경우 변동의 폭을 줄이고, 캐싱된 데이터에 대한 검색 시간을 줄이는 장점을 가진다. 데이터 분포와 관련해서 균일 분포를 가지지 못하므로 확장성이 없으며 요청 집중 현상을 처리하지 못한다.
CARP	서버의 추가/삭제에 대한 변화가 적고, 비교적 균일한 분포를 가지지만 요청이 왔을 때 할당 서버를 찾는 연산회수가 많고, 요청 집중 현상에 대한 고려가 되어 있지 않다.
MOD-CH	기존 MIT-CH 보다 데이터의 균일한 할당은 개선하였으나 완벽한 성능에 대한 확장성을 지원하지 못하고, 요청 집중 현상에 대한 고려가 되어 있지 않다.
Adaptive-LB	요청이 몰리는 컨텐츠에 대해서 라운드 로빈을 사용하여 처리하나, 이외의 일반적인 컨텐츠에 대해서는 해시 기반의 부하 분산을 사용하므로 데이터의 균일 분포나 성능의 확장성이 부족하다.
EXT-CH	요청 윈도우를 사용하여 자주 요청 되는 데이터에 대해서는 여러 개의 캐시 서버를 할당하여 요청 집중을 처리한다. 그러나 기존 MIT-CH의 문제인 데이터가 균일하게 할당되지 못하는 문제를 가지므로 캐시 서버의 확장성을 보장하지 못한다.
CC-DH	서버의 부하 정보를 고려하여 부하 분산을 함으로써 요청 집중 현상을 해결하였다. 그러나 할당된 캐시 서버를 찾기 위해 2번 이상의 해시 연산이 필요하고, 적용 분야 또한 클러스터링 환경이 아닌 분산 캐시 서버 구조에서 사용되도록 제안되었다.

지 않았던 캐시 서버의 부하 정보를 반영한 동적 해싱 방법을 사용한다. 그러나 요청의 균일한 분포 문제와 확장성에 따른 성능 증가 및 요청 집중 문제에 대해서 고려되지 않은 것을 볼 수 있다.

2.8 비교

기존 해시 기반 부하 분산 방식의 단점을 정리하면 <표 1>과 같다.

3. 동적 서버 정보 기반의 해싱 기법 (HT-CU)

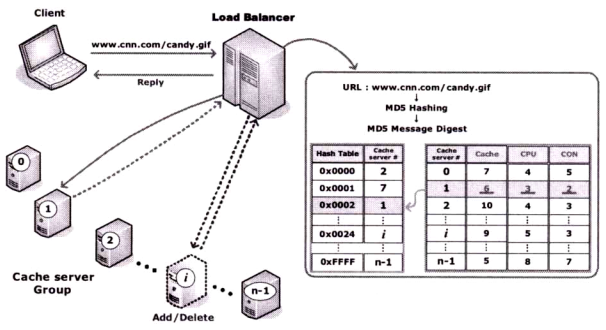
3.1 전체 구조

위에서 언급된 해싱 기법들은 해시 값들을 직접 캐시 서버들에게 정적인 정보를 이용하여 매핑하는 방법으로 동작한다. 본 논문에서 제안하는 방법은 하나의 특정 캐시 서버에 매핑하기 전에 정적인 정보 외에 동적인 캐시 서버의 정보를 고려하는 방법으로 동작한다. 여기서 동적인 캐시 서버의 정보는 캐시의 이용률, 캐시 서버 CPU 이용률 및 캐시 서버에 연결된 커넥션의 수로 구성된다. 이를 수식(l_cur)으로 정리하면 다음과 같다.

l_cur = W_{cache} × (캐시 이용률/평균 캐시 이용률) + W_{cpu} × (CPU 이용률/평균 CPU 이용률) + W_{con} × (커넥션 수/평균 커넥션 수)

여기서, W_{cache}는 캐시 이용률에 대한 가중치, W_{cpu}는 CPU 이용률에 대한 가중치 및 W_{con}은 커넥션 수에 대한 가중치이다. 각각의 정보들을 평균으로 나눈 이유는 각각이 값들이 가질 수 있는 범위가 다르기 때문에 이를 표준화하기 위해서이다.

캐시 서버에 요청을 할당하는 것은 동적인 캐시 서버 정보를 가지는 수식(l_cur)에서 가장 작은 값을 찾음으로써 이루어진다. 이러한 동적인 캐시 서버 정보는 모든 해시 값과 이들이 할당되는 서버들이 모든 캐시 서버들 사이에서 균일하게 분포시킨다. (그림 1)은 이러한 제안된 방법을 보여준다. 여기에 2개의 테이블이 존재한다. 첫 번째 테이블은 해시 값과 이들에게 할당된 캐시 서버로 구성되고, 두 번째 테이블은 동적인 캐시 서버 정보들로 구성된다. 이러한 두 번째 동적 서버 정보 테이블에는 각 캐시 서버의 동적인 정보(캐시의 이용률, 캐시 서버 CPU 이용률 및 캐시 서버에 연결된 커넥션의 수)들이 들어가 있다.



(그림 1) 동적 서버 정보 기반의 해싱 기법

3.2 동작 과정

하나의 요청(URL)을 받았다고 가정하자. MD5 해싱은 0x0000에서 0xFFFF 범위에서 하나의 해시 값을 생성할 것이다. 위의 예에서 해시 값은 0x0002로 가정한다. 해시 테이블 엔트리에 이미 캐시 서버가 할당되어 있다면, 요청은 해당 서버로 보내질 것이다. 그러나 엔트리가 비어 있고, 캐시 서버가 할당되지 않았다면, 캐시 서버는 동적 서버 정보 테이블을 검색하고 수식(l_cur)을 통해 최소 값을 찾음으로써 할당될 것이다. 위 그림에 따르면, 캐시 서버 1이 수식(l_cur)을 통해 가장 적은 값을 가지게 되고 현재의 요청에 대한 서버로써 선택되어진다.

모든 캐시 서버들이 수식(l_cur)을 통해 같은 값을 가지게 되면, 캐시 서버의 할당은 캐시 서버 0(서버 번호가 가장 작은 값)이 된다. 이러한 동적 캐시 서버 정보의 사용은 부하 분산기가 요청들을 모든 캐시 서버들 사이에서 균일하게 분포되도록 한다. 만약, MD5 해싱만을 이용하여 사용자의 요청을 캐시 서버에 할당하였다면 서버 2가 할당된다. 왜냐하면 MD5 해싱만을 이용하여 사용자의 요청을 캐시 서버에 할당하는 방법은 해쉬값을 서버의 개수로 나눈 후 나머지를 서버에 할당하는 방법이기 때문이다. 이러한 방법의 문제점은 캐시 서버의 상태를 전혀 고려하지 않았고, 해시의 기본적인 특성으로 인해 사용자의 요청의 일부 캐시 서버에 집중된다는 점이다.

만약 요청 집중이 발생하게 되면 요청 집중을 발생시킨 해당 URL을 하나의 캐시 서버(해당 URL을 처리했던)에서 처리하는 것이 아니라, 모든 캐시 서버들이 요청 집중 URL을 처리한다. 이때, 처리 방식은 라운드 로빈을 이용한다. 즉, 하나의 캐시 서버로 요청이 집중되는 URL이 발생하면 해당 URL을 라운드 로빈 부하 분산 방식을 이용해 모든 캐시 서버가 요청 집중 URL을 공평하게 분산해서 처리한다. 시간이 흘러 요청 집중 URL이 없어지면 해당 URL은 다시 이를 처리했던 원래의 캐시 서버가 다시 처리를 담당하게 된다.

3.3 기존 방법과의 비교

<표 2>는 기존 방법과 제안된 방법을 알고리즘 복잡도, 서버 추가/삭제, 확장성, 균일 분포 및 요청 집중 관점에서 비교한 표이다. 해쉬 기반 알고리즘의 효율성을 측정하는 기본적인 항목은 서버 추가/삭제에도 제대로 동작하는가, 사

〈표 2〉 기존 방법과 제안된 방법

방법	알고리즘 복잡도	서버 추가/삭제	확장성	균일 분포	요청 집중
MIT-CH	2 단계	O	X	X	X
CARP	3 단계	O	X	X	X
MOD-CH	3 단계	O	X	X	X
Adaptive-LB	3 단계	O	X	X	O
EXT-CH	3 단계	O	X	X	O
CC-DH	3 단계	O	X	X	O
제안된 방법	3 단계	O	O	O	O

용자의 요청이 서버들 사이로 균일하게 분포하는가, 요청이 캐시 서버 중 일부로 집중되면 이를 처리할 수 있는가 이다. 또한 해당 기법은 소프트웨어로 제작되었기 때문에 알고리즘의 복잡도를 통해 그 효율성이 입증되어야 한다. 마지막으로 여러대의 서버들을 사용하기 때문에 서버를 추가할 때마다 성능도 비례해서 증가하는 확장성을 가져야 한다.

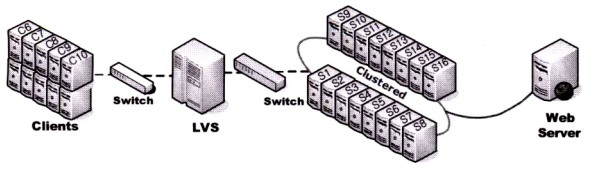
4. 실험 결과

4.1 실험 환경

제안된 방법을 테스트하기 위해, 본 논문에서는 (그림 2)에서 보이는 실험 환경을 구축하였다. 실험 환경은 4종류의 서버들(클라이언트, LVS 기반의 부하 분산 서버, 클러스터 서버들 및 웹 서버)로 구성된다. 서버들의 사양은 <표 3>과 같다.

클라이언트들은 인터넷으로부터 얻을 수 있는 Webstone[19, 20]과 Surge[21, 22]를 이용해서 요청을 생성한다. 이러한 요청들은 텍스트 파일 및 이미지들로 구성된다. LVS 부하 분산기는 해싱 방법에 기반해서 요청들을 어디로 어떻게 보낼 것인지를 결정한다. 본 논문에서 논의된 6개의 해싱 방법(RR, MD5, MIT-CH, CARP, MOD-CH, 및 HT-CU(제안된 방법))들이 구현되었고, 이들의 결과를 토대로 성능을 분석할 것이다.

요청된 페이지들이 16개의 서버들 중 하나에 저장(캐싱)되면, 선택된 서버는 응답을 LVS를 통하지 않고 클라이언트에게 직접 보낸다. 응답을 할 때 LVS를 통하게 되면, 요청이 많아질 때 LVS가 요청과 응답을 모두 처리해야하기 때문에 클러스터의 전체적인 성능이 LVS에게 종속된다. 이를 해결하기 위해 본 실험에서는 LVS에서 지원하는 DR(Direct Routing) 방식을 사용하여 웹 서버가 응답을 클라이언트에게 직접 하도록 하였다. LVS는 들어오는 요청에 대한 처리만을 담당한다. 요청된 페이지가 캐시 서버에 저장되어 있지 않다면, 웹 서버에게 요청된다. 웹 서버로부터 새로운 컨텐츠를 받았다면, 이들은 선택된 캐시 서버에 저장(캐싱)된 후에 클라이언트에게 직접 전송될 것이다. Squid[23, 24]는 웹 서버로부터 받은 컨텐트를 저장(캐싱)하는데 사용된다.



(그림 2) 실험 환경

<표 3> 서버 사양

		하드웨어		소프트웨어	개수
		CPU (MHz)	RAM (MB)		
사용자 / 관리자		P-4 2260	256	Webstone, Surge	10 / 1
LVS		P-4 2400	512	DR	1
서버	캐시	P-2 400	256	Squid	16
	압축기			JPEG-6b	
웹 서버		P-4 2260	256	Apache	1

<표 4> 실험에 사용된 주요 변수 및 값

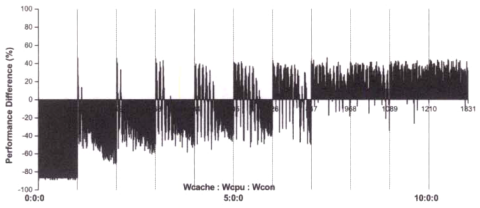
주요변수	값
캐시 서버 수	16
해싱 기반의 부하 분산 정책	RR, MD5, MIT-CH, CARP, MOD-CH, HT-CU (나머지 알고리즘은 기본적으로 MD5에 기반한 알고리즘이기 때문에 실험 결과의 특성을 고려하여 실험에서 제외함)
웹 트레이스 (Traces) (클라이언트의 요청 종류)	가상 웹 트레이스 : Surge (서로 다른 가중치를 가지는 100개의 이미지와 HTML 파일) 실제 웹 트레이스 : UNC [25, 26], Berkeley [27, 28] (서로 다른 가중치를 가지는 100개의 HTML 파일)
LVS를 통해 클라이언트에서 각 서버로 연결되는 동시 연결 개수	63
성능 지표	커넥션 수, 해시된 URL 수, CPU 이용률, 초당 처리된 요청 수

웹 서버로부터 받은 새로운 페이지가 저장될 때, 라운드 로빈 방식은 이들을 클러스터 내의 모든 서버에 저장한다. 반면, 다른 알고리즘은 선택된 하나의 캐시 서버에만 저장된다. 해당 요청에 대한 새로운 컨텐츠를 하나의 캐시 서버에만 저장하게 되면, 캐시 중복 문제를 없앰으로써 스토리지(저장 공간)의 확장성을 보장하게 된다. 라운드 로빈 방법은 비교 목적을 위해 사용된다. 왜냐하면, 모든 스케줄링 알고리즘 중에 가장 좋은 성능을 가지기 때문이다. 라운드 로빈 방법에서는 모든 서버가 동일한 파일들을 가지고 있음으로 특별한 요청 분배 방법이 필요 없고, 이는 특정 페이지에 대한 요청 집중 문제가 발생하지 않음을 의미한다.

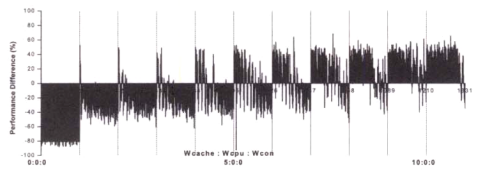
<표 4>는 실험에 사용된 주요 변수들을 나타낸다. 여러 변수들 중에 초당 처리되는 요청의 수는 제안된 방법들의 성능을 비교하는 중요한 변수로 사용된다. 실험에 사용된 웹 트레이스는 가상 웹 트레이스와 실제 웹 트레이스로 나눌 수 있다. 가상 웹 트레이스를 만들기 위해 사용된 소프트웨어는 Surge로서, 실제 사용자의 요청 패턴(작은 용량의 데이터를 많이 요청하고 큰 용량의 데이터를 적게 요청하는 패턴)과 유사한 웹 트레이스를 제공해 준다. 실제 웹 트레이스는 2001년도에 North Carolina 대학의 웹 분포 및 1998년도의 Berkely 대학의 웹 분포 중 100개를 샘플링 하여 사용하였다. 이러한 가상 및 실제 웹 트레이스들은 다른 여러 논문들에서 사용되는 대표적인 웹 트레이스들이다[21, 22, 25-28].

4.2 사전 실험

최적의 가중치를 찾기 위해 다음과 같은 실험을 수행하였다. 실험 조건은 16대의 캐시 서버 및 모든 가중치(W_{cache}, W_{cpu}, W_{con})를 0에서 10까지 변화시켜 가면서 테스트 (총 경우의 수는 1331 = 11 * 11 * 11) 하였다. 가중치의 변화는 W_{cache} W_{cpu} W_{con} (0 0 0 -> 0 0 1 -> 0 0 2 -> ... -> 0 0



(그림 3) 제안된 방법과 MD5 해싱의 성능 비교
(캐시 서버 동일 사양)



(그림 4) 제안된 방법과 MD5 해싱의 성능 비교
(캐시 서버 비동일 사양)

10 -> 0 1 0 -> 0 1 1 -> ... -> 0 10 10 -> 1 0 0 -> ... -> 10 10 10)로 변화시켰다. (그림 3)은 캐시 서버가 동일한 사양에서 대표적으로 사용되는 MD5 해싱과 제안된 방법의 성능 비교를 나타낸다. +%는 제안된 방법이 기존 방법에 비해 성능 향상이 있음을 의미하고, -%는 제안된 방법이 기존 방법에 비해 성능 감소가 있음을 의미한다.

(그림 4)는 캐시 서버가 동일하지 않은 사양에서 MD5 해싱과 제안된 방법의 성능 비교를 나타낸다. 캐시 서버가 동일하지 않은 사양을 만들기 위해 부하 발생기를 짝수 번째의(2, 4, ..., 16) 캐시 서버에서 실행하였다. 부하 발생기는 1-10초 사이에서 랜덤하게 CPU를 사용하거나 사용하지 않는 작업을 반복하도록 만들었다.

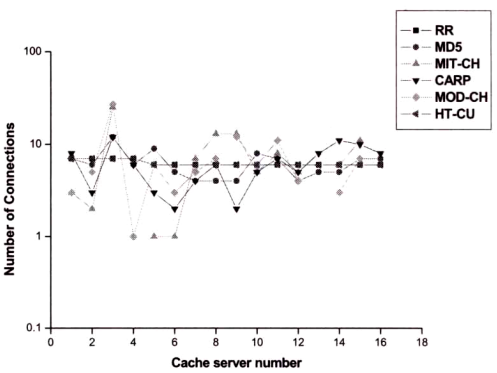
(그림 3)과 (그림 4)를 보면 주기성을 가지면서 MD5 보다 성능이 좋은 경우(Performance Difference가 +%인 경우)와 좋지 않은 경우(Performance Difference가 -%인 경우)로 나누어지는 것을 볼 수 있다. MD5 해싱 보다 성능이 좋아지는 경우는 캐시 이용률의 가중치(W_{cache})가 다른 가중치에 비해 상대적으로 높은 가중치를 가지는 경우이다. MD5 해싱에 비해 최고로 성능이 좋아지는 경우는 W_{cache} 가중치만 사용되고 나머지 가중치는 0인 경우이다. MD5 해싱 보다 성능이 나빠지는 경우는 캐시 이용률의 가중치(W_{cache})가 다른 가중치에 비해 상대적으로 낮은 가중치를 가지는 경우이다. MD5 해싱에 비해 최저로 성능이 나빠지는 경우는 다른 가중치만 사용되고 W_{cache} 가중치가 0인 경우이다.

가중치에 따라 이러한 성능이 나타나게 된 이유는 캐시 이용률의 정보는 부하 분산기(LVS)에서 업데이트됨으로 요청에 대한 캐시 선택에 실시간으로 이용될 수 있으나 CPU 이용률과 커넥션 수의 정보는 캐시 서버에서 주기적으로 전송해 오는 정보임으로 요청에 대한 캐시 선택에 실시간으로 이용될 수 없기 때문이다. 왜냐하면 현재 이용하려는 CPU 이용률과 커넥션 수의 정보는 이전 주기에서 얻은 과거 정보이기 때문이다. 이러한 잘못된 과거의 정보를 이용해서 캐시를 선택하게 되면 요청이 일부 캐시 서버로 집중되고, 이러한 집중은 전체 성능을 저하 시키는 결과를 가지고 오게 된다.

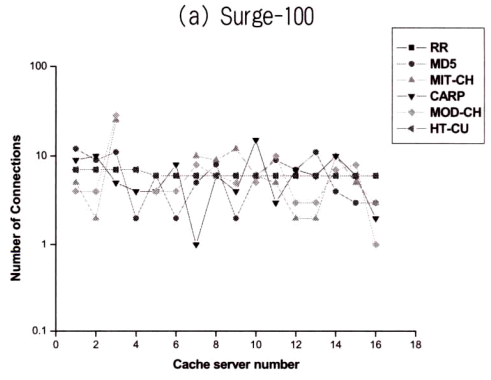
본 논문에서는 최적의 가중치를 구하기 위한 실험 결과를 토대로 다음과 같은 방법을 제안한다. 즉, 제안된 방법은 캐시 이용률을 이용해서 요청에 대해 캐시 서버를 할당하는 방법(W_{cache} = 10, W_{cpu} = 0, W_{con} = 0의 가중치를 가지는 방법)이다. 본 논문에서는 캐시 이용률을 이용하는 방법을 토대로 기존의 다른 방법들과 성능을 비교 및 분석한다.

4.3 커넥션 수

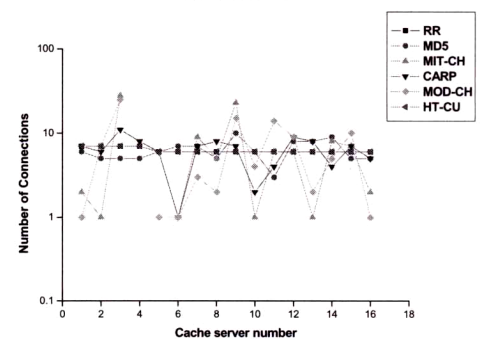
(그림 5)는 각 캐시 서버에 대한 커넥션 수를 보여준다. 여기서 하나의 커넥션은 한 개의 클라이언트 요청을 나타낸다. X 축은 서버의 수를 나타내고, Y 축은 커넥션의 수를 나타낸다. Y 축 가독성을 위해 로그 스케일로 그렸다.



(a) Surge-100



(b) UNC-2001



(c) Berkeley-1998

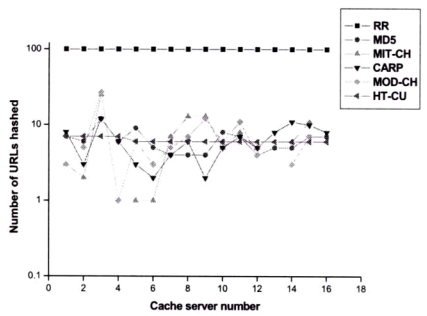
(그림 5) 커넥션 수

3개의 웹 Traces에 대해, 제안된 방법(HT-CU)이 16대의 캐시 서버들 사이에서 상대적으로 일정한 커넥션 수를 가진다. 반면, 나머지 다른 5개의 방법은 요청을 캐시 서버에 균일하게 분포시키지 못하는 해싱 방법으로 인해 커넥션 수가 일정하지 않고 평균에서 높낮이를 반복하게 된다. 이러한 평균으로부터의 높낮이는 전체적인 클러스터의 성능에 영향을 미치게 된다.

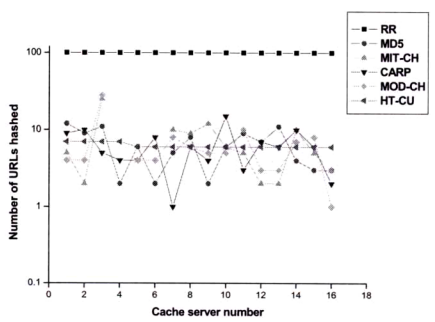
4.4 해시된 URL 수

(그림 6)은 각 캐시 서버에 대한 해시된 URL의 수를 보여준다. X 축은 서버의 수를 나타내고, Y 축은 해시된 URL의 수를 나타낸다. Y 축은 가독성을 위해 로그 스케일로 그려졌다.

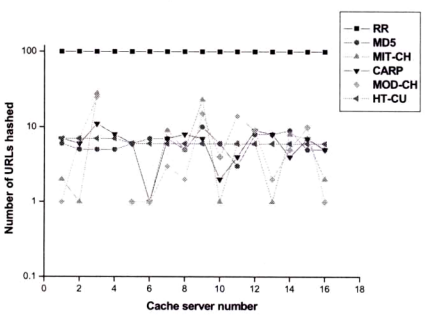
그림을 보면 제안된 방법이 모든 캐시 서버들 사이에서 상대적으로 일정한 해시된 URL의 수를 가지는 것을 보여준다. 반면, 모든 다른 방법들은 해시된 URL의 수가 평균으로부터 높낮이를 가지는 것을 보여준다.



(a) Surge-100



(b) UNC-2001



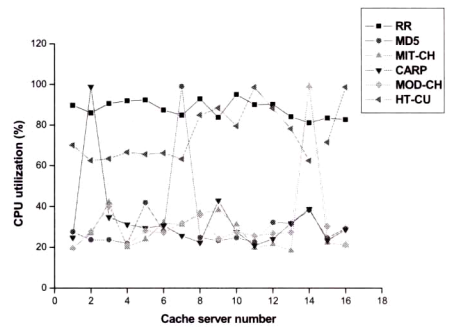
(c) Berkeley-1998

(그림 6) 해시된 URL 수

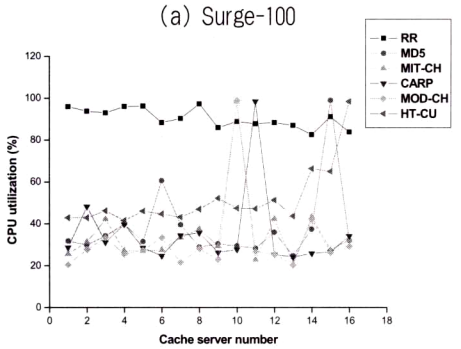
4.5 CPU 이용률

(그림 7)은 CPU 이용률을 보여준다. 이러한 이용률은 리눅스 유틸리티 중의 하나인 vmstat를 이용하여 측정되었다. X 축은 캐시 서버의 수를 나타내고, Y 축은 CPU 이용률(%)를 나타낸다.

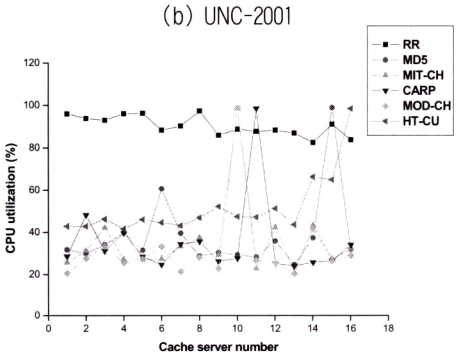
그림에서 보이듯이, 제안된 방법의 CPU 이용률이 라운드 로빈(RR)을 제외하고 다른 방법들에 비해 높음을 알 수 있다. 이유는 제안된 방법이 클라이언트의 요청을 균일하게



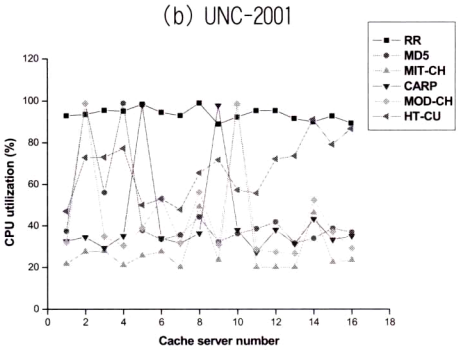
(a) Surge-100



(b) UNC-2001



(b) UNC-2001



(c) Berkeley-1998

(그림 7) CPU Utilization

캐시 서버에 분포시킴으로써 캐시 서버들을 균일하게 활용하는 반면, 다른 방법들을 일부 특정 캐시 서버만 활용하고 대부분의 캐시 서버들을 제대로 활용하지 못하기 때문이다. 라운드 로빈은 3개의 traces에 대해 모든 서버들이 매우 높은 CPU 이용률을 가지는데, 이는 라운드 로빈이 모든 동일 컨텐츠를 모든 캐시 서버에 저장하는 반면, 다른 방법들이 그렇지 않기 때문이다.

5. 성능 분석

5.1 표준 편차

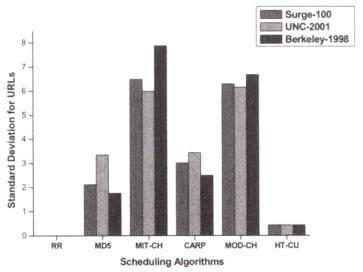
표준 편차는 하나의 샘플이 모든 샘플의 평균으로부터 얼마나 떨어져 있는가를 보여준다. 이를 이용해서 각각의 해싱 방법이 다른 방법들에 비해 커넥션 수, 해시된 URL의 수 및 CPU 이용률 관점에서 얼마나 효율적인가를 보고자 하였다. 각 해싱 방법에 대해 표준 편차는 각 서버로부터 얻어진 16개의 값을 사용해서 계산되었다. 각각의 표준 편차는 각 서버가 커넥션 수, 해시된 URL 수 및 CPU 이용률의 평균값에서 얼마나 떨어져 있는가를 나타낸다. 표준 편차가 작으면 작을수록, 더 좋은 해싱 방법임을 나타낸다. 왜냐하면, 이것은 각 캐시 서버로 요청이 균일하게 분포되었음을 의미하기 때문이다. (그림 8)는 모든 해싱 방법들의 표준 편차들을 나타낸다.

(그림 8) (a)는 각 캐시 서버로 해시된 URL의 표준 편차를 나타낸다. 그림에서 볼 수 있듯이, 제안된 방법(HT-CU)이 6개의 방법들 중에 라운드 로빈(RR)을 제외하고 가장 작음을 볼 수 있다. 라운드 로빈은 그림에 나타나지 않았는데, 이는 라운드 로빈의 표준 편차가 0임을 의미한다. 라운드 로빈 방법은 각 캐시 서버로 정확하게 동일한 수의 URL의 해시함으로써 표준 편차가 0이 된다. (그림 8) (b)에서 보이는 커넥션 수의 표준 편차는 해시된 URL 수의 표준 편차와 약간 다르다. 여기서 라운드 로빈은 약간의 표준 편차를 가지고, 이는 커넥션 수가 서버마다 약간 차이가 남을 의미한다. 그림은 분명하게 제안된 방법(HT-CU)이 라운드 로빈을 제외한 다른 방법들에 비해 우수한 성능을 가짐을 나타낸다.

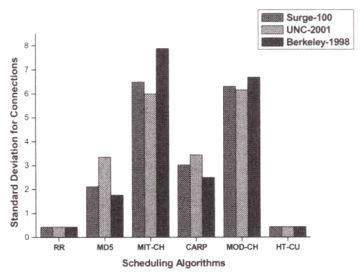
5.2 확장성

확장성은 초당 처리된 요청의 수로서 측정된다. (그림 9)는 Surge-100 trace를 사용하여 측정된 모든 방법들의 확장성을 보여준다. X 축은 캐시 서버의 개수를 나타내고, Y 축은 초당 처리된 요청의 수를 나타낸다.

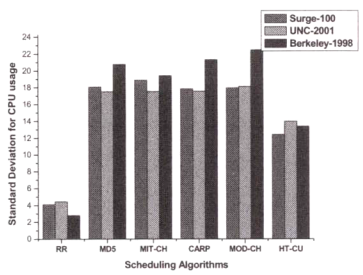
(그림 9)에서 왼쪽 그림은 초당 처리된 요청의 개수를 나타내고, 오른쪽 그림은 왼쪽 그림을 캐시 서버 당 처리 속도(Speedup)로 바꾼 것을 나타낸다. 한 예로, Surge-100 trace에 대해 제안된 해싱 방법(HT-CU)을 고려해보자. 서버의 수가 1개에서 2개로 늘어날 때 초당 처리된 요청의 수는 250에서 350으로 늘어난다. 이는 약 50% 정도 성능이 증가하였음을 나타낸다. 오른쪽 그림에서 보이는 전체적인 캐시 서버당 처리 속도는 제안된 방법이 16대의 캐시 서버 상에서 약 6배의 성능이 증가됨을 나타낸다. 반면, 다른 방법들은 약 2배의 성능 증가만이 있다. 라운드 로빈(RR)은 예상대로 가장 높은 확장성을 보여준다. 이것은 라운드 로빈



(a) Standard deviation for URLs

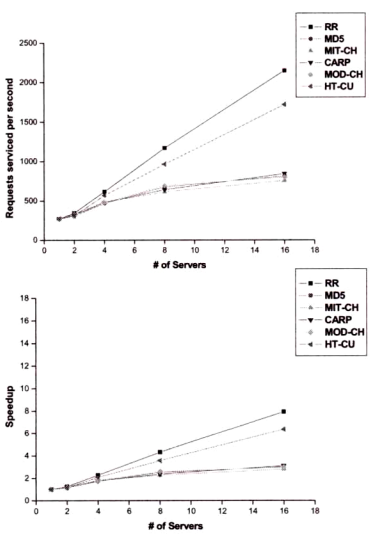


(b) Standard deviation for connections



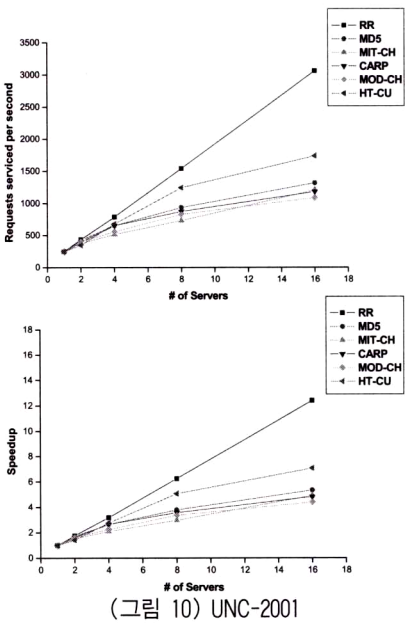
(c) Standard deviation for CPU usage

(그림 8) 표준 편차

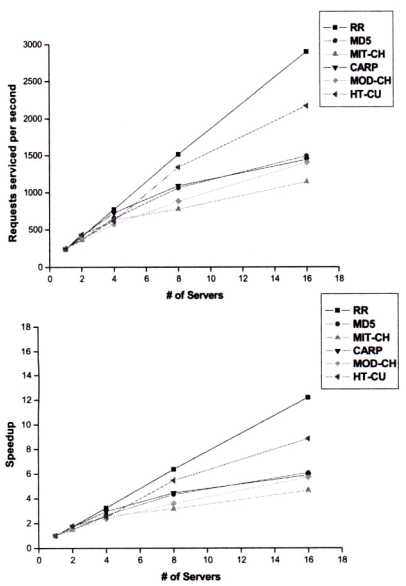


(그림 9) Surge-100

이 동일 파일들을 모든 캐시 서버에 저장하기 때문이며, 결과적으로 스토리지(저장 공간) 확장성의 제약을 가지고 온다. 비슷한 확장성이 다른 2개의 trace((그림 10)의 UNC-2001, (그림 11)의 Berkeley-1998)에서도 나타난다. 3개의 그림에서 보이듯이, 제안된 해싱 방법들이 기존의 다른 방법들에 비해 확장성(성능)을 가지는 것을 알 수 있다. 이는 캐시 이용률과 같은 동적 서버 정보가 요청(URL)을 캐시 서버에 할당할 때 꼭 필요한 정보임을 나타낸다.



(그림 10) UNC-2001



(그림 11) Berkeley-1998

6. 결 론

제안된 방법은 클라이언트가 요청한 URL의 해시 값을 이용하여 캐시 서버를 선택할 때, 캐시 이용률을 이용하는 것이다. 이러한 방법을 이용하게 되면 클라이언트가 요청한 URL의 해시 값의 특성에 영향을 받지 않고 요청된 URL을 캐시 서버에 균일하게 할당할 수 있다. 사용자 요청 URL을 캐시 서버에 균일하게 할당하게 되면 시스템이 전체적으로 확장성을 가지게 되며, 이는 시스템이 높은 성능을 유지하도록 만들어준다. 해시 기반의 부하 분산의 특성상 하나의 URL로 요청이 집중되는 경우가 발생하면, 요청 집중이 발생한 해당 URL 요청에 대해서만 라운드 로빈 부하 분산을 사용함으로써 요청 집중 문제를 해결할 수 있다.

제안된 방법을 테스트하기 위해 우리는 16대의 서버로 이루어진 클러스터, LVS 부하 분산기, 10대의 클라이언트들과 하나의 웹 서버로 구성된 실험 환경을 사용하였다. 클라이언트들은 인터넷을 통해 얻은 웹 Traces(Berkeley-1998 [27, 28], UNC-2001 [25, 26], Surge-100 [21, 22], Webstone [19, 20] 및 다양한 크기의 이미지와 HTML 페이지들)에 기반하여 요청들을 생성하였다. 실험 결과는 제안된 방법이 기존 방법들에 비해 해시된 URL 수, 커넥션 수, CPU 이용률의 균일 분포(전체적인 표준 편차가 낮음) 및 확장성 측면에서 우수함을 알 수 있다.

제안된 방법은 요청 URL을 캐시 서버에 할당 할 때 캐시 이용률을 이용하는 것이다. 캐시 이용률의 경우 추가적인 계산 시간(요청 URL을 캐시 서버에 할당 할 때마다 가장 적은 캐시 이용률을 가지는 캐시 서버를 찾아내는 시간)이 필요하다. 기존 방법의 경우 캐시간 협동성을 위해 해시 기반 부하 분산을 사용할 경우 낮은 확장성과 성능을 가지게 된다. 이는 요청 URL을 해시 값을 이용하여 캐시 서버에 할당 할 때 특정 캐시 서버들로 요청 URL이 몰리는 해시의 특성에 기인한다. 제안된 방법은 요청 URL을 캐시 서버에 할당할 때 해시의 특성에 영향을 받지 않게 함으로써 요청 URL을 캐시 서버에 균일하게 할당하였다. 이러한 균일한 할당은 높은 확장성과 성능을 보장한다.

참 고 문 헌

[1] D. Zeng, F. Wang, and M. Liu, "Efficient web content delivery using proxy caching techniques", IEEE Transactions on Systems, Man and Cybernetics, Vol. 34, No. 3, pp. 270-280, 2004.

[2] J. Challenger, P. Dantzig, A. Iyengar, M. Squillante, and L. Zhang, "Efficient serving dynamic data at highly accessed web sites", IEEE/ACM Transactions on Networking, Vol. 12, No. 2, pp. 233-246, 2004.

[3] P. Trianfillou and I. Aekaterinidis, "ProxyTeller: a proxy placement tool for content delivery under performance constraints", Proceedings of the 4th International Web Information Systems Engineering, pp. 62-71, 2003.

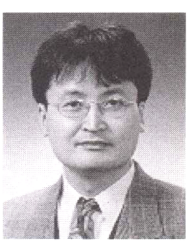
[4] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks", IEEE Transactions on Mobile Computing, Vol. 5, No. 1, pp. 77-89, 2006.

[5] J. Ni and D. Tsang, "Large-scale cooperative caching and application-level multicast in multimedia content delivery networks", IEEE Communications Magazine, Vol. 43, No.

5, pp. 98-105, 2005.
[6] G. Cao, L. Yin, and C. Das, "Cooperative cache-based data access in ad hoc networks", IEEE Computer, Vol. 37, No. 2, pp. 32-39, 2004.
[7] L. Ramaswamy and L. Liu, "An expiration age-based document placement scheme for cooperative Web caching", IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 5, pp. 585-600, 2004.
[8] X. Fu and L. Yang, "Improvement to HOME based Internet caching protocol", IEEE 18th Annual Workshop on Computer Communications, pp. 159-165, 2003.
[9] H. Mei, C. Lu, and C. Lai, "An automatic cache cooperative environment using ICP", International Conference on Information Technology: Coding and Computing, pp. 144-149, 2002.
[10] C. Chan, S. Huang, and J. Wang, "Cooperative cache framework for video streaming applications", International Conference on Multimedia and Expo, pp. 313-316, 2003.
[11] L. Ramaswamy and L. Liu, "A new document placement scheme for cooperative caching on the internet", Proceedings of 22nd International Conference on Distributed Computing Systems, pp. 95-103, 2002.
[12] D. Rivest, "The MD5 Message Digest Algorithm", RFC 1321, 1992.
[13] David Karger and al. "Web Caching with consistent hashing", In WWW8 conference, 1999.
[14] Micorsoft Corp., "Cache Array routing protocol and microsoft proxy server 2.0", White Paper, 1999.
[15] F. Baboescu, "Proxy Caching with Hash Functions", Technical Report CS2001-0674, 2001.
[16] Toyofumi Takenaka, Satosi Kato, and Hidetosi Okamoto, "Adaptive load balancing content address hashing routing for reverse proxy servers", IEEE International Conference on Communications, Vol. 27, No. 1, pp. 1522-1526, 2004.
[17] S. Lei and A. Grama, "Extended consistent hashing: an efficient framework for object location", Proceeding of 24th International Conference on Distributed Computing Systems, pp. 254-262, 2004.
[18] L. Ramaswamy, Ling Liu, and A. Iyengar, "Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks", Proceedings of 25th IEEE International Conference on Distributed Computing Systems", pp. 229-238, 2005.
[19] Mindcraft, Inc., "WebStone : The Benchmark for Web Server", http://www.mindcraft.com/web- stone.
[20]] J. Nakano, P. Montesinos, K. Gharachorloo, and J. Torrellas, "ReViveI/O: efficient handling of I/O in highly-available rollback-recovery servers", The 12th Internation Symposium on High-Performance Computer Architecture, pp. 200-211, 2006.
[21] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", In Proc. ACM SIGMETRICS Conf., Madison, WI, Jul. 1998.
[22] R. Zhang, T. Abdelzaher, and J. Stankovic, "Efficient TCP connection failover in Web serer clusters", 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1219-1228, March 2004.
[23] Squid Web Proxy Cache, http://www.squid-cache.org.
[24] W. Liao and P. Shih, Architecture of proxy partial caching using HTTP for supporting interactive video and cache consistency, 11th International Conference Computer Communications and Networks, 2002, pp. 216-221.
[25] H. Felix, K. Jeffay, and F. Smith, "Tracking the Evolution of Web Traffic", Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 16-25, 2003.
[26] D. Lu, Y. Qiao, P. Dinda and F. Bustamante, "Modeling and Taming Parallel TCP on the Wide Area Network", Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, April 2005.
[27] B. A. Mah, "An Empirical Model of HTTP Network Traffic", Proceedings of INFOCOM, pp. 592-600, 1997.
[28] J. Xu and W. Lee, "Sustaining availability of Web services under distributed denial of service attacks", IEEE Transactions on Computers, Vol. 52, No. 2, pp. 195-208, Feb. 2003.

곽 후 근

e-mail : gobarian@q.ssu.ac.kr
1996년 호서대학교 전자공학과(학사)
1998년 숭실대학교 전자공학과 대학원 (석사)
1998년~2006년 숭실대학교 전자공학과 대학원(박사)
1998년 8월~2000년 7월 : (주) 3R 부설 연구소 주임 연구원
2006년 3월~현재 숭실대학교 전자공학과 대학원 (postdoc)
관심분야 : 네트워크 컴퓨팅 및 보안



정 규 식

e-mail : kchung@q.ssu.ac.kr
1979년 서울대학교 전자공학과(공학사)
1981년 한국과학기술원 전산학과 (이학석사)
1986년 미국 University of Southern California (컴퓨터공학석사)
1990년 미국 University of Southern California (컴퓨터 공학박사)
1998년 2월~1999년 2월 미국 IBM Almaden 연구소 방문 연구원
1990년 9월~현재 숭실대학교 정보통신전자공학부, 교수
관심분야 : 네트워크 컴퓨팅 및 보안