무선 인터넷 프록시 서버 환경에서 자체 학습 기반의 적응적 클러스터링

(A Self-Learning based Adaptive Clustering in a Wireless Internet Proxy Server Environment)

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

요 약 서버들이 서로 다른 데이타를 저장하고 있는 협동성 캐싱을 사용하는 클러스터링 기반의 무선 인터넷 프록시 서버에서는 Hot-Spot 혹은 임의의 입력 요청 패턴이 발생하면 일부 서버만 과부하가 되어 전체적인 성능이 떨어지는 문제점을 가진다. 본 논문에서는 기존 클러스터링이 가지는 Hot-Spot 및 임의의 입력 요청 패턴을 반영하지 못하는 문제점을 해결하기 위해 새로운 자체 학습 기반의 적응적 클러스터링 기법을 제안한다. 제안된 방법에서는 요청을 처리하는 일부 서버들이 과부하가 되면 해당 요청을 다른 서버들로 재 분산한다. 이러한 재 분산은 자체 학습 알고리즘에 의해 수행되고, 다양한 입력 패턴 혹은 서로 다른 성능의 서버들을 가지는 클러스터에도 적용이 가능하다. 제안된 방법들은 16대의 컴퓨터와 부하 분산기를 가지고 클러스터링 환경에서 실험되었고, 실험 결과는 기존 방법들에 비해 54.62% 성능이 향상되었음을 보여준다.

키워드 : 무선 인터넷, 프록시 서버, 자체 학습, 적응적 클러스터링

Abstract A clustering based wireless internet proxy server with cooperative caching has a problem of minimizing overall performance because some servers become overloaded if client request pattern is Hot-Spot or uneven. We propose a self-learning based adaptive clustering scheme to solve the poor performance problems of the existing clustering in case of Hot-Spot or uneven client request pattern. In the proposed scheme, requests are dynamically redistributed to the other servers if some servers supposed to handle the requests become overloaded. This is done by a self-learning based method based dynamic weight adjustment algorithm so that it can be applied to a situation with even various request pattern or a cluster of hosts with different performance. We performed experiments in a clustering environment with 16 PCs and a load balancer. Experimental results show the 54.62% performance improvement of the proposed schemes compared to the existing schemes.

Key words : Wireless Internet, Proxy server, Self-Learning, Adaptive Clustering

1. 서 론

현재 정보화 사회는 인터넷과 무선 이동 통신이라는 커다란 두개의 축에 의해 선도되고 있으며 인터넷과 무선 이동 통신은 서로 조화를 이루면서 급속도로 성장하고 있다. 무선 인터넷이란 언제 어디서나 인터넷에 접속하여 다양한 정보검색과 전자상거래 등을 하는 것으로 기존 유선 인터넷 환경의 공간적 제약을 극복하여 서비스하는 방식이며, 좀 더 넓은 의미로는 무선 랜 등 고정 무선 인터넷 서비스를 포함하여 무선을 통해 인터넷에 접속하는 것을 뜻한다. 급격한 인터넷 사용의 증가와 핸드폰, 노트북, PDA 등 휴대용 단말기의 사용 급증으로 인해 언제든지 인터넷망에 접속하여 정보를 얻고, 통신할 수 있는 무선 인터넷은 생활 속의 필수적인 요소가 되어가고 있다. 무선 인터넷의 사용이 증가하고 있는 현실에서 무선 인터넷의 본질적인 문제 역시 무시할 수 없는 요소로 부각되고 있다. 현재까지 나와 있는 무선 인터넷의 근본적인 문제점은 낮은 대역폭, 빈번하게 연결이 끊기는 현상, 단말기의 낮은 컴퓨팅 파워 및 작은 화면, 단말기의 이동성, 네트워크 프로토콜 및 보안 등

· 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11167-0)지원으로 수행되었음
† 정 회 원 : 숭실대학교 전자공학과 대학원
gobarian@q.ssu.ac.kr
†† 종신회원 : 숭실대학교 정보통신전자공학부 교수
kchung@q.ssu.ac.kr
논문접수 : 2006년 1월 15일
심사완료 : 2006년 4월 7일

으로 나눌 수 있다. 이러한 문제점들에 대한 해결책은 압축(Transcoding, Distillation)[1-3], 캐싱(Caching)[4], 쿠키(Cookie), Mobile IP[5], TCP Migration[6], Wireless TCP[7], 프록시 서버[8,9] 등을 고려해 볼 수 있다. 이에 위의 문제를 캐싱(Caching)과 압축(Transcoding, Distillation)으로 해결하는 방법으로 무선 프록시를 사용한다. 그림 1은 무선 인터넷에서 사용되고 있는 프록시 서버를 나타내고 있으며 이의 기능을 정리하면 압축(Transcoding, Distillation), 캐싱(Caching), 보안(Security), 멀티미디어 전송[10,11], FEC (Forward Error Correction)[12], 핸드오프(Handoff)[13], 프로토콜 변환(Protocol Translation)[14], 오류 복구(Fault Tolerance)[15] 등으로 나눌 수 있다.

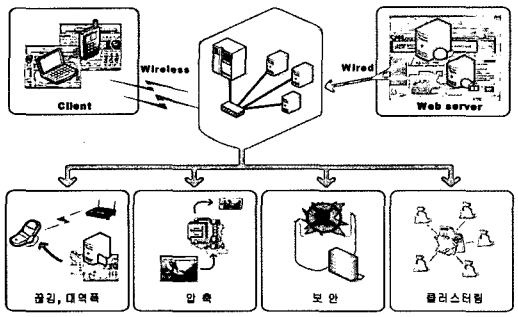


그림 1 무선 인터넷 프록시 서버

본 논문에서는 기존 클러스터링 기반의 무선 인터넷 프록시 서버가 갖는 문제점을 분석하고 이 문제점을 해결하는 새로운 무선 인터넷 프록시 서버를 제안한다. 제안된 자체 학습 기반의 적응적 클러스터링은 기존 클러스터링이 가지는 클러스터 내 호스트 자원의 비효율적 사용 및 적응적 클러스터의 형성 시점이 고정되어 있고 임의의 요청 패턴을 반영하지 못하는 문제점을 해결한다. 본 논문의 구성은 다음과 같다. 제2장에서는 기존 클러스터링의 문제점을 분석하고, 제3장에서는 이를 해결하는 새로운 적응적 클러스터링을 제안한다, 제3장에서는 실험을 통해 제안된 구조의 유효성을 확인하고, 제5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 연구 배경

2.1 클러스터링 : 일반

클러스터링은 독립적인 서버들을 하나로 통합하여 대용량 요청을 처리하기 위해 사용되는 방법[16,17]이다. 그림 2는 일반적인 클러스터링을 나타낸다. 독립적인 각각의 서버가 부하 분산기를 통해 하나의 클러스터로 만들어지고, 사용자는 부하 분산기의 부하 분산에 따라 이 중 하나의 서버로부터 서비스를 받게 된다.

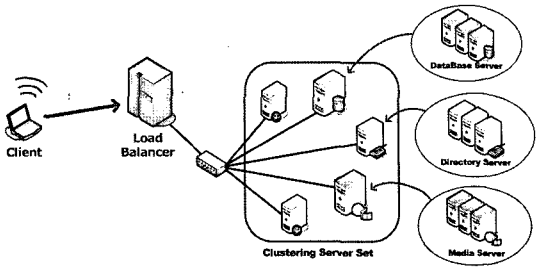


그림 2 일반 클러스터링

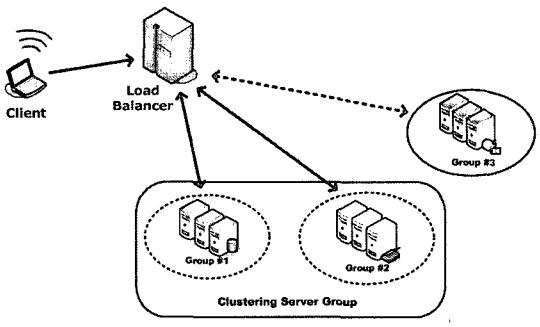


그림 3 그룹 기반 클러스터링

2.2 클러스터링 : 그룹 기반

그룹 기반 클러스터링은 일반적인 클러스터링 기법의 문제점을 해결하기 위해 제안된 방법이다. 그림 3은 그룹 기반 클러스터링의 전체적인 구조를 나타낸다. 그룹 기반 클러스터링은 독립된 서버들을 작은 그룹으로 만들어서 이들을 클러스터링하는 것이다. 기본적으로 요청은 하나의 작업 그룹이 서비스를 수행하고, 세부적으로는 작은 그룹 중에 가장 부하가 작은 부하가 서비스를 하게 된다.

2.3 적응적 클러스터링 : 일반(다중 클러스터)

일반 및 그룹 기반 클러스터링의 문제점을 해결하기 위해 본 논문에서 제안된 방식은 적응적 클러스터링 방식이다. 그림 4는 적응적 클러스터링의 전체적인 구조를 나타낸다. 적응적 클러스터링 방식은 그룹 기반과 마찬가지 방식으로 동작하나 일부 서버로 요청이 몰릴 때

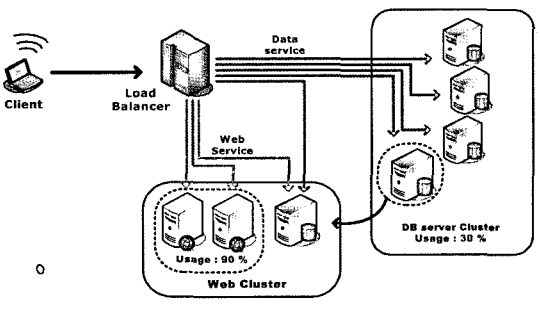


그림 4 적응적 클러스터링

다른 방식으로 동작하게 된다. 일부 서버로 요청이 몰리게 되면 다른 그룹의 서버 중에 부하가 낮은 서버(요청을 적게 처리하는 서버)를 자신의 클러스터로 가지고 와서 요청을 함께 처리한다. 이 서버는 자신의 원래 요청도 처리하고 새로운 클러스터에서의 요청도 동시에 처리하게 된다. 이후에 시간이 지나 일부 서버로의 요청이 더 이상 몰리지 않으면 다시 원래의 클러스터로 돌아가게 된다.

2.4 적응적 클러스터링 : LVS-LBLCR(단일 클러스터)

LBLCR(Locality-based Least-Connection with Replication)은 LVS에서 캐시 클러스터링을 위해 사용되는 부하 분산 방식이다. 같은 수신 IP는 동일 서버 그룹으로 요청이 할당되고, 서버 그룹 내에서는 가장 연결 개수가 적은 서버로 요청이 할당된다. 만약, 서버 그룹 내의 모든 서버가 과부하라면, 서버 그룹을 제외한 다른 서버들 중 가장 연결 개수가 적은 서버를 이 서버 그룹에 포함한다. 일정 시간 동안 서버 그룹에 변화가 없다면(부하를 그대로 유지하고 있다면), 서버 그룹 내에서 가장 부하가 높은 서버를 그룹에서 삭제한다. 그림 5는 이러한 LBLCR의 알고리즘을 나타낸다.

2.5 접근 방식

일반 혹은 그룹 기반 클러스터링은 서버 자원을 비효율적으로 사용하는 문제점을 가진다. 즉, 일부 서버에 요청이 몰리거나 더 이상 요청을 처리할 수 없는 상황에서 여유가 있는 다른 서버들을 사용하지 못한다. 이로 인해 전체적인 서버 클러스터의 성능은 병목인 일부 서버에 종속되게 된다. 적응적 클러스터링 기법은 자신의 클러스터가 요청을 처리할 여력이 없다면 다른 클러스터에서 서버를 빌려오고 요청을 다 처리하면 다시 원래 클러스터로 돌려보내는 방식이다. 그러나 이 방식은 클러스터를 형성하는 시점이 고정되어 있다는 문제점을 가진다. 즉, 상황(입력 패턴 혹은 서버의 부하 상태)이 불특정하게 변해도 클러스터를 형성하는 가중치가 고정되어 있어 전체적으로 일정한 성능을 내지 못하게 된다.

본 논문에서는 일반 혹은 그룹 기반 클러스터링의 문제점을 해결하는 적응적 클러스터링 기법을 제안한다. 또한 적응적 클러스터링의 문제점을 해결하는 방법으로 동적 가중치를 제안한다. 동적 가중치는 클러스터의 형성 시점이 상황에 따라 바뀔 수 있도록 한 것이다. 여러 가지 상황에 대한 가중치를 만들어놓고 일정 시간 동안의 상황을 판단하여 적절한 가중치를 주는 것이다. 그러나 이러한 동적 가중치도 미리 정의해놓은 상황이 발생하지 않는 다면 성능을 보장할 수 없다는 단점을 가진다. 마지막으로 본 논문에서는 동적 가중치가 갖는 문제점을 해결하고자 자체 학습 방법을 제안한다. 제안된 자체 학습 방법은 상황에 따른 가중치 테이블을 미리 설정하지 않고 시간을 통해 자동적으로 테이블이 업데이트 되도록 한 것이다. 즉, 특정 상황에 따른 가중치가 존재하지 않는다면 특정 상황에 대해 여러 가지 가중치를 적용하고 그 결과를 분석하여 최적의 가중치를 찾아낸 다음 이를 테이블에 자동으로 업데이트하는 것이다.

3. 자체 학습 기반의 적응적 클러스터링

3.1 동적 가중치(Dynamic Weights)

적응적 클러스터링은 클러스터의 형성 시점이 고정되어 있다는 단점을 가진다. 이는 상황(입력 요청 패턴 혹은 프록시 서버의 부하 상태)에 따라 서로 다르게 클러스터를 형성해야함에도 불구하고 고정된 클러스터 형성 시점(가중치)으로 인해 적절한 클러스터를 형성하지 못하고 결과적으로 성능이 떨어지게 되는 문제를 가지게 된다. 본 논문에서 제안하는 동적 가중치는 이러한 적응적 클러스터링의 문제를 해결하기 위해 제안된 것으로

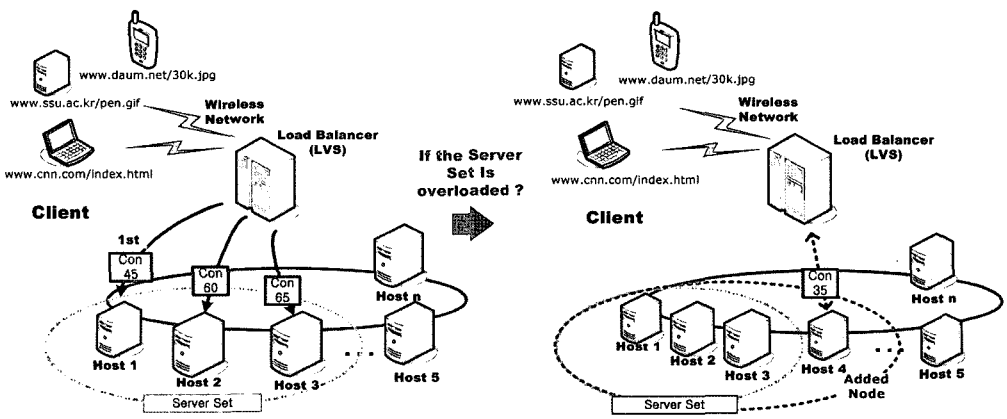


그림 5 LVS-LBLCR

상황에 따라 클러스터의 형성 시점(가중치)을 다르게 구성한다. 이러한 결과는 상황에 따라 최적의 클러스터 구성이 가능함으로 높은 성능 향상을 기대할 수 있다. 적응적 클러스터의 구조 및 동작 과정은 다음과 같다. 동적 가중치의 구조는 그림 6과 같다.

동적 가중치의 동작 과정을 예비 과정과 실제 과정으로 나눌 수 있고 이를 정리하면 다음과 같다.

• 예비 과정

1) 상황(클라이언트의 연결 개수와 호스트의 CPU에 대한 표준 편차)에 대해 다양한 가중치 조합(연결 개수 비율, CPU 비율, 클러스터 형성 임계값)을 적용한다.

2) 실험에 사용된 가중치 조합 중에서 가중 높은 성능(bps)을 갖는 가중치 조합을 상황-가중치 테이블에 정리한다.

3) 다양한 상황에 대해 1)-2)의 실험을 반복하고 이를 상황-가중치 테이블에 업데이트한다.

• 실제 과정

1) 주기적으로(3초) 현재의 상황을 확인한다. 현재의 상황은 연결 개수(클라이언트)와 CPU(호스트)의 표준 편차로 나타낸다.

2) 현재의 상황(연결 개수-CPU의 표준 편차)이 상황-가중치 테이블에 존재하면 해당 가중치를 설정한다. (적응적 클러스터 형성 조건을 위한 가중치)

3.2 자체 학습(Self-Learning)

동적 가중치는 상황-테이블이 사전 실험을 통해 미리 고정되어 있다는 단점을 가진다. 즉, 미리 설정된 상황(입력 요청 패턴 혹은 프록시 서버의 부하 상태)이 아닌 다른 상황이 발생하면 가중치를 설정할 수 없고 이로 인해 성능이 저하됨을 알 수 있다. 본 논문에서 제안하는 자체 학습은 이러한 동적 가중치의 문제를 해결하기 위해 제안된 것으로 상황-테이블이 고정된 것이 아니라 계속 업데이트가 된다는 장점을 가진다. 이는 알 수 없는 상황이 발생하면 프록시 서버의 관리자가 아닌 부하 분산기에서 자체적으로 여러 가지 가중치를 적용하고 결과를 분석해서 최적의 가중치를 얻어낸 후 상황-가중치 테이블을 새로 업데이트하는 것이다. 이 업데이트된 정보는 이후의 동일 상황에 적용하게 됨으로 동적 가중치에 비해 높은 성능 향상을 기대할 수 있으며 상황-가중치 테이블을 형성하기 위해 관리자가 사전 실험을 수행할 필요가 없다. 자체 학습의 구조 및 동작 과정은 다음과 같다.

자체 학습의 구조는 그림 7과 같다. 자체 학습은 전체적으로 3가지 테이블로 구성된다. 첫 번째는 상황 테이블로서 사용자의 입력 패턴을 연결 개수의 표준 편차와 호스트의 CPU Utilization의 표준 편차로 표현한 것이다. 두 번째는 상황-가중치 테이블로서 각 상황에 따라 가중치를 설정하는 것이다. 이때, 가중치는 연결 개수에 대한 비율, CPU Utilization에 대한 비율, 클러스터를 형성하기 위한 임계값(Threshold)으로 구성된다. 마지막으로 세 번째는 자체 학습 테이블로서 상황에 최적인 가중치를 설정하기 위해 학습을 하고 이를 기록하는 테이블이다.

자체 학습의 동작 과정을 정리하면 다음과 같고 이를 의사 코드(pseudo code)의 형태로 나타내면 표 1과 같다.

1) 주기적으로 현재의 상황을 확인한다. 이때, 주기는 1초로 한다. 왜냐하면 1초 이상으로 하면 같은 학습 결과에 학습 시간이 더 길어지고, 1초미만으로 하면 부하 분산기가 1초미만으로 패킷에 대한 통계를 업데이트해

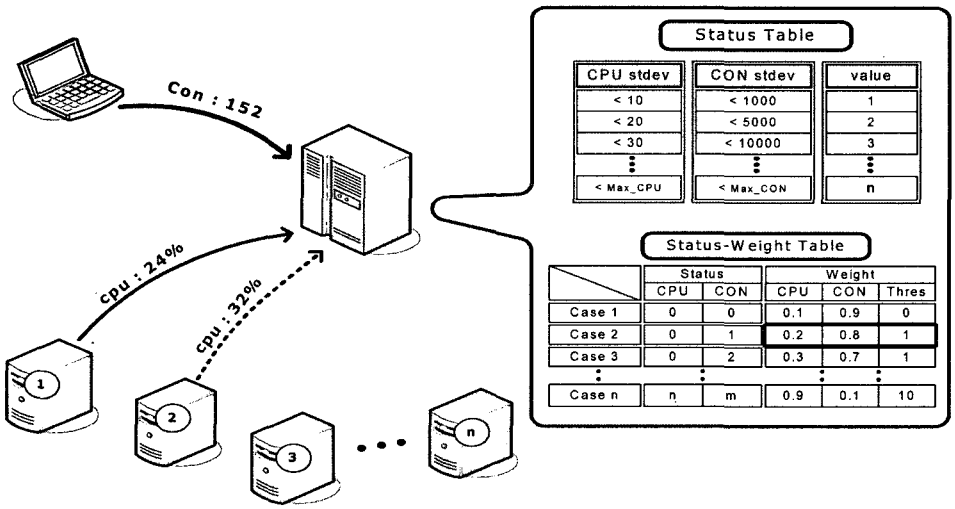


그림 6 동적 가중치

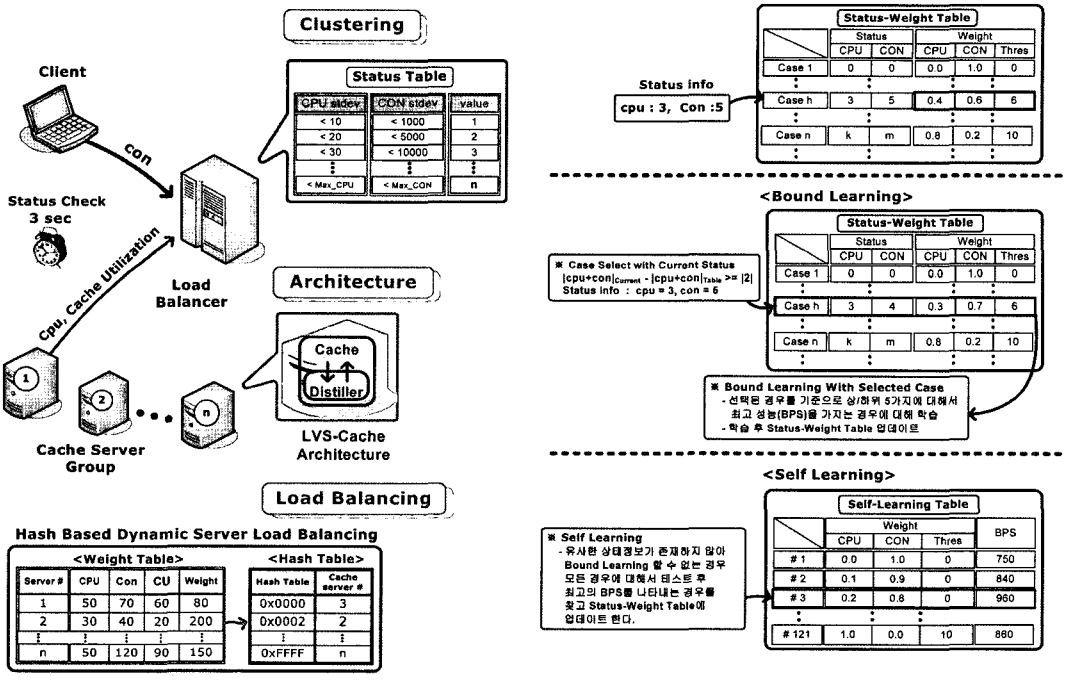


그림 7 자체 학습

표 1 자체 학습의 동작 과정(의사 코드)

```
// 1초 주기로 다음 함수를 호출. CPU=현재의 CPU 표준 편차, CON=현재의 Connection 표준 편차
void self_learning(int CPU, int CON)
{
  // 현재의 상황이 상황-가중치 테이블에 존재하는지 확인
  if (CheckLearningTable(CPU, CON)) // 현재의 상황이 상황-가중치 테이블에 존재한다면
    SetCurrentWeight(CPU, CON); // 현재의 상황에 맞는 가중치를 설정한다.
  else {
    // 이전에 비슷한 상황이 존재했는지 확인
    if (CheckHistory(CPU, CON)) { // 이전에 현재의 상황과 비슷한 상황이 있었다면
      LearningHistory(CPU, CON); // 비슷한 상황의 가중치를 중심으로 학습을 수행한다.
    }
    else { // 이전에 현재의 상황과 비슷한 상황이 없다면
      LearningAll(CPU, CON); // 모든 경우의 가중치를 고려하여 학습을 수행
    }
    UpdateLearningTable(CPU, CON); // 학습이 완료되면 현재의 상황-가중치 테이블을 업데이트
    SetCurrentWeight(CPU, CON); // 학습 후에 만들어진 현재의 상황에 맞는 가중치를 설정한다.
  }
}
```

야 함으로 부하 분산기의 전체 부하를 가중시키게 된다. 현재의 상황은 연결 개수(클라이언트)와 CPU(호스트) Utilization의 표준 편차로 나타내고 이는 모두 일정 범위의 숫자(100×100 or 10×10)를 갖는다. 연결 개수와 CPU Utilization의 표준 편차로 한 이유는 현재의 상황(입력 패턴)을 표현할 수 있는 가장 객관적인 데이타이기 때문이다.

2) 현재의 상황(연결 개수-CPU의 표준 편차)이 상황-가중치 테이블에 존재하면 해당 가중치를 설정한다. (적응적 클러스터 형성 조건을 위한 가중치)

3) 현재의 상황이 상황-가중치 테이블에 존재하지 않는다면 이전의 비슷한 상황이 존재했는지 확인한다.

4) 현재의 상황과 비슷한 상황이 있다면(현재의 상황과 비슷한 상황이 절대차가 2이하이면)

4-1) 비슷한 상황의 가중치를 중심으로 학습을 수행한다.

4-2) 학습이 끝나면 가장 높은 성능(bps)의 가중치로 현재의 상황-가중치 테이블에 업데이트한다.

5) 현재의 상황과 비슷한 상황이 없다면

5-1) 모든 경우의 가중치를 고려하여 학습을 수행한다.

5-2) 학습이 끝나면 가중 높은 성능(bps)의 가중치로 현재의 상황-가중치 테이블을 업데이트한다.

4. 실험 및 토론

4.1 실험 환경

표 2는 실험에 사용된 하드웨어와 소프트웨어를 나타낸다. 무선 인터넷 프록시 서버 클러스터는 PC 16대로 구성된 LVS-Cache 구조를 사용하였으며, 클러스터내의 부하 분산기로는 LVS를 사용하였다. 사용자가 Webstone [18,19]과 Surge[20,21]라는 프로그램을 수행하여 무선 인터넷 프록시 서버에 콘텐츠(HTML과 이미지)를 요청하는 방식으로 실험을 수행하였다. LVS는 DR 방식을 사용하여 서버가 처리한 요청을 직접 사용자에게 전송하도록 하였으며, 서버 내 캐시와 압축기는 오픈 소스인 Squid와 JPEG-6b 라이브러리를 각각 사용하였다. 표 2에서 사용자와 LVS가 서버보다 하드웨어 성능이 좋은 이유는 부하 분산 실험을 할 때 사용자와 LVS에서는 병목이 발생하지 않는 상황에서 무선 인터넷 프록시 서버 클러스터 내 서버들 사이의 부하 분산을 확인하고자 했기 때문이다.

표 3은 실험에 사용된 변수들을 정리한 것이다. 사용자의 요청 시간은 30분에서 60분 사이로 하였다. 사용자의 요청 콘텐츠는 HTML과 웹에서 가장 많은 사용 빈도를 갖는 JPEG 이미지를 사용하였으며 요청 크기는 500 bytes에서 5 Mbytes 사이의 HTML 및 300 bytes에서 100 Kbytes 사이의 이미지를 사용하였다. 실제 웹 트래픽을 모델링하기 위해 Surge라는 프로그램을 사용하여 실제 웹의 분포를 따르는 콘텐츠 및 크기를 가지고 요청을 수행하였다. 또한 UNC(University of North Carolina)[22,23]와 Berkely 대학[24,25]의 실제 웹 trace 데이타를 실험에 사용하였다.

4.2 실험 결과

(1) 학습 행동(Learning Behavior)

그림 8은 4개의 서로 다른 시나리오에 대해 60분 동안 요청을 처리한 결과를 보여준다. 그림 8(a)는 Image (14개 : 300 Bytes - 100 Kbytes)와 HTML(5개 : 500 Bytes - 5 Mbytes)에 대한 실험을 나타내고, 그림 8(b)는 Surge라는 트래픽 발생기를 이용하여 만든 분포에 대한 실험을 나타낸다. 이때, Surge 뒤에 붙은 숫자는 실험에 사용한 파일의 총 수(Image와 HTML의 조합)를 나타낸다. 그림 8(c)는 UNC(University of North Carolina) 대학의 2003-1999년도 실제 웹 trace 데이타에 대한 실험을 나타내고, 그림 8(d)는 Berkeley 대학의 1997년도 실제 웹 trace 데이타에 대한 실험을 나타낸다. 학습 행동은 서로 다른 시나리오(입력 패턴 및 클라이언트 요청)에 대해 기본적으로 동일하다는 것을 알 수 있다. 그림을 보면 처음 몇 분간은 초당 요청을 처리한 수(Throughput)가 계속적으로 변하는 것을 알 수 있는데, 이는 적응적 클러스터를 형성하는 임계값(Threshold)이 계속적으로 변하는 것(학습을 수행하는 것)을 나타낸다. 일정 시간이 지난 후에, 초당 요청을 처리한 수(Throughput)는 일정하게 유지된다(Saturate).

(2) 새로운 패턴에 대한 학습

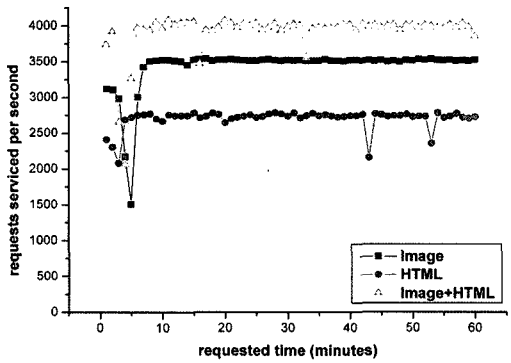
그림 9는 기존의 학습 결과를 새로운 입력 패턴에 대해 어떻게 적용하는지에 대한 결과를 나타낸다. 시스템을 5개의 서로 다른 패턴(Image, HTML, Surge)으로

표 2 실험용 하드웨어 & 소프트웨어

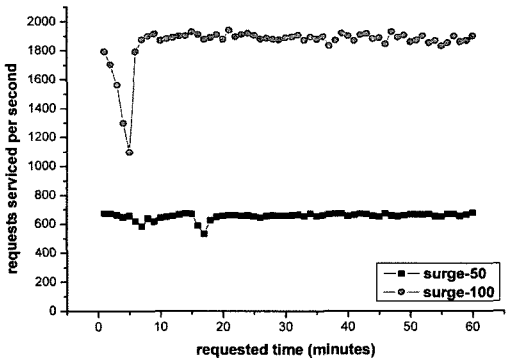
		하드웨어		소프트웨어	개수
		CPU (MHz)	RAM (MB)		
사용자 / 관리자		P-4 2260	256	Webstone, Surge	10 / 1
LVS		P-4 2400	512	DR	1
서버	캐시	P-2 400	256	Squid	16
	압축기			JPEG-6b	
웹 서버		P-4 2260	256	Apache	1

표 3 실험에 사용된 변수

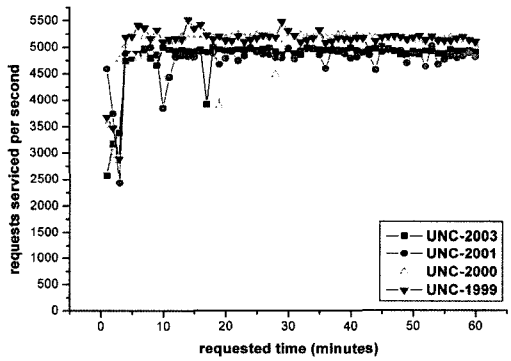
요청 시간	◦30분 - 60분
요청 콘텐츠	◦HTML과 JPEG
요청 크기	◦HTML : 500 bytes, 5 K, 50 K, 500 K, 5 Mbytes ◦JPEG : 300 bytes, 1 K, 10 K, 100 Kbytes, Variation(1, 2, ..., 10 Kbytes) ◦Surge : Surge를 통해 만들어진 HTML과 JPEG의 다양한 크기
요청 방식	◦서로 다른 크기의 HTML과 이미지 파일을 가중치에 따라 요청
사용자 정보	◦압축 정도(이미지 해상도) = 중간(HTML은 압축을 수행하지 않음)
부하 분산 알고리즘	◦RR, LVS-LBLCR, 동적 가중치-자체학습(해쉬 기반 동적 서버 부하 분산)



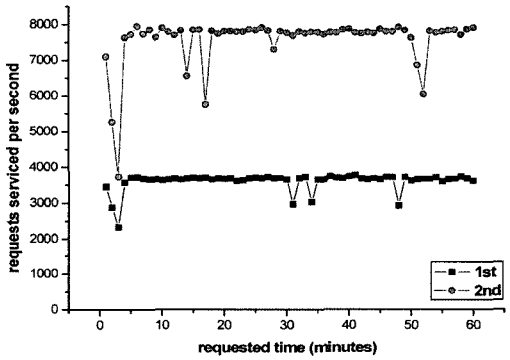
(a) Image, HTML



(b) Surge



(c) UNC



(d) Berkeley

그림 8 학습 행동

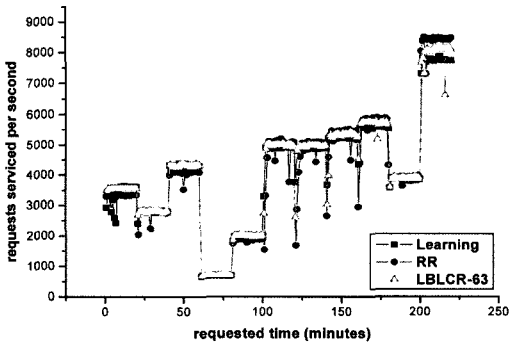


그림 9 새로운 패턴에 대한 학습

학습하여 임계값(Threshold)을 설정한 후, 새로운 패턴(UNC와 Berkeley 대학의 웹 trace 데이타)을 입력하면 기존의 임계값을 토대로 최고의 성능(Throughput)을 유지함을 알 수 있다. 표 4는 20분을 주기로 클라이언트의 개수를 변화시켜가면서 입력한 학습 패턴과 새로운 패턴을 나타낸다. 이 경우는 새로운 입력 패턴에 대해 기존의 학습된 가중치를 그대로 사용하는 경우이다. 이 때, 성능(requests serviced per second)는 그대로 유지할 수 있으나, 적응적 클러스터의 개수를 최소로 유지할 수는 없다. 즉, 새로운 입력 패턴에 대해 새롭게 학습을 수행하는 경우보다 중복 저장되는 데이타의 양이 더 많다(적응적 클러스터의 개수가 더 많다). 새로운 입력 패턴에 대해 새롭게 학습을 수행한다는 것은 적응적 클러스터의 개수를 최적(최소)로 유지한다는 것을 의미한다.

(3) LBLCR(가중치)

그림 10은 가중치에 따른 LBLCR의 성능(초당 처리된 요청 수)을 나타낸다. 가중치가 높을수록 낮은 성능을 보이고, 가중치가 낮을수록 높은 성능을 보임을 알

표 4 학습 패턴과 새로운 패턴 (주기 : 20분)

	학습 패턴					새로운 패턴					
패턴	Image	HTML	Image+HTML	Surge-50	Surge-100	UNC-2003	UNC-2001	UNC-2000	UNC-1999	Berkeley-1	Berkeley-2
사용자	1000	1000	1000	1000	1000	1000	500	1000	500	1000	500

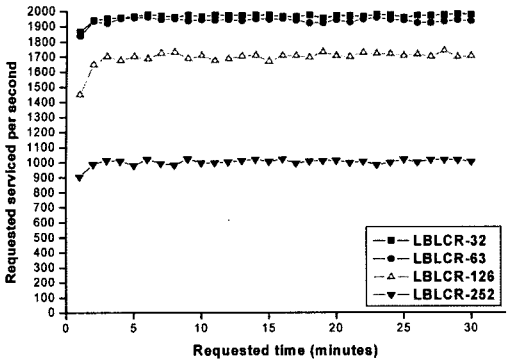


그림 10 가중치에 따른 LBLCR의 성능

수 있다. 가중치가 낮으면 적응적 클러스터를 형성하는 개수가 많아지고, 이에 따라 중복 저장되는 데이타의 양도 많아진다. 반대로, 가중치가 높으면 적응적 클러스터를 형성하는 개수가 줄어들고, 이에 따라 중복 저장되는 데이타의 양은 적어짐을 알 수 있다. 그림 10의 실험 결과는 LBLCR의 가중치에 따라서 성능 및 적응적 클러스터의 형성 개수가 틀려지고, 최적의 가중치(성능이 높으면서 적응적 클러스터의 형성 개수를 최적으로 유지)를 찾기 위해서는 수많은 사전 실험이 필요함을 의미한다. 이에 반해 본 논문에서 제안된 자체 학습 기반의 적응적 클러스터링은 사전 실험이 필요 없이 자체 학습을 통해 최적의 가중치를 자동으로 찾아준다.

(4) 학습 주기

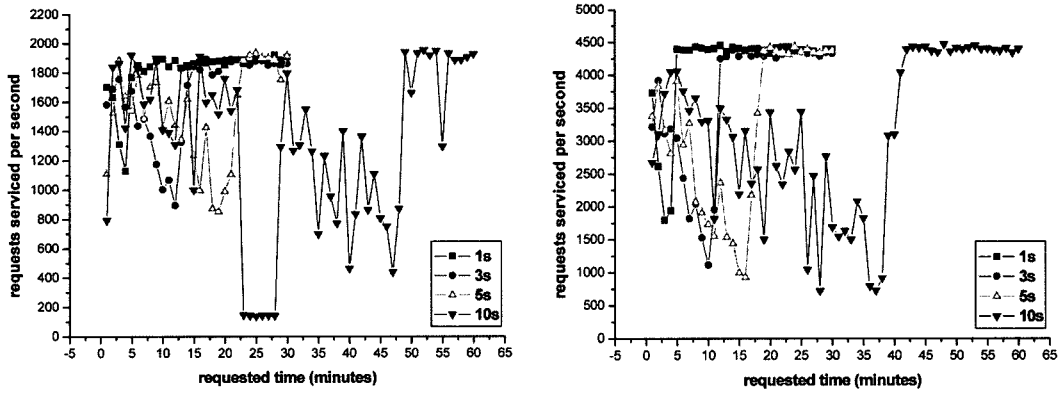
그림 11은 성능(초당 처리된 요청 수)과 학습 주기와의 관계를 나타낸다. 위에서 언급하였듯이, 학습 주기는 얼마나 자주 시스템을 학습을 할 것인가를 나타낸다. 그림에서 보면 학습 주기가 늘어남에 따라 전체 학습 시간이 비례하여 증가하는 것을 알 수 있다. 학습 주기를 1초 미만으로 하려면 부하 분산기의 패킷 관련 Statistics 업데이타 속도를 1초 미만으로 해야한다. 이는 부하 분산기의 부하를 가중시킴으로 업데이트 속도를 1초로 제한하였다.

(5) 학습 상황

그림 12는 성능(초당 처리된 요청 수)과 학습 상황과의 관계를 나타낸다. 학습 상황은 얼마나 많은 상황을 고려하여 학습을 수행할 것인가에 대한 것을 나타낸다. 상황(10x10)에서 처음 10은 호스트 간 커넥션 수의 표준 편차가 가질 수 있는 상황의 수를 나타내고, 뒤의 10은 호스트 간 CPU 이용률의 표준 편차가 가질 수 있는 상황의 수를 나타낸다. 그림에서 보면 학습 상황이 늘어남에 따라 전체 학습 시간이 비례하여 증가하는 것을 알 수 있다. 상황의 개수를 크게 하면(100x100) 전체 학습 시간은 증가하지만, 상황에 대한 민감도(Sensitivity)는 감소한다. 반대로, 상황의 개수를 작게 하면(10×5) 전체 학습 시간은 감소하지만, 상황에 대한 민감도(Sensitivity)는 증가한다.

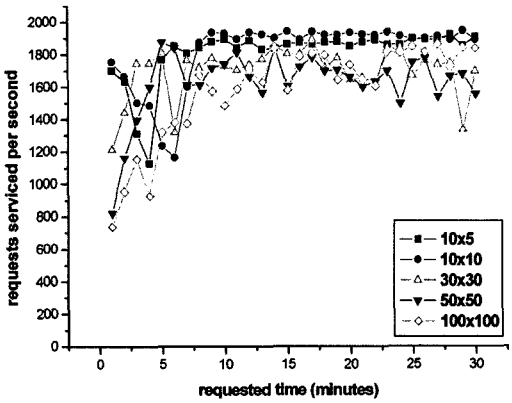
(6) 학습 범위

그림 13은 성능(초당 처리된 요청 수)과 학습 범위와의 관계를 나타낸다. 학습 범위는 하나의 상황에 대한 학습의 양을 얼마로 할 것인가에 대한 것을 나타낸다. 범위(11×11)에서 처음 11은 커넥션의 수와 CPU 이용률이 가질 수 있는 가중치(0.0-1.0)를 나타내고, 뒤의 11은 적응적 클러스터를 형성하는 임계값(Threshold)이 가질 수 있는 가중치(0.0-1.0)를 나타낸다. 그림에서 보면 학습 범위가 늘어남에 따라 전체 학습 시간이 비례하여 증가하는 것을 알 수 있다. 학습의 범위를 크게 하면(121) 전체 학습 시간은 증가하지만, 범위(가중치)에 대한 민감도(Sensitivity)는 감소한다. 반대로, 학습의 범위를 작게 하면(36) 전체 학습 시간은 감소하지만, 범위

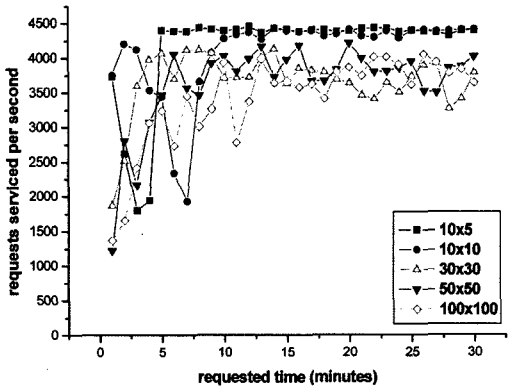


(a) Surge-100 (b) Image+HTML

그림 11 학습 주기

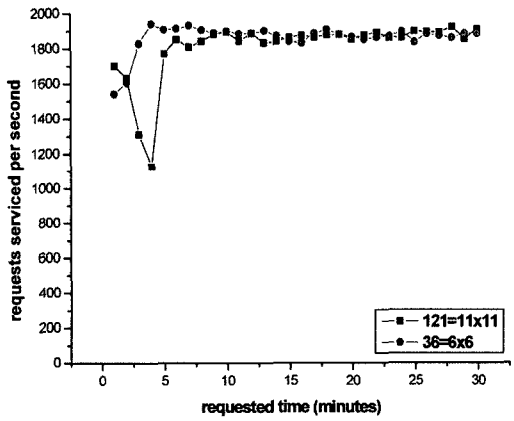


(a) Surge-100

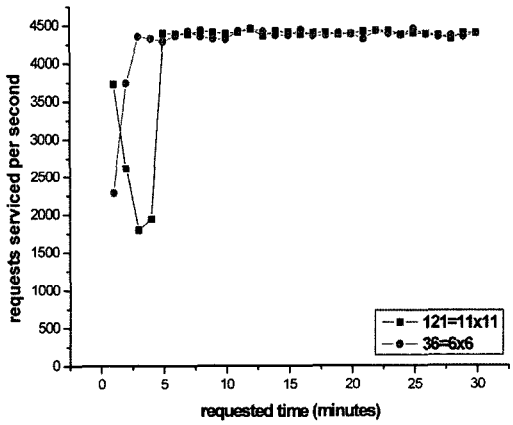


(b) Image+HTML

그림 12 학습 상황



(a) Surge-100



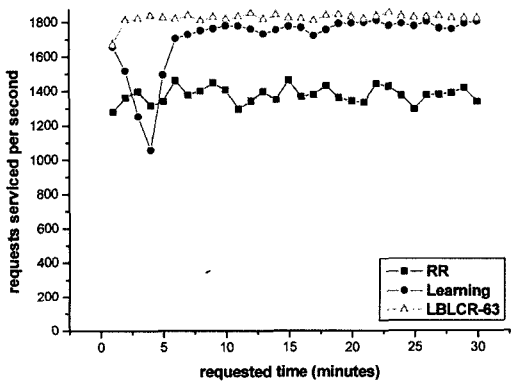
(b) Image+HTML

그림 13 학습 범위

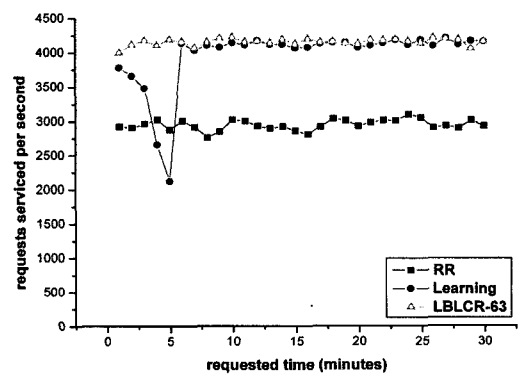
(가중치)에 대한 민감도(Sensitivity)는 증가한다.

(7) 서버 사양

그림 14는 성능(초당 처리된 요청 수)과 서버 사양과의 관계를 나타낸다. 서버의 사양은 16개의 서버 중에 4대에 부하 발생기(Load Generator)를 동작 시켜 서로 틀린 사양을 갖도록 하였다. 부하 발생기는 1-10초 사



(a) Surge-100



(b) Image+HTML

그림 14 서버 사양

이에서 랜덤하게 CPU를 사용한다. RR은 서버의 성능에 무관하게 요청을 분산함으로 전체적인 성능이 다른 알고리즘에 비해 떨어짐을 알 수 있다. 자체 학습은 연결 개수, CPU 부하 정보 및 적응적 클러스터를 바탕으로 학습 후에 최고의 성능을 유지함을 볼 수 있다. 마지막으로, LBLCR은 연결 개수와 적응적 클러스터를 바탕으로 최고의 성능을 유지하나 가중치가 최적으로(여기서는 63) 사전에 설정되어야 한다.

(8) 확장성

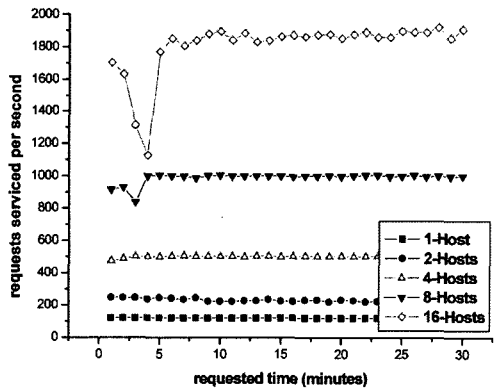
그림 15는 성능(초당 처리된 요청 수)과 서버 개수와의 관계를 나타낸다. 서버 개수가 증가함에 따라 성능이 증가함으로 제안된 자체 학습이 확장성을 가짐을 알 수 있다. 그림에서 보면 서버의 개수에 비례하여 학습 시간이 증가함을 알 수 있는데, 이는 서버의 개수가 많아질수록 학습 상황이 다양하게 발생하여 학습 시간이 증가된 것이다.

(9) 기존 알고리즘과의 비교

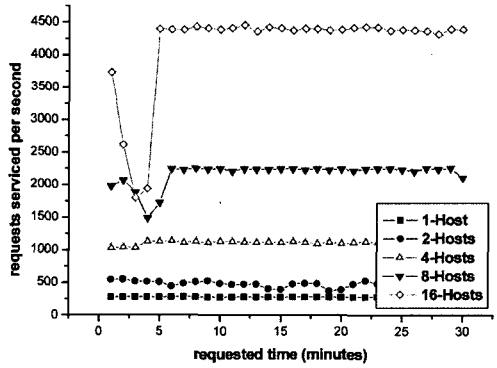
그림 16은 성능(초당 처리된 요청 수) 관점에서 학습, 학습을 수행하지 않은 제안 알고리즘, RR, LBLCR을 비교한 것이다. 자체 학습은 학습을 통해 최고의 성능을 유지하는데 반해, 학습을 수행하지 않는 제안 알고리즘은 사전에 설정된 가중치에 따라 성능에 차이가 나타남을 알 수 있다. LBLCR도 학습을 수행하지 않는 제안 알고리즘과 마찬가지로 사전에 설정된 가중치에 따라 성능에 차이가 나타남을 알 수 있다. RR은 데이타의 중복 저장을 고려하지 않는 알고리즘이지만, 라우팅 테이블을 유지해야해야 하는 부하(요청 패킷의 응답 주소를 테이블로 유지)로 인해 다른 알고리즘에 비해 약간의 성능 저하가 나타남을 볼 수 있다.

(10) 전체 비교

그림 17은 전체적인 비교를 나타낸다. 그림 17(a)은 학습 주기 1초, 학습 상황 10×5, 학습 범위 121=11×11에서 Surge-100에 대한 실험 결과를 나타낸다. 그림 17(b)은 학습 주기 1초, 학습 상황 10x5, 학습 범위 121=11×11에서 Image+HTML에 대한 실험 결과를 나타낸다. 그림 17(c)-(f)는 학습 주기, 학습 상황, 학습

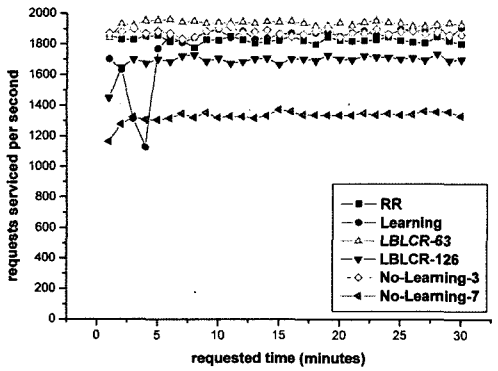


(a) Surge-100

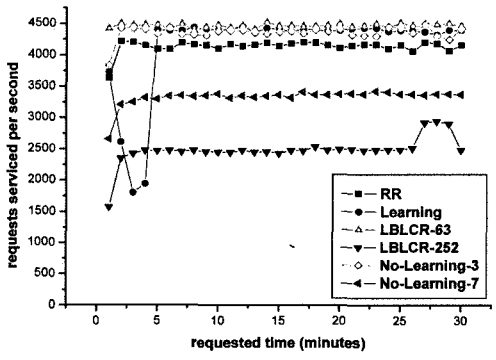


(b) Image+HTML

그림 15 확장성

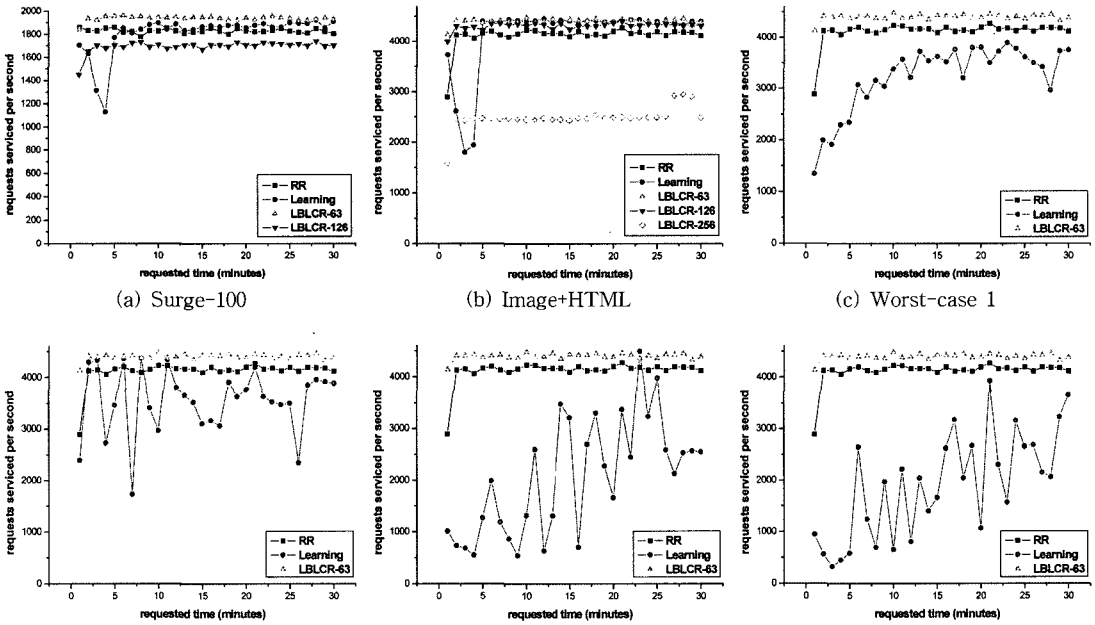


(a) Surge-100



(b) Image+HTML

그림 16 기존 알고리즘과의 비교



(a) Surge-100 (b) Image+HTML (c) Worst-case 1

(d) Worst-case 2 (e) Worst-case 3 (f) Worst-case 4

그림 17 전체 비교

범위에 대한 Worst-case를 고려하여 Image+HTML로 실험한 결과를 나타낸다. 표 5는 각 Worst-case에 대한 학습 주기, 학습 상황, 학습 범위를 나타낸다. 그림 17(a)와 (b)를 보면 RR의 경우 라우팅 테이블을 유지해야 하는 부하로 인해 3개의 알고리즘 중에 가장 낮은 성능을 가짐을 알 수 있다. 자체 학습의 경우 수 분이 지난 후에(학습 후에) 최고의 성능을 유지하는 것을 볼 수 있다. LBLCR도 사전에 가중치만 제대로 설정된다면 최고의 성능을 유지하는 것을 볼 수 있으나, 이는 많은 사전 실험이 필요함을 의미한다. 그림 17(c)-(f)를 보면 학습 주기, 학습 상황, 학습 범위를 Worst-case로 설정하면 학습 시간이 길어지는 것을 볼 수 있다. 이는 최고의 성능에 도달하기 위해 수 시간이 필요함을 의미한다.

(11) 확장성 비교

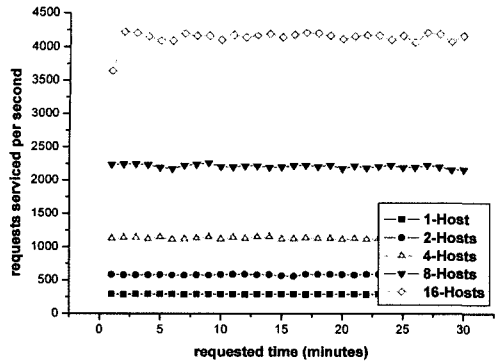
그림 18은 알고리즘 간 확장성 비교를 나타낸다. RR은 일반적인 확장성을 유지하지만, 자체 학습과 LBLCR-63에 비해 약간의 성능 감소를 가진다. 자체 학습은 호스트 개수 별로 학습을 통해 최고의 성능을 유지함으로 확장성을 가짐을 알 수 있다. LBLCR-63은 가중치가 최적으로 설정되어 확장성을 가지지만, LBLCR-252는 가중치가 최적으로 설정되어 있지 않아 확장성을 보장하지 못함을 볼 수 있다.

(12) 가중치 업데이트

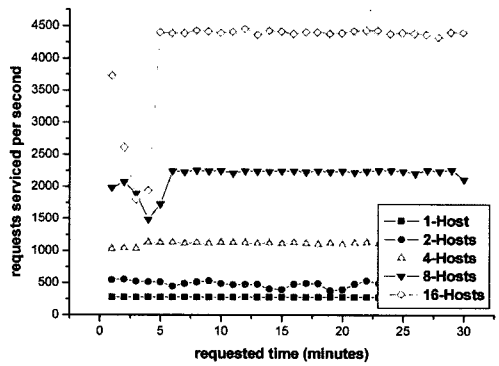
그림 19는 자체 학습을 통해 만들어진 가중치를 업데이트하는 경우와 업데이트하지 않는 경우에 대한 예를 그림으로 표현한 것이다. 그림 19(a)는 가중치를 업데이트하는 경우로써, 매 30분마다 요청 파일을 변경하면서 실험을 수행하였다. 처음 30분은 이미지를 요청하고, 그 다음 30분은 HTML을, 나머지 30분 동안에는 이미지와 HTML을 동시에 요청하였다. 실험 결과를 보면 처음 30분 동안 이미지를 통해 만들어진 가중치를 나머지 60분 동안에도 그대로 사용할 수 있음을 볼 수 있다. 그러나 이는 성능 측면을 고려한 것으로, 최적의 적응적 클러스터를 형성하기 위해서는 그림 19(b)에서처럼 주기적인 재학습을 해야 한다. 그림 19(b)는 그림 19(a)와 같은 실험 조건에서 30분 단위로 주기적인 재학습을 실행한 결과이다. Learning-10×5-11×11-3s는 학습 상황이 10×5, 학습 범위가 11×11, 학습 시간이 3초인 조건

표 5 Worst-case

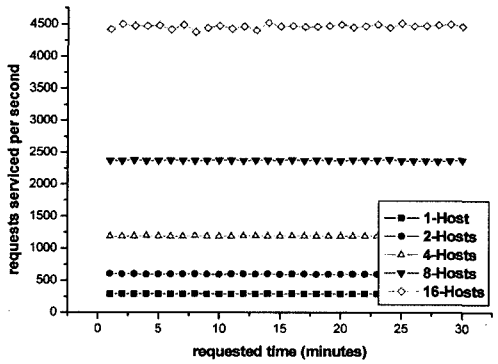
경우	학습 주기	학습 상황	학습 범위
Worst-case 1	1 sec	100×100	36=6×6
Worst-case 2	10 sec	10×5	36=6×6
Worst-case 3	10 sec	100×100	121=11×11
Worst-case 4	10 sec	100×100	36=6×6



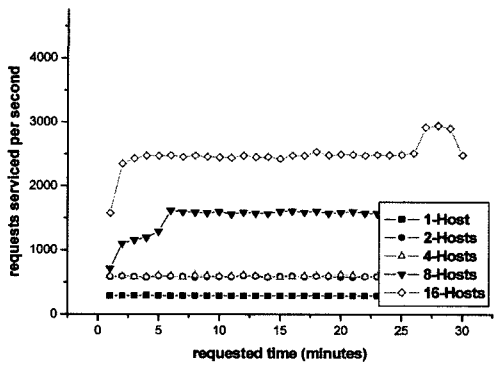
(a) RR



(b) Learning

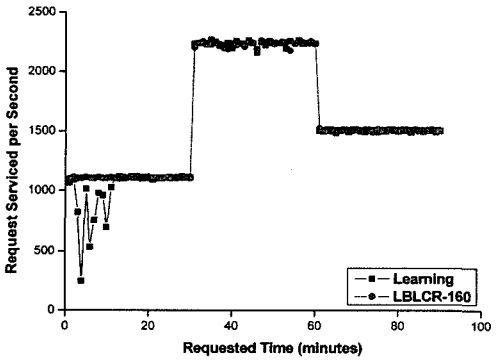


(c) LBLCR-63

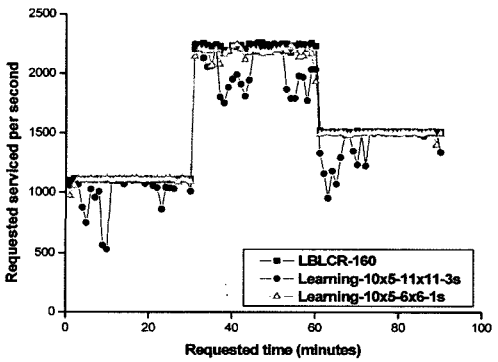


(d) LBLCR-252

그림 18 확장성 비교



(a) 업데이트하는 경우



(b) 업데이트하지 않는 경우

그림 19 가중치 업데이트

에서 재학습을 수행한 결과이고, Learning-10×5-6×6-1s는 학습 상황이 10x5, 학습 범위가 6x6, 학습 시간이 1초인 조건에서 재학습을 수행한 결과이다. 그림에서 보면 학습 범위와 학습 시간을 줄임으로써 전체적인 재학습 시간을 줄일 수 있음을 알 수 있다. 이러한 재학습은 입력 패턴에 상관없이 최고의 성능을 유지하면서 최적의 적응적 클러스터 형성 개수를 가짐을 의미한다.

4.3 토론

제안된 적응적 클러스터링의 단점은 클러스터를 형성하는 시점이 고정되어 있다는 것이다. 이는 상황(입력 패턴 혹은 서버의 부하 상태)이 변해도 일정한 가중치를 사용하여 클러스터링 함으로 전체적으로 동일한 성능을 가질 수 없음을 의미한다. 동적 가중치는 상황에 따른 클러스터 형성 시점인 가중치가 바뀌지 않는 문제를 해결하기 위해 사용된다. 상황에 따라 가중치가 바뀜으로 상황이 바뀌어도 동일한 성능을 유지한다는 장점

을 가진다. 이에 반해 동적 가중치는 미리 정해놓은 상황이 아니면 적응적 클러스터링과 마찬가지로 동일 성능을 보장하지 못하는 단점을 가진다. 즉, 미리 정의해 놓은 상황-가중치 테이블에 존재하지 않는 상황이 발생하면 클러스터를 적절하게 형성해 주지 못한다. 자체 학습은 불특정 상황을 해결하기 위해 제안된 것으로써 상황이 바뀌고 이 상황이 미리 정의해놓지 않은 상황이라도 최적의 성능을 보장한다. 즉, 불특정 상황이 발생하면 자체 학습을 통해 상황-가중치 테이블을 업데이트하고 이후의 동일한 불특정 상황에서 이를 이용하게 된다. 반면 자체 학습은 불특정 상황이 발생한 초기에 자체 학습 시간이 필요하다는 단점을 가진다.

5. 결 론

본 논문에서는 기존의 클러스터링 기반의 무선 인터넷 프록시 서버가 가지는 문제점을 분석하고 이를 해결하는 새로운 무선 인터넷 프록시 서버를 제안하였다. 제안된 클러스터링 기반의 무선 인터넷 프록시 서버는 클러스터내의 호스트 자원이 비효율적 사용 및 적응적 클러스터의 형성 시점이 고정되어 있고 임의의 입력 요청 패턴을 반영하지 못하는 클러스터링시의 문제점을 해결하기 위해 자체 학습 기반의 적응적 클러스터링을 제안하였다. 실험을 통해 제안된 무선 인터넷 프록시 서버가 기존 무선 인터넷 프록시 서버의 문제점을 해결하면서 성능 향상(54.62%)에 기여했음을 확인하였다. 제안된 클러스터링 기반의 무선 인터넷 프록시 서버를 실제로 적용하게 되면 다음과 같은 효과를 얻을 수 있다. 클러스터 내 호스트 자원을 효율적으로 사용함으로써 전체 클러스터가 확장성을 가지고 자체 학습을 통해 클러스터의 형성 시점을 입력 패턴에 따라 단시간 내에 자동적으로 바꾸어 줌으로써 클러스터 시스템 운영(Operation)에 들어가는 시간을 최소로 유지시켜준다.

참 고 문 헌

[1] A. Savant, N. Memon, and T. Suel, "On the scalability of an image transcoding proxy server," International Conference on Image Processing, to appear, 2003.

[2] Keqiu Li and Hong Shen, "Coordinated enroute multimedia object caching in transcoding proxies for tree networks" ACM Transactions on Multimedia Computing, Communications, and Applications, Vol. 1, Issue 3, 2005.

[3] Keqiu Li, K. Tajima, and Hong Shen, "Cache Replacement for Transcoding Proxy Caching," Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, pp. 500-507, 2005.

[4] 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.

[5] C. Perkins, "Mobile IP," Communications Magazine, IEEE, Vol. 35, No. 5, pp. 84-99, 1997.

[6] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory TCP: connection migration for service continuity in the Internet," Proceedings of 22nd International Conference on Distributed Computing System, IEEE, pp. 469-470, 2002.

[7] N. Eshak and M. Baba, "Design a new transport protocol (wireless TCP) to support mobility for mobile ad hoc networks," NCTT 2003 Proceedings of 4th National Conference on Telecommunication Technology, IEEE, pp. 144-147, 2003.

[8] S. Ross, J. Hill, M. Chen, A. Joseph, D. Culler, and E. Brewer, "A security architecture for the post-PC world," U. C. Berkeley Technical Report, to appear.

[9] B. Zenel and D. Duchamp, "A general purpose proxy filtering mechanism applied to the mobile environment," Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking, pp. 248-259, 1997.

[10] Z. Sahinoglu and P. Orlik, "Power efficient transmission of layered video through wireless proxy servers," IEEE Electronics Letters, Vol. 39, Issue 8, pp. 698-699, 2003.

[11] P. Yong and J. Modestino, "Interactive video coding and transmission over wired-to-wireless IP networks using an edge proxy," IEEE Internation Conference on Acoustics, Speech, and Signal Processing, Vol. 4, pp. 281-284, 2003.

[12] P. Mckinley, T. Chiping, and A. Mani, "A study of adaptive forward error correction for wireless collaborative computing," IEEE Transactions on Parallel and Distributed Systems, Vol. 13, Issue 9, pp. 936-947, 2002.

[13] Z. Jiang, K. Leung, B. Kim, and P. Henry, "Seamless mobility management based on proxy server," Wireless Communications and Networking Conference, IEEE, Vol. 2, pp. 563-568, 2002.

[14] J. Rendon, F. Casadevall, and J. Carrasco, "Wireless TCP proposals with proxy servers in the GPRS network," The 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Vol. 3, pp. 1156-1160, 2002.

[15] B. Yao and W. Fuchs, "Recovery proxy for wireless applications," Proceedings of 12th International Symposium on Software Reliability Engineering, IEEE, pp. 112-119, 2001.

[16] E. Carrera and R. Bianchini, "PRESS: a clustered server based on user-level communication," IEEE Transactions on Parallel and Distributed Systems, Vol. 16, Issue 5, pp. 385-395, 2005.

[17] I. Elhanany and M. Kahane, "Heterogeneous bursty traffic dispersion over multiple server clusters," IEEE Communications Letters, Vol. 9, Issue 3, pp. 261-263, 2005.
[18] Mindcraft, Inc., "WebStone : The Benchmark for Web Server," http://www.mindcraft.com/web- stone.
[19] 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.
[20] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," In Proc. ACM SIGMETRICS Conf., Madison, WI, Jul. 1998.
[21] 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.
[22] 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.
[23] 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.
[24] B. A. Mah, "An Empirical Model of HTTP Network Traffic," Proceedings of INFOCOM, pp. 592-600, 1997.
[25] 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

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. 2003년 6월~현재 (주) 펌킨넷 코리아 선임 연구원. 관심분야는 네트워크 컴퓨팅 및 보안

정 규 식

1979년 서울대학교 전자공학과(공학사) 1981년 한국과학기술원 전산학과(이학석사). 1981년~1984년 금성사(현 LG전자) 중앙연구소 선임연구원. 1986년 미국 University of Southern California(컴퓨터공학석사). 1990년 미국 University of Southern California(컴퓨터공학박사). 1990년~1993년 (주) 코리아써키드 기술자문. 1993년 12월~1994년 3월 미국 IBM Wastson 연구소 방문 연구원. 1998년 2월~1999년 2월 미국 IBM Almaden 연구소 방문 연구원. 1990년 9월~현재 숭실대학교 정보통신전자공학부, 교수. 관심분야는 네트워크 컴퓨팅 및 보안