무선 인터넷을 위한 캐시 서버 클러스터 환경에서 캐시 이용률 기반의 스케줄링

(Scheduling based on Cache Utilization in a Cache Server Cluster for Wireless Internet)

곽 후 근 † 정 규 식 ††
(Hukeun Kwak) (Kyusik Chung)

요 약 웹 페이지를 캐싱하는 것은 웹 하부 구조 상에서 중요한 역할을 한다. 캐싱 서비스의 효과는 제안된 대역폭을 가지는 무선 하부 구조 상에서 더욱 중요하게 여겨진다. 큰 규모의 하부 구조에서는 캐싱에서 발생할 수 있는 확장성과 요청 집중 현상(Hot-Spot) 문제를 해결하기 위해 서버들을 클러스터로 구성한다. 이에 본 논문에서는 무선 인터넷 프록시 서버 클러스터 환경에서 캐시 이용률 기반의 스케줄링 기법을 제안한다. 제안된 방법은 클라이언트의 요청을 캐시 서버 클러스터로 균일하게 분포시키고 요청 몰림 현상을 해결하기 위해 캐시 이용률을 이용하였다. 제안된 방법은 리눅스 클러스터 상에서 구현하였고, 실제로 사용되는 다양한 웹 traces들을 이용하여 실험을 수행하였다. 16대의 캐시 서버에서 수행된 실험 결과는 제안된 해싱 기법이 요청 집중 현상을 해결하면서, 기존에 많이 사용되는 방법들에 비해 45%에서 114%까지 성능이 향상됨을 확인하였다.

키워드 : 캐시 서버, 클러스터, 스케줄링, 해싱, 동적 서버 정보

Abstract Caching web pages is an important part of web infrastructures. The effects of caching service are even more pronounced for wireless infrastructures due to their limited bandwidth. Medium to large-scale infrastructures deploy a cluster of servers to solve the scalability problem and hot spot problem inherent in caching. In this paper we present scheduling scheme based on cache utilization in a wireless internet proxy server cluster environment. The proposed method uses cache utilization for distributing evenly client requests to a cluster of cache servers and solving hot spot problem. We have implemented our approach and performed various experiments using publicly available traces. Experimental results on a cluster of 16 cache servers demonstrate that the proposed hashing method gives 45% to 114% performance improvement over other widely used methods while addressing the hot spot problem.

Key words : Cache server, Cluster, Scheduling, Hashing, Dynamic Server Information

1. 서 론

웹 사이트를 운영할 때 캐싱(Caching)을 사용하지 않는다면, 사용자가 매번 웹 사이트를 방문해야 함으로 요청에 대한 응답 시간이 길어지고, 웹 사이트에 요청이 몰리거나(Hot-Spot) 죽으면 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 몰리거나 죽어도 일시적으로는 요청에 대한 응답을 받을 수 있다[3]. 캐시간 협동성(Cache Cooperation 또는 Cooperative Caching)[4]을 사용하지 않는다면, 웹 사이트를 구성하는 서버들이 모든 데이타를 중복해서 저장해야 한다. 데이타의 크기가 늘어나면 전체 데이타의 합은(서버의 수 × 데이타의 크기)가 된다. 캐시간 협동성을 사용한다면, 웹 사이트를 구성하는 서버들이 모든 데이타를 나누어서 저장한다. 데이타의 크기가 늘어나도 전체 데이타의 합은 (1 × 데이타의 크기)로 일정하게 유지된다. 캐시간 협동성을 위한 방법에는 프로토콜을 이용하는 방법과 해싱을 사용하는 2가지

· 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11167-0) 지원 및 숭실대학교 교내 연구비 지원으로 이루어졌음
† 정 회 원 : 숭실대학교 정보통신전자공학부 대학원(postdoc)
gobarian@q.ssu.ac.kr
†† 종신회원 : 숭실대학교 정보통신전자공학부 교수
kchung@q.ssu.ac.kr
논문접수 : 2007년 6월 13일
심사완료 : 2007년 7월 24일

방법이 존재한다. 프로토콜을 이용하는 방법[5]은 요청을 받은 캐시 서버가 데이타를 가지고 있지 않다면, 주변의 캐시 서버에게 해당 데이타를 가지고 있는지 물어본다. 만약, 주변 캐시 서버가 클라이언트의 요청 데이타를 가지고 있다면 그 데이타를 사용하고, 가지고 있지 않다면 웹 서버에게 데이타를 요청한다. 해싱을 이용하는 방법[6-12]은 사용자가 요청한 데이타(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 분산기(Load Balancer)가 수행하게 된다.

기존 방법들의 단점은 요청을 캐시 서버에 균일하게 분포시키는 문제, 확장성 문제, 요청 몰림(Hot-Spot) 처리 문제로 나누어서 생각할 수 있다. 균일 분포(Evenly Distribution)의 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 분포하지 않으면, 특정 캐시 서버들로 요청이 집중된다. 특정 캐시 서버들로만 요청이 집중되면 캐시 서버의 전체적인 성능은 특정캐시 서버에 의존하게 됨으로 전체적인 확장성은 떨어지게 된다. 요청 몰림(Hot-Spot) 처리의 문제에서는 클라이언트가 요청한 동일한 URL은 동일한 캐시 서버에게 할당이 된다. 이때, 특정 URL로 요청이 몰리는 경우(Hot-Spot) 캐시 서버의 전체적인 성능은 특정 URL을 처리하는 캐시 서버에 의존하게 된다.

본 논문에서 제안한 방법은 캐시 이용률에 기반하여 해싱을 수행하는 방법이다. 균일 분포와 확장성을 위해 해시 값으로 캐시 서버를 선택할 때 캐시 이용률 정보를 이용하고, 요청 몰림(Hot-Spot) 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 해싱[6]은 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[7]의 캐시 포인트는 이진트리(Binary Tree) 형식으로 저장될 수 있고, 캐싱된 URL에 대한 검색은 이진트리 내에서 단일 검색(Single Searching) 방식이 이용된다. N개의 캐시 서버에서 검색을 위한 시간이 O(logn) 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 몰림 현상(Hot-Spot)에 대처할 수 없는 단점을 가진다.

2.3 CARP (Cache Array Routing Protocol)

CARP[8]는 일반적인 해시 기반 부하 분산 방식과 같이 ICP(Inter Cache Protocol) 요청(Query) 메시지 없이 해시에 의하여 데이타를 캐싱하고 있는 서버로 요청을 할당하며, 데이타의 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 몰림 현상(Hot-Spot)에 대해서 대처할 수 없는 단점을 가진다.

2.4 MOD-CH (Modified Consistent Hashing)

MOD-CH[9]는 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 상황

을 반영하지 못하기 때문에 순간적인 요청 몰림 현상(Hot-Spot)에 대해서 대처할 수 없는 단점을 가진다.

2.5 Adaptive-LB (Adaptive Load Balancing)

Adaptive-LB[10]는 2004년에 제안된 방식으로서 기존의 방식과 달리 해시 기반 부하 분산과 함께 라운드 로빈 부하 분산 방식을 동시에 사용하여 부하 분산 한다. 이 방식은 기본적으로 해시 기반 부하 분산 방식을 사용하지만, 요청 몰림(Hot-Spot) 상황의 처리에 초점을 둔다. 즉, 가장 자주 요청되는 데이타에 대해서 목록으로 관리하고, 해당요청이 발생하면 전체 캐시 서버에 라운드 로빈으로 분산시키는 방법을 사용한다. Adaptive-LB는 다른 해시 기반 부하 분산 방식과 달리 요청 몰림 현상(Hot-Spot)에 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[11]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 몰림 현상(Hot-Spot)을 처리할 수 있다. EXT-CH는 요청 컨텐츠의 요청 빈도 및 이들의 평균 부하에 따라 캐시 서버 할당 영역을 설정하여 요청 몰림 현상(Hot-Spot)에 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[12]는 기존의 일반적으로 작은 캐시 클러스터 그룹 환경에서 운영되는 해시 기반 부하 분산 방식과 달리 네트워 끝단(Edge) 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 사용하지 않았던 캐시 서버의 부하 정보를 반영한 동적 해싱 방법을 사용한다. 그러나 요청의 균일한 분포 문제와 확장성에 따른 성능 증가 및 요청 몰림 문제에 대해서 고려되지 않은 것을 볼 수 있다.

3. 캐시 이용률 기반의 스케줄링 (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. 본 논문에서는 동적인 캐시 서버의 정보로써 캐시의 이용률을 이용한다.

캐시 서버에 요청을 할당하는 것은 각 서버의 캐시 이용률 중 가장 작은 값을 찾음으로써 이루어진다. 이러한 캐시 이용률 정보는 모든 해시 값과 이들이 할당되는 서버들이 모든 캐시 서버들 사이에서 균일하게 분포시킨다. 그림 1은 이러한 제안된 방법을 보여준다. 여기

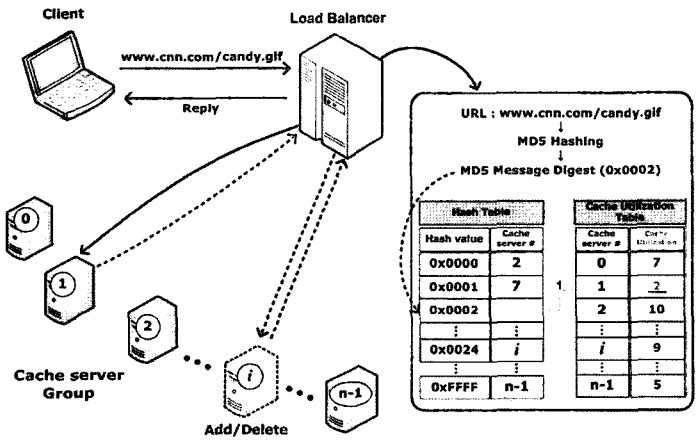


그림 1 캐시 이용률 기반의 스케줄링

에 2개의 테이블이 존재한다. 첫 번째 테이블은 해시 값과 이들에게 할당된 캐시 서버로 구성되고, 두 번째 테이블은 캐시 이용률 정보들로 구성된다. 이러한 두 번째 캐시 이용률 정보 테이블에는 각 캐시 서버의 캐시의 이용률 정보가 들어가 있다.

3.2 동작 과정

하나의 요청(URL)을 받았다고 가정하자. MD5 해싱은 0×0000에서 0×FFFF 범위에서 하나의 해시 값을 생성할 것이다. 위의 예에서 해시 값은 0x0002이다. 해시 테이블 엔트리에 이미 캐시 서버가 할당되어 있다면, 요청은 해당 서버로 보내질 것이다. 그러나 엔트리가 비어있고, 캐시 서버가 할당되지 않았다면, 캐시 서버는 캐시 이용률 정보 테이블을 검색하고 각 서버의 캐시 이용률 중 최소 값을 찾음으로써 할당될 것이다. 위 그림에 따르면, 캐시 서버 1이 캐시 서버들 중에 캐시 이용률이 가장 적은 값을 가지게 되고 현재의 요청에 대한 서버로써 선택되어진다.

모든 캐시 서버들이 캐시 이용률을 통해 같은 값을 가지게 되면, 캐시 서버의 할당은 캐시 서버 0(서버 번호가 가장 작은 값)이 된다. 이러한 캐시 이용률의 사용은 부하 분산기가 요청들을 모든 캐시 서버들 사이에서 균일하게 분포되도록 한다. 만약 요청 몰림이 발생하게 되면 요청 몰림을 발생시킨 해당 URL을 하나의 캐시 서버(해당 URL을 처리했던)에서 처리하는 것이 아니라, 모든 캐시 서버들이 요청 몰림 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 몰림 URL을 공평하게 분산해서 처리한다. 시간이 흘러 요청 몰림 URL이 없어지면 해당 URL은 다시 이를 처리했던 원래의 캐시 서버가 다시 처리를 담당하게 된다.

3.3 기존 방법과의 비교

표 1은 기존 방법과 제안된 방법을 알고리즘 복잡도, 서버 추가/삭제, 확장성, 균일 분포 및 요청 몰림(Hot-Spot) 관점에서 비교한 표이다.

4. 실험 결과

4.1 실험 환경

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

표 3은 실험에 사용된 주요 변수들을 나타낸다. 여러 변수들 중에 초당 처리되는 요청의 수는 제안된 방법들의 성능을 비교하는 중요한 변수로 사용된다.

본 논문에서 논의된 6개의 해싱 방법(RR, MD5, MIT-CH, CARP, MOD-CH, 및 HT-CU(제안된 방법))들이 구현되었고, 세 가지 요청 파일을 대상으로 실험을 수행하였다. 첫 번째는 요청 하는 파일들이 가중치를 가지지 않는 가장 이상적인 경우로써, 모든 파일들이 동일한 가중치를 가지고 요청된다. 두 번째는 요청 하는 파일들이 가중치를 가지는 가장 현실적인 경우로써, 각

표 1 기존 방법과 제안된 방법

방법	알고리즘 복잡도	서버 추가/삭제	확장성	균일 분포	요청 몰림 (Hot-Spot)
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

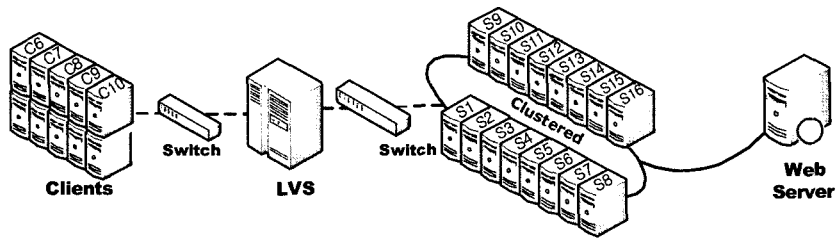


그림 2 실험 환경

표 2 서버 사양

		하드웨어		소프트웨어	개수
		CPU (MHz)	RAM (MB)		
사용자 / 관리자		P-4 2260	256	Webstone[13], Surge[14]	10 / 1
LVS		P-4 2400	512	DR	1
서버	캐시	P-2 400	256	Squid[15]	16
	압축기			JPEG-6b	
웹 서버		P-4 2260	256	Apache	1

표 3 실험에 사용된 주요 변수 및 값

주요 변수	값
캐시 서버 수	16
해싱 기반의 부하 분산 정책	RR, MD5, MIT-CH, CARP, MOD-CH, HT-CU
웹 Traces (클라이언트의 요청 종류)	실제 웹 Traces : UNC [16], Berkeley [17] (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

파일들이 각각의 가중치를 가진다. 이때, 실제 사용자의 요청 파일 분포를 모델링 하여 적은 크기의 파일에는 높은 가중치를 큰 크기의 파일에는 낮은 가중치를 주었다. 마지막으로 세 번째는 요청 집중이 발생하는 최악의 경우로써, 모든 파일들이 동일한 가중치를 가지고 요청되는 가운데 특정 파일에 대해서만 요청을 집중하도록 하였다.

4.2 가중치를 가지지 않은 경우 (이상적인 경우)

표 4와 그림 3은 300 Bytes에서 100 Kbytes의 크기를 가지는 이미지를 가중치 없이 클라이언트가 요청을 보냈을 때, 무선 인터넷 프록시 서버가 초당 처리하는 요청의 개수를 나타낸다. 표에서 Variation은 1 Kbytes에서 10 Kbytes의 크기의 이미지들을 랜덤하게 요청한 경우이다. 그림에서 보면 MD5의 경우 서버의 개수가 증가해도 초당 처리수가 비례적으로 증가하지 않는 반면에, RR과 제안된 방법(HT-CU)은 서버의 개수가 증

표 4 호스트 개수에 따른 초당 요청수

(a) RR

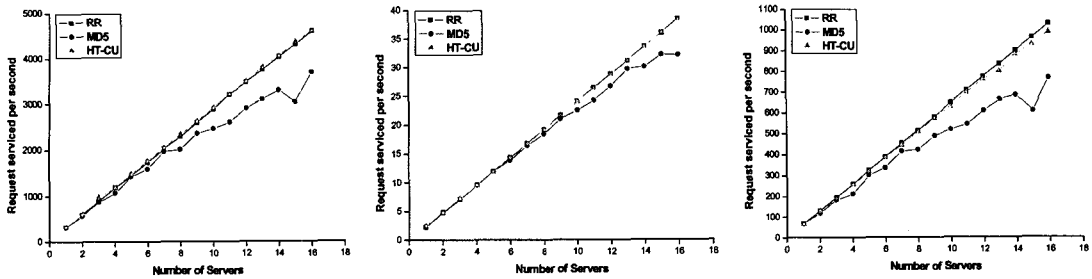
호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	316	596	895	1188	1446	1732	2025	2285	2598	2890	3212	3474	3750	4034	4315	4603
1 Kbytes	233	444	664	882	1090	1282	1516	1732	1921	2137	2353	2577	2799	3026	3244	3458
10 Kbytes	36	71	110	143	179	215	251	287	322	358	393	429	465	501	539	574
100 Kbytes	2.35	4.82	7.16	9.57	11.94	14.29	16.76	19.15	21.58	23.97	26.37	28.78	31.16	33.59	35.99	38.46
Variation	66	129	193	258	323	386	451	513	578	646	709	772	834	898	961	1026

(b) MD5

호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	316	564	873	1069	1430	1589	1964	2013	2362	2470	2607	2908	3103	3305	3046	3694
1 Kbytes	233	415	636	771	1056	1160	1455	1489	1735	1809	1931	2148	2277	2390	2195	2703
10 Kbytes	36	67	104	123	172	191	231	232	277	297	311	342	373	388	345	434
100 Kbytes	2.15	4.92	7.15	9.54	11.93	13.85	16.38	18.32	20.99	22.49	24.20	26.71	29.68	30.03	32.20	32.16
Variation	65	117	181	211	299	336	414	421	486	517	544	608	662	684	608	766

(c) 제안된 방법(HT-CU)

호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 Bytes	317	599	983	1191	1487	1766	2051	2340	2626	2923	3220	3491	3815	4055	4371	4600
1 Kbytes	232	441	662	878	1093	1304	1510	1748	1958	2136	2401	2606	2793	3038	3254	3456
10 Kbytes	36	71	109	143	181	216	253	285	325	360	397	431	467	501	546	567
100 Kbytes	2.47	4.83	7.36	9.53	11.90	14.34	16.70	19.15	21.55	23.96	26.33	28.75	31.18	33.57	36.00	38.42
Variation	65	127	191	256	319	383	443	506	572	626	696	758	798	881	925	987



(a) 300 Bytes (b) 100 Kbytes (c) Variation

그림 3 호스트 개수에 따른 초당 요청수

가함에 따라 초당 요청수가 비례적으로 증가하는 것을 볼 수 있다. 이는 MD5 해쉬의 특성으로 인해 클라이언트의 요청이 캐시 서버들 사이로 균일하게 분포하지 않고, 이로 인해 전체적인 캐시 서버의 성능이 일부 캐시 서버에 종속된다(저장 공간상의 확장성을 가지지만, 성능상의 확장성을 가지지 않음). 이에 반해 제안된 방법은 해싱에 라운드 로빈 방법을 적용하여 클라이언트의 요청을 캐시 서버들 사이로 균일하게 분포시킴으로써 기존 방법에 비해 전체적인 캐시 서버의 성능이 크게 향상되었음을 볼 수 있다(성능과 저장 공간상의 확장성을 가짐). RR은 비교의 목적으로 실험된 것으로, 모든 캐시 서버가 동일 클라이언트 요청 데이타를 가짐으로써 가장 좋은 성능을 나타냄을 볼 수 있다(성능상의 확장성을 가지지만, 저장 공간상의 확장성을 가지지 않음).

4.3 가중치를 가지는 경우 (실제의 경우)

(1) 커넥션 수

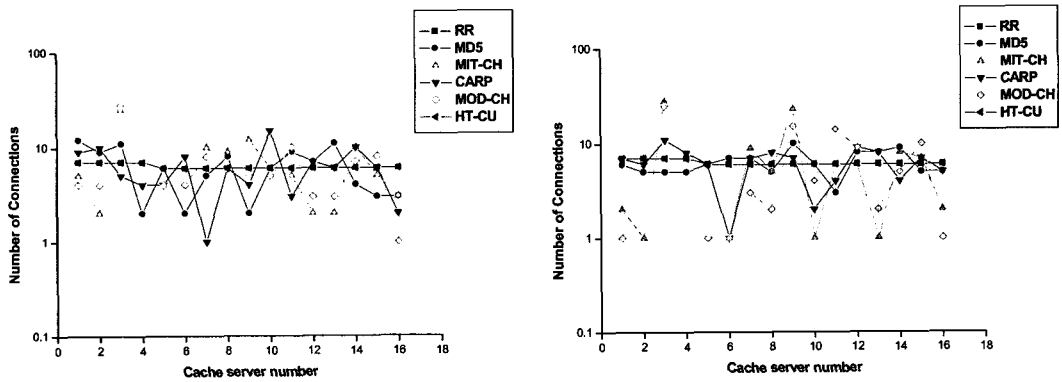
그림 4는 각 캐시 서버에 대한 커넥션 수를 보여준다. 여기서 하나의 커넥션은 한 개의 클라이언트 요청을 나타낸다. X 축은 서버의 수를 나타내고, Y 축은 커넥션의 수를 나타낸다. Y 축 가독성(Readability)을 위해 로그(Log) 스케일로 그렸다. 2개의 웹 Traces에 대해, 제안된 방법(HT-CU)이 16대의 캐시 서버들 사이에서 상대적으로 일정한 커넥션 수를 가진다. 반면, 나머지 다른 5개의 방법은 요청을 캐시 서버에 균일하게 분포시키지 못하는 해싱 방법으로 인해 커넥션 수가 일정하지 않고 평균에서 높낮이를 반복하게 된다(Fluctuation). 이러한 평균으로부터의 높낮이는 전체적인 클러스터의 성능에 영향을 미치게 된다.

(2) 해시된 URL 수

그림 5는 각 캐시 서버에 대한 해시된 URL의 수를 보여준다. X 축은 서버의 수를 나타내고, Y 축은 해시된 URL의 수를 나타낸다. Y 축이 가독성(Readability)을 위해 로그(Log) 스케일로 그려졌음을 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 것(Fluctuation)을 보여준다.

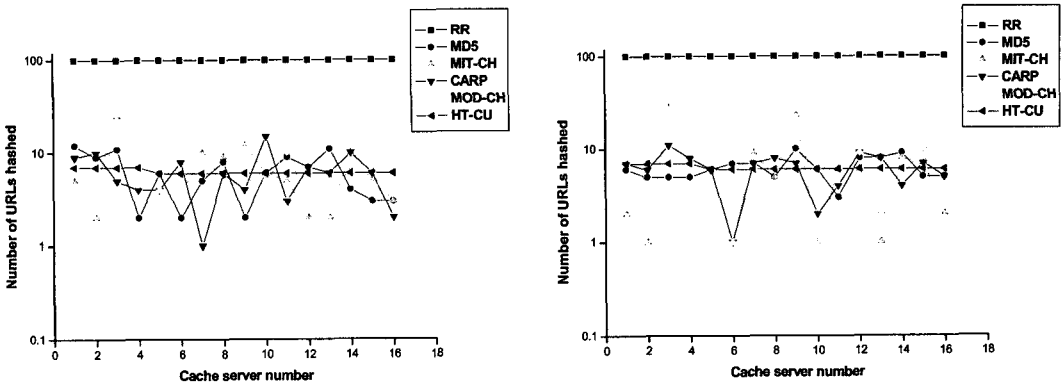
(3) CPU 이용률

그림 6은 CPU 이용률을 보여준다. 이러한 이용률은 리눅스 유틸리티 중의 하나인 vmstat를 이용하여 측정되었다. X 축은 캐시 서버의 수를 나타내고, Y 축은



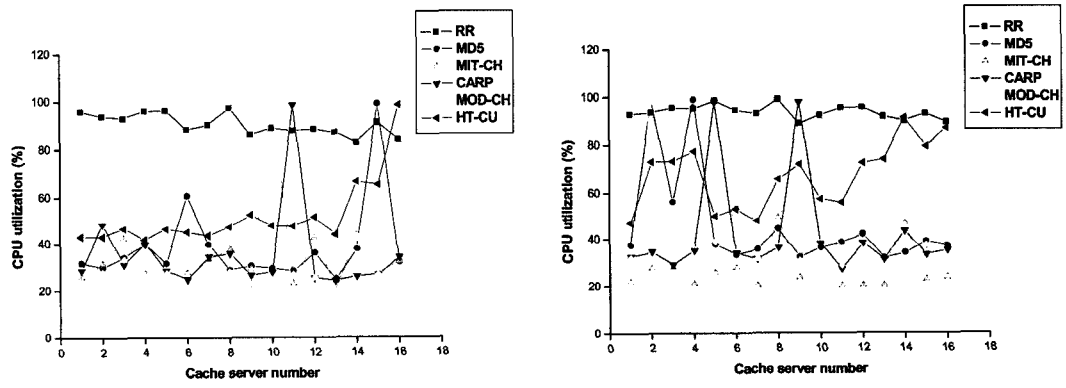
(a) UNC-2001 (b) Berkeley-1998

그림 4 커넥션 수



(a) UNC-2001 (b) Berkeley-1998

그림 5 해시된 URL 수



(a) UNC-2001 (b) Berkeley-1998

그림 6 CPU Utilization

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 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.

(4) 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 얼마다 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 7은 모든 해싱 방법들은 표준 편차들을 나타낸다.

그림 7(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 7(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 분명

하게 제안된 방법(HT-CU)이 라운드 로빈을 제외한 다른 방법들에 비해 우수한 성능을 가짐을 나타낸다.

(5) 확장성

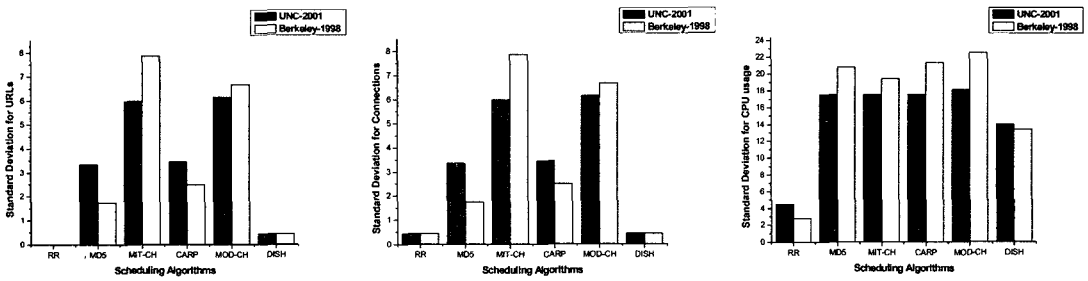
확장성(Scalability)은 초당 처리된 요청의 수로서 측정된다. 그림 8은 UNC-2001 trace를 사용하여 측정된 모든 방법들의 확장성을 보여준다. X 축은 캐시 서버의 개수를 나타내고, Y 축은 초당 처리된 요청의 수를 나타낸다. 그림 8에서 왼쪽 그림은 초당 처리된 요청의 개수를 나타내고, 오른쪽 그림은 왼쪽 그림을 캐시 서버당 처리 속도(Speedup)로 바꾼 것을 나타낸다. 서버의 수가 1개에서 2개로 늘어날 때 초당 처리된 요청의 수는 250에서 350으로 늘어난다. 이는 약 50% 정도 성능이 증가하였음을 나타낸다. 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 로빈이 동일 파일들을 모든 캐시 서버에 저장하기 때문이며, 결과적으로 스토리지(저장 공간) 확장성의 제약을 가지고 온다. 비슷한 확장성이 그림 9의 Berkeley-1998에서도 나타난다. 2개의 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.

4.4 요청 몰림(Hot-Spot)의 경우 (최악의 경우)

표 5와 그림 10은 MD5와 제안된 방법(HT-CU)에서 요청 몰림(Hot-Spot)이 발생한 경우의 실험 결과이다. 요청 몰림은 특정 파일을 지속적으로 요청하는 경우로써, 표와 그림에서 보면 MD5에서 요청이 하나의 파일에 집중되는 경우 성능상의 확장성이 없음을 볼 수 있다. 이는 해쉬의 특성으로 특정 파일은 특정 서버만이 처리할 수 있고, 프록시 서버의 전체 성능은 요청이 집중되는 특정 파일을 처리하는 서버에 종속되게 된다. 이에 반해 제안된 방법은 요청 집중이 발생해도 성능상의 확장성을 보장함을 알 수 있다. 이는 제안된 방법은 요청 집중이 발생한 특정 파일에 한해서 모든 서버들이 동시에 이 파일을 처리할 수 있는 구조로 되어 있기 때문이다. 시간이 지나 더 이상 특정 파일에 대한 요청 집중이 발생하지 않으면 특정 파일은 다시 특정 서버가 처리하게 된다.

6. 결 론

제안된 방법은 클라이언트가 요청한 URL의 해시 값



(a) URL (b) Connection (c) CPU usage

그림 7 표준 편차

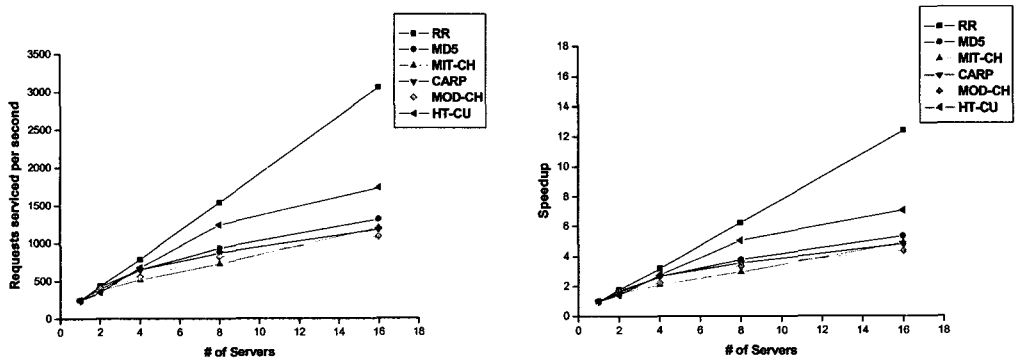


그림 8 UNC-2001

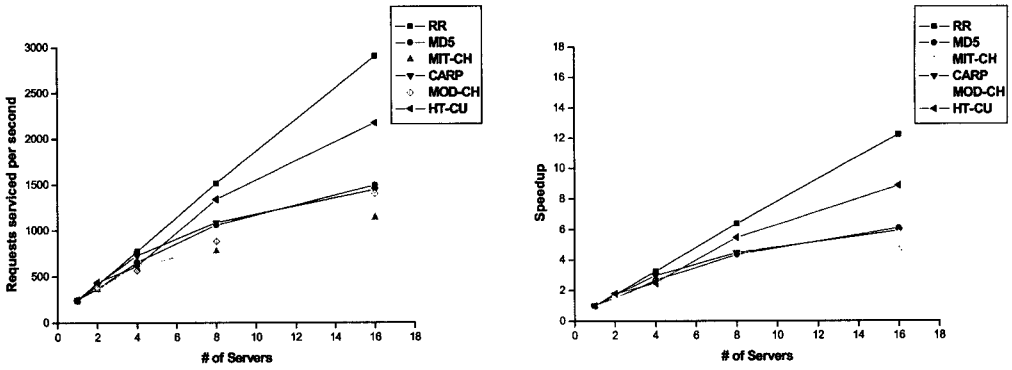


그림 9 Berkeley-1998

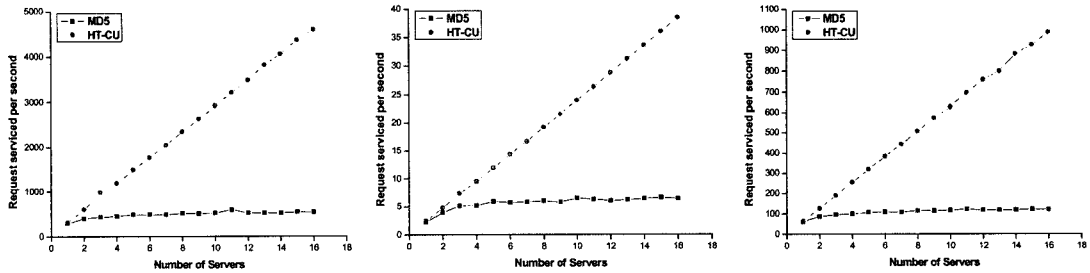
표 5 호스트 개수에 따른 초당 요청수

(a) MD5

호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	299	405	432	451	485	486	486	515	511	521	591	523	520	520	544	538
1 Kbytes	232	316	326	341	365	366	366	386	385	394	433	426	425	420	421	415
10 Kbytes	35	48	54	57	62	62	62	65	65	66	70	67	68	67	70	68
100 Kbytes	2.29	3.96	5.15	5.18	5.88	5.71	5.81	6.00	5.75	6.48	6.27	6.04	6.19	6.40	6.57	6.38
Variation	62	87	96	100	107	109	108	114	114	117	122	118	118	118	121	120

(b) 제안된 방식(HT-CU)

호스트 개수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
300 bytes	317	599	983	1191	1487	1766	2051	2340	2626	2923	3220	3491	3815	4055	4371	4600
1 Kbytes	232	441	662	878	1093	1304	1510	1748	1958	2136	2401	2606	2793	3038	3254	3456
10 Kbytes	36	71	109	143	181	216	253	285	325	360	397	431	467	501	546	567
100 Kbytes	2.47	4.83	7.36	9.53	11.90	14.34	16.70	19.15	21.55	23.96	26.33	28.75	31.18	33.57	36.00	38.42
Variation	65	127	191	256	319	383	443	506	572	626	696	758	798	881	925	987



(a) 300 Bytes (b) 100 Kbytes (c) Variation

그림 10 호스트 개수에 따른 초당 요청수

을 이용하여 캐시 서버를 선택할 때, 캐시 이용률을 이용하는 것이다. 이러한 방법을 이용하게 되면 클라이언트가 요청한 URL 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 몰리는 경우(Hot-Spot)가 발생하면, 요청 몰림이 발생한 해당 URL 요청에 대해서만 라운드 로빈 부하 분산을 사용함으로써 요청 몰림 문제를 해결할 수 있다.

캐시 이용률의 경우 추가적인 계산 시간(매번 요청 URL을 캐시 서버에 할당 할 때 가장 적은 캐시 이용률을 가지는 캐시 서버를 찾아내는 시간)이 필요하다. 기존 방법의 경우 캐시간 협동성(Cooperative Caching)을 위해 해시 기반 부하 분산을 사용할 경우 낮은 확장성

과 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] X. Fu and L. Yang, "Improvement to HOME based Internet caching protocol," IEEE 18th Annual Workshop on Computer Communications, pp. 159-165, 2003.

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

[7] David Karger and al. "Web Caching with consistent hashing," In WWW8 conference, 1999.

[8] Micorsoft Corp., "Cache Array routing protocol and microsoft proxy server 2.0," White Paper, 1999.

[9] F. Baboescu, "Proxy Caching with Hash Functions," Technical Report CS2001-0674, 2001.

[10] 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.

[11] 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.

[12] 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.

[13] Mindcraft, Inc., "WebStone : The Benchmark for Web Server," http://www.mindcraft.com/web-stone.

[14] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," In Proc. ACM SIGMETRICS Conf., Madison, WI, Jul. 1998.

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

[16] 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.

[17] B. A. Mah, "An Empirical Model of HTTP Network Traffic," Proceedings of INFOCOM, pp. 592-600, 1997.

곽 후 근

1996년 호서대학교 전자공학과 졸업(학사). 1998년 숭실대학교 전자공학과 대학원 졸업(석사). 1998년~2006년 숭실대학교 전자공학과 대학원 졸업(박사). 1998년 8월~2000년 7월 (주) 3R 부설 연구소 주임 연구원. 2006년 3월~현재 숭실대학교 전자공학과 대학원(postdoc). 관심분야는 네트워크 컴퓨팅 및 보안

정 규 식

1979년 서울대학교 전자공학과(공학사) 1981년 한국과학기술원 전산학과(이학석사). 1986년 미국 University of Southern California(컴퓨터공학석사). 1990년 미국 University of Southern California(컴퓨터공학박사). 1998년 2월~1999년 2월 미국 IBM Almaden 연구소 방문 연구원. 1990년 9월~현재 숭실대학교 정보통신전자공학부, 교수. 관심분야는 네트워크 컴퓨팅 및 보안