컨시스턴스 해싱을 이용한 분산 웹 캐싱 시스템의 성능 향상을 위한 Hot Spot 예측 방법

정회원 정 성 칠*, 정 길 도**

Hot Spot Prediction Method for Improving the Performance of Consistent Hashing Shared Web Caching System

Sung-Chil Jung*, Kil-To Chong** *Regular Members*

요 약

월드 와이드 웹에서 사용자의 요청에 대한 웹 서버의 신속하고 정확한 서비스는 매우 중요하다. 그러나 최근에는 인터넷 사용자의 급속한 증가로 인하여 신속한 서비스가 어려운 상황이다. 이러한 문제를 해결하기 위해 분산 웹 캐싱이 사용되고 있다. 분산 웹 캐싱의 성능은 히트 율에 의해서 결정되며 히트 율은 메모리 사이즈, 서버의 처리속도, 로드밸런싱 등의 영향을 받는다. 기존의 로드밸런싱은 주로 현재의 시스템 상태를 기준으로 실시하고 있으나, 미래에 발생할 서비스를 예측하는 방법을 이용하여 로드밸런싱을 실시함으로써 히트 율의 향상이 가능하다. 본 연구에서는 웹 서버 또는 프록시에 요청하는 가장 빈번한 hot spot을 예측하고, 예측된 hot spot을 공유하고 있는 프록시에 미리 패치함으로써 프록시의 활용을 개선하는 Hot Spot Prediction Method (HSPM)을 제안한다. 시뮬레이션을 통하여 제안한 방법이 기존의 consistent hashing보다 로드밸런싱, 히트 율 측면에서 우수함을 확인하였다.

Key Words : WWW, Shared Web Caching System, Consistent Hashing, Hot Spot, Load balancing

ABSTRACT

The fast and precise service for the users request is the most important in the World Wide Web. However, the fast service is difficult due to the rapid increase of the Internet users recently. The Shared Web Caching (SWC) is one of the methods solving this problem. The performance of SWC is highly depend on the hit rate and the hit rate is effected by the memory size, processing speed of the server, load balancing and so on. The conventional load balancing is usually based on the state history of system, but the prediction of the state of the system can be used for the load balancing that will further improve the hit rate. In this study, a Hot Spot Prediction Method (HSPM) has been suggested to improve the throughputs of the proxy. The predicted hot spots, which is the item most frequently requested, should be predicted beforehand. The result show that the suggested method is better than the consistent hashing in the point of the load balancing and the hit rate.

Ⅰ. 서 론

월드 와이드 웹이 정보 제공의 중요한 매체가 되어짐에 따라 효율적이고 신뢰성 있는 정보를 전달하는 기술이 필요하게 되었다. 그러나 현재의 인터넷을 이용한 데이터 전송은 네트워크상의 예측할 수 없는 지연과 전송의 실패가 빈번하게 발생하는 경향이 있다. 이러한 지연과 실패는 네트워크상의

*전북대학교 네트웍 시스템제어연구실(jsc@chonbuk.ac.kr) **전북대학교 전자정보공학부 부교수(kitchong@chonbuk.ac.kr)
논문번호 : 030402-0915, 접수일자 : 2003년 09월 15일

swamped 서버와 병목현상으로 인하여 발생한다. 서버가 최대 처리용량에 비하여 많은 양의 요청을 받았을 때 정보제공이 매우 느리거나 실패하게 되는데, 이를 swamped 상태라 한다. 병목(bottleneck)은 네트워크에서 데이터 전송이 갑자기 느려지거나 정체하는 현상으로서 사용자의 폭주에 의해서 발생한다. 이것은 일반적인 웹 문서 요청 패턴이 Zipf's law[1,2]를 따른다는 연구결과에 따라 인기 있는 몇 개의 object들이 전체 요청량의 90%이상을 차지하면서 서비스 지연과 전송의 실패를 야기하는 것이다. 사용자들에 의해 자주 요청되는 object들을 hot spot이라 부르며, hot spot의 영향을 줄이고자 많은 연구가 진행 되고 있다.

분산 웹 캐싱 시스템[3,4,5]를 사용하여 사용자의 요구를 만족시키고 효과적인 네트워크의 자원을 활용하기 위하여, 프록시를 여러 네트워크상에 분산시킴으로써 로드를 분산하도록 한다. 분산 웹 캐싱 시스템은 사용자가 네트워크상의 가장 가까운 프록시에 접속하거나, RTT(round trip time)[6]가 작은 프록시에 접속하여 응답시간을 빠르게 한다. 그리고 해쉬 라우팅[7,8]을 사용하여 여러 프록시에 웹 object들의 중복 복사를 방지하여 프록시의 효율을 최대화시킨다. 또한 캐쉬 정책으로서 분산 디렉토리 환경에서 서비스 처리속도를 향상시키기 위하여 원격지의 object에 대한 질의와 결과를 사용자의 캐시에 저장하는 방법[9]과, 근접한 지역에 위치하면서 특성이 유사한 소규모 집단을 연결하는 웹 캐시 구성으로 동일한 캐쉬 용량으로도 히트 율이 높고 웹 문서 사용자들이 요구하는 URL이 급격히 바뀌어도 히트 율 저하를 예방할 수 있는 방법[10]이 연구되어 졌다. DNS를 이용하여 object의 요청 상태를 감시하는 동안 일정 임계값을 넘으면 모든 프록시에 로드밸런싱을 시켜주는 방법[7]과 hot spot의 참조와, 공유 프록시간의 로드 불균형을 경감하기 위해 동료 공유 프록시에 hot spot인 object를 복사한다[11]. 이때 복사의 양을 cpu 사용도에 따라 제어함으로 동료 프록시의 효율을 극대화시킨다.

이러한 연구는 hot spot의 영향을 해결하기 위하여 hot spot의 정보를 DNS나 과거의 캐쉬된 정보를 이용하였지만, 본 연구에서 제한한 Hot Spot Prediction Method (HSPM)는 분산 웹 캐싱 시스템에서 사용자들이 요청할 hot spot을, 오리지널 서버나 프록시의 access logs 파일을 분석하여, 시간대별로 예측하고 미리 공유된 여러 프록시에 분산 패치 하는 방법이다. 따라서, 기존의 방법에서는 현재나 과거의 정보를 이용하여 로드 분산을 수행하는 반면, 본 연구에서는 미래의 정보를 이용하여 여러 공유된 프록시에 미리 hot spot을 패치 하여 로드밸런싱을 수행한다.

본 연구에서는 consistent hashing[7]을 이용하여 분산 웹 캐싱 시스템을 구성하였으며, 예측된 hot spot을 이용하여 공유 프록시에 분산 패치하므로 히트 율을 향상시키고, 로드 불균형을 해결 하고자 한다. 로드밸런싱을 위해 proxyhelper 프로그램을 이용하여 각 프록시의 상태를 동적으로 수집하고, 제안한 HSPM으로 로드밸런싱을 실시하였다. 로드밸런싱을 실시할 때 요청 받은 hot spot을 어느 프록시에 할당시킬 것인지, 얼마나 많은 프록시에 분산 패치할지는 각 프록시의 로드상태에 따라서 결정한다.

실험결과를 확인하기 위하여 시뮬레이터를 PERL 프로그램 언어로 작성하였고, 각 모듈은 Virtual, Web, Dns와 Proxy로 구성되어 있다. 모듈 Virtual에서는 전체적인 시뮬레이션을 담당하고 이곳에서 모든 데이터가 처리된다. 모듈 Web에서는 object를 요청하는 기능을 수행한다. 또한 모듈 Dns에서는 object를 쿼리하여 적당한 프록시에 할당하여 주며, 모듈 Proxy에서는 할당받은 object를 저장한다.

본 연구를 통해 제안된 HSPM은 hot spot을 프록시에 미리 패치 함으로써 해당하는 hot spot의 요청에 대한 서비스 시간을 단축하며, hot spot을 공유된 프록시에 균등하게 분산함으로써 프록시 효율을 증가함은 물론 사용자의 요청에 대한 히트 율이 상승 할 수 있다.

본 논문의 전체적인 구성을 살펴보면 Ⅱ장에서는 웹 캐싱과 consisting hashing에 대해서 살펴보았고, Ⅲ장에서는 사용된 데이터의 생성과 특성에 대해서 논의하며, Ⅳ장에서는 제안한 HSPM에 대하여 다루었다. Ⅴ장에서는 시뮬레이터의 구성과 시뮬레이션 방법에 대해서 살펴보았고, Ⅵ장에서는 실험 결과를 평가하고 분석한다. 마지막 Ⅶ장에서 결론에 대하여 기술하였다.

Ⅱ. 웹 캐싱과 Consistent Hashing

1. 웹 캐싱

웹 캐싱은 일반적으로 단일 캐싱과 분산 웹 캐싱로 구분한다. 단일 캐싱은 다수의 사용자들이 하나의 프록시를 사용하여 서비스를 제공받는 간단한

캐싱 방법이다. 요청한 정보를 프라이머리 프록시에서 제공받지 못할 경우, 요청 받은 프록시는 오리지널 웹 서버에게 자료를 요청한다. 먼저 사용자에게 자료를 제공하고 또한 제공한 자료를 프록시에 저장하여 향후 요청될 것을 대비한다. 그러나 단일 캐싱은 하나의 프록시를 이용하여 모든 요청을 처리하기 때문에 프록시의 부하, 네트워크 대역폭, 히트율, 지연시간 등의 한계를 갖는다.

이에 반하여, 분산 웹 캐싱은 여러 프록시에 요청을 분산하여 처리하는 방식이다. 사용자는 자신을 담당하는 프라이머리 프록시에게 정보를 요청한다. 만약 요청자료를 프라이머리 프록시가 보유하는 경우에는 자료를 제공하고, 그렇지 않을 경우에는 ICP(Internet Cache Protocol)[6] 또는 해쉬 라우팅을 이용하여 다른 공유 프록시에게 자료를 요청하고 자료를 보유한 프록시가 사용자에게 자료를 제공하게 된다. 요청한 자료가 공유한 프록시에 존재하지 않을 경우에는 단일 캐싱과 동일한 동작을 수행한다. 이런 방법으로 요청에 대해서 서비스를 제공하기 때문에 단일 캐싱보다 프록시의 부하, 지연시간, 대역폭 사용량이 감소하게 되며, 히트 율이 높아진다.

2. Consistent Hashing

분산 웹 캐싱 시스템에서 각 프록시에 적절한 요청량을 분산시켜주는 로드밸런싱은 중요한 문제이다. 본 연구에서는 분산 웹 캐싱 방법 중의 하나인 consistent hashing에 기초하여 로드밸런싱을 수행하였다. consistent hashing은 해쉬 함수를 사용하여 여러 종류의 object를 공유된 프록시에 할당한다.

표준 해쉬 함수를 사용하여 사용자가 요청한 object를 여러 프록시에 할당 할 때 프록시 ID는 각 object를 해쉬 함수를 통해 나온 값에 대하여 모듈러 연산자(MOD)를 사용하여 구한다. 만약 프록시가 추가되거나 작동이 멈춘다면, 모든 object에 대해서 MOD 연산을 다시 실시하여 프록시를 재 할당해야 한다. 결과적으로 프록시가 추가되거나 감소할 경우에 항상 새로운 할당을 수행하기 때문에 기존의 저장된 모든 캐시를 더 이상 사용할 수 없게 되며, 이로 인하여 히트 율이 감소하고 프록시의 성능이 현격히 저하된다. 그러나 consistent hashing 방법은 프록시가 추가되거나 작동이 중단될 경우에도 모든 object에 대하여 프록시 할당을 다시 수행하지 않고 일부 object에 대해서만 재 할당을 수행한다. 그러므로, 기존에 저장된 대부분의 캐시를 사용할 수 있으며 시스템의 변동이 히트 율에 큰 영향을 미치지 않는다. 그림 1은 consistent hashing의 동작 원리를 나타낸 그림이다. 이 방법은 모든 object의 해쉬 값을 모듈러 연산자를 사용하여 구간 [0 1]사이의 단위 원 상에 랜덤 하게 할당한다. 마찬가지로 분산 웹 캐싱 시스템의 각 프록시들 또한 해쉬 함수를 사용하여 단위 원 상에 랜덤 하게 할당한다. 각 object가 할당된 프록시를 찾는 방법은 해당 단위 원 상의 object의 모듈러 값에서 시계방향으로 이동하여 프록시의 모듈러 값을 만나면, 그 프록시가 object를 담당한다.

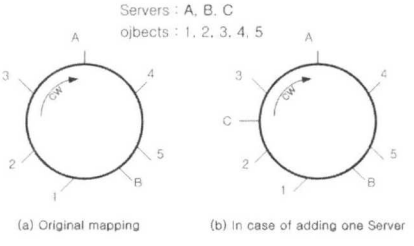

그림 1. Consistent Hashing 알고리즘의 동작 원리

만약 공유 시스템에서 세 개의 프록시 A, B, C가 설치되어 있고, 다섯 종류의 object 1, 2, 3, 4, 5가 캐시 된다고 가정하자. 그림 1.(a)는 object 1, 2, 3이 프록시 A에 4, 5는 프록시 B가 담당하는 경우의 단위 원 상에 해쉬 하여 모듈러 연산을 취한 결과를 나타낸다. 그리고 그림 1.(b)는 프록시 C가 추가된 경우를 나타낸 것이다. 해쉬 함수를 통과하여 모듈러 연산을 수행한 object를 프록시에 할당한 결과를 보면, object 1, 2는 프록시 C에 object 3은 A에 object 4, 5는 B에 할당한다. 따라서 consistent hashing에서는 모든 object를 재 할당하지 않고 해당되는 object를 새로운 프록시에 할당하게 된다.

Ⅲ. 실험 데이터

본 연구에서는 NASA 홈페이지의 access logs 파일을 분석하여 실험데이터를 생성하였다. Hot spot을 예측하는 과정에 필요한 access logs 파일의 분석 방법, 실험데이터 생성 및 특성에 대해서 살펴보자

1. NASA 홈페이지 access logs 파일분석

access logs 파일은 웹 서버를 통해 이루어지는

모든 작업을 기록한 파일이다. 사용자가 웹 서버에 접속하면 모든 작업에 관련된 자료가 access logs 파일에 저장된다. 즉, 특정 웹 페이지를 사용자가 요구하면, 웹 서버는 해당 웹 페이지와 관련된 모든 object들에 접근하게 되며, 사용자가 요청한 웹 페이지뿐만 아니라 그 페이지와 관련된 이미지 파일, 링크 데이터 등 요청된 object를 처리하는 모든 과정의 정보가 access logs 파일에 저장된다. 그러므로 access logs 파일 분석을 통하여 요청 데이터 수, 접속시간, 방문자수 및 방문 경로 등의 데이터를 얻을 수 있으므로 사용자가 사이트에 방문한 목적을 알 수 있는 중요한 자료가 된다. 또한 웹 서버의 작업 요청과 성공 여부에 대한 것뿐만 아니라 실패 했을 경우, 그 해결책에 대한 정보가 저장되어 있다. 본 연구에서 사용한 NASA 홈페이지의 access logs 파일에 다음과 같은 형식으로 내용이 기록되어 있다.

in24.inetnebr.com - - [01/Aug/1995:00:00:01 -0400] "GET /shuttle/missions/sts-68/news/sts-68-mcc-05.txt HTTP/1.0" 200 1839

위에 기록된 내용은 7가지 정보로 구성되어 있으며, 순서대로 호스트, 검증, 사용자 승인, 시간 표시, HTTP 요청 부분, 상태코드, 그리고 전송량을 나타낸다. 본 연구에서는 위의 7가지의 정보 중에서 요청 시간과 HTTP 요청부분의 2가지 요소를 추출하여 hot spot을 예측하는데 필요한 데이터로 사용하였다.

2. 실험 데이터 생성

NASA 홈페이지의 access logs 파일을 분석하여 추출한 데이터의 요청 패턴은 Zipf's law을 따르는 것을 확인할 수 있었다. 그러나 access logs 파일에서 얻은 데이터는 매 순간마다 요청된 모든 데이터가 저장되어 있으므로, 본 연구에서 제안한 시스템의 응용에 용이하지 않아 적절한 시간단위로 데이터를 재구성하였다. 본 실험에서는 hot spot의 예측을 30분단위로 설정하였다. 이때 각 구간의 요청량을 확인해본 결과 hot spot의 요청량이 현저히 적어, hot spot이 급격히 증가하는 상황에 대비하여 시스템의 성능을 향상시키려는 본 연구의 목적을 위한 데이터로 적절하지 못하다는 결론을 내렸다. 그래서 보스톤 대학에서 개발한 로드 생성기인 Surge 프로그램[12]를 이용하여 실험 데이터를 본 연구에 적절하도록 생성하였다. 데이터를 생성할 때 임의적으로 생성하지 않고, NSAS 홈페이지의 access logs 파일에서 시간의 구간동안 요청된 양과 동일한 요청량이 되도록 생성하였다. 다만, hot spot의 양을 증대시켰다. 즉, 각 구간마다 전체 object의 종류를 500으로 설정하고, hot spot에 대한 요청량을 100으로 설정하여, Zipf's law에 따라 각 object의 요청량을 결정하였다. 그러나 Surge를 이용하여 생성한 데이터를 분석한 결과 모든 구간에서 오직 특정한 하나의 object가 hot spot으로 요청이 되었다. 이 때문에, hot spot이 된 object가 동일하게 발생하여 hot spot들 사이의 요청 순위 변동에 대한 예측 알고리즘을 응용할 수 없게 되었다. 그래서 이러한 문제점을 해결하고자 원본 데이터를 6시간 이동시키고, object의 종류를 다르게 하여 위와 같은 방법으로 데이터를 생성하여 다른 종류의 object가 hot spot되도록 데이터를 구성하였다. 그래서 원래 데이터와 6시간 이동시켜 얻은 데이터를 합산하여 실험데이터로 사용하였다. 이와 같은 과정을 통해서 생성한 실험데이터는 object의 종류는 1000개이며, 전체 사이즈는 18MB이다.

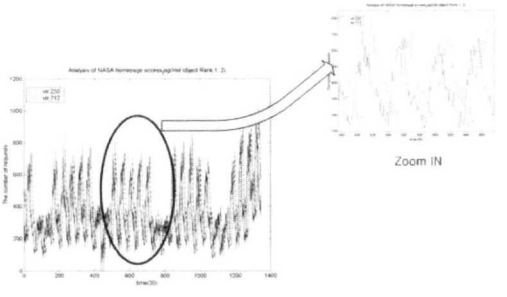

그림2. 시간에 따른 Hot Spot의 요청량 추이

그림 2는 최종적으로 생성한 실험 데이터의 시간에 따른 hot spot의 요청량 추이를 나타낸 것이다. 그림을 통해서 두 종류의 hot spot이 시간이 변함에 따라 요청에 대한 순위 변동이 있는 것을 알 수 있다.

Ⅳ. Hot Spot 예측 방법 (HSPM)

본 논문에서 제안한 방법은 consistent hashing을 적용한 라운드 로빈 DNS와 유사하며 분산 공유 시스템의 성능향상을 위해 hot spot을 미리 예측하고, 예측된 hot spot을 각 프록시에 라운드 로빈 방법을

이용하여 로드밸런싱을 적용하였다. 본 연구에서 적용한 글로벌 호스팅 시스템(GHS)[13]은 hot spot에 의하여 특정한 어느 프록시에 많은 요청이 발생하는 문제점이 있다. 그러나 본 연구는 그림 3에서 hot spot을 미리 예측하여 6의 과정을 먼저 수행하여 공유 프록시에 분산 패치하는 방법이다. hot spot의 예측은 어느 특정한 프록시의 access logs 파일에서 요청되었던 빈도수가 높은 몇 개의 object에 대해서 분석하여 이루어진다. 분석을 통해 입수한 자료에서 기존에 요청되었던 hot spot의 양을 이용하여 향후 요청될 양을 예측한다. 예측하는 방법으로는 신경회로망을 이용한 시계열 방식을 사용한다. 그리고 예측된 요청량에 따라서 hot spot을 분산 가능한 프록시들에 저 부하 프록시부터 우선적으로 패치 시킨다. 이러한 방식으로 hot spot을 사용자에게 지리적으로 가까운 프록시에 미리 패치 시키면 갑자기 폭증할 요청량에 대비할 수 있으며 Content Delivery Network (CDN)의 성능을 증가시킬 수 있게 된다. 본 연구에서는 예측 모델을 사용하지 않고 NASA 홈페이지의 access logs 파일의 실제 데이터를 직접 이용하여 예측 데이터를 대신하였다. 제안한 방식의 성능을 실험하기 위하여 그림 3과 같이 글로벌 호스팅 시스템을 적용하였다.

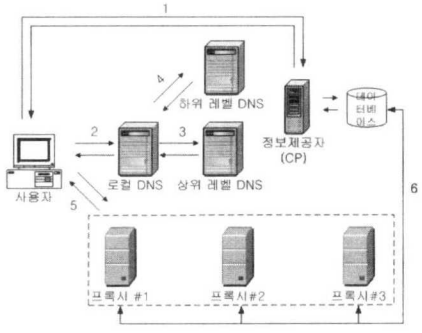

그림 3. 글로벌 호스팅 시스템

그림 3을 통하여 응답 수행 과정을 구체적으로 기술하면 다음과 같다. 먼저 사용자는 CP의 IP를 미리 알고 있다고 가정한다. 사용자가 웹 페이지를 요청하면 정보 제공자(CP)에서는 HTML에 포함되어 있는 텍스트를 제공하고, 그 외의 문서에 포함되어 있는 그림이나 미디어 파일 등의 embedded objects는 사용자 근처에 있는 프록시에 의해서 제공받는다. 사용자가 1의 과정에서 CP에 HTML 페이지를 요청하면 CP는 일반적인 페이지를 보내주는 대신 embedded object URLs이 포함된 페이지를 보내준다. 이때 사용자는 스크립트를 이용하여 받은 embedded object URLs의 이름을 해쉬하여 새로운 embedded object URLs을 만들고, 새로운 embedded object URLs이 저장되어 있는 프록시의 IP를 찾기 위해 2의 과정에서 로컬 DNS에 접속한다. 3의 과정에서 상위 레벨 DNS는 네트워크에서 사용자가 어디에 위치하는가를 결정하고, 도메인 위임 기능을 이용하여 사용자 근처에 있는 하위 레벨 DNS로 연결 시켜 준다. 4의 과정에서 하위 레벨 DNS는 consistent hashing 방법에 의하여 embedded object가 맵핑되어 있는 프록시 IP 주소를 제공하며, 5의 과정에서 제공된 프록시 IP 주소로 접속하여 embedded object를 요청한다. 요청한 자료가 존재하면 프록시는 embedded object를 제공하고, 그렇지 않을 경우에는 6의 과정을 통하여 프록시가 CP 서버에게 다시 embedded object를 요청하여 자신의 캐시에 저장과 동시에 사용자에게 제공한다.

V. 시뮬레이션

본 연구에서 제안한 HSPM의 성능을 검증하기 위하여 시뮬레이터를 구성하였으며, 기존의 방법인 consistent hashing 방법과 본 연구에서 제안한 HSPM 방법을 시뮬레이션을 통해서 비교하였다.

1. 시뮬레이터 구성

시뮬레이터는 수정이 용이하고, 관리의 편의성을 위해서 모듈화 방식으로 작성했으며, PERL언어를 사용하여 프로그래밍 하였다. 시뮬레이터는 GHS를 모델링 하였으며, 전체적인 구성은 그림 4와 같다.

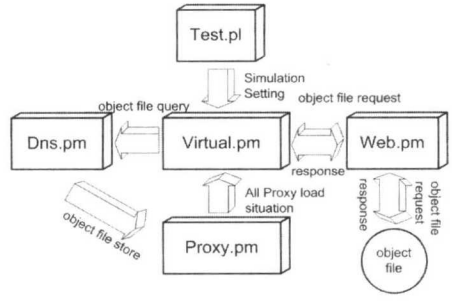

그림 4. 시뮬레이터의 전체적인 데이터 흐름도

시뮬레이션을 실시할 때 관련된 모듈의 흐름을 살펴보면 다음과 같다. 모든 프로그램의 모듈에 관

련된 옵션들은 Test.pl에서 설정한다. 이때 사용할 수 있는 옵션들은 프록시 수, object 하나를 처리하는 속도, 캐시 사이즈, 예측 오차율 등이다. 각 모듈은 Virtual, Web, Dns와 Proxy으로 구성되어 있다. 모듈 Virtual에서는 전체적인 시뮬레이션을 담당하고 이곳에서 모든 데이터가 처리된다. 모듈 Web에서는 object를 요청하는 기능을 수행한다. 모듈 Dns에서는 object를 쿼리하여 적당한 프록시를 할당하여 주며, 모듈 Proxy에서는 할당받은 object를 저장한다.

5.2 시뮬레이션 방법

웹 서버의 성능 실험을 위하여 프록시 수, 큐의 메모리 사이즈, 처리시간을 다양하게 설정하여 시뮬레이션을 실시하였다. 프록시의 저장공간과 관련된 메모리의 캐시 정책[14]은 LRU(Least Recently Used)방식을 선택하였고, 프록시 서버의 서비스 정책은 FIFO(first input first output)형식에 따른다. 프록시의 처리속도는 프로그램상에서 bound라는 옵션으로 제어하는데, 이것은 한 개의 object를 처리하는데 사용되는 시간을 의미한다. 그리고 매개변수 bound는 서비스의 성공과 실패를 결정하는 변수로 사용되기도 한다. 또한 프로그램에서 매개변수 A가 사용되는데 이 변수는 한 개의 데이터를 처리하는데 걸리는 시간을 나타낸다. 즉 A는 일정 시간동안에 요청된 object 전체의 수를 소요된 시간으로 나누어 얻을 수 있다. 그래서 bound < A일 때는 프록시 서버의 성능이 요청한 서비스를 충분히 처리할 수 있기 때문에 miss가 발생하지 않는다. 그러나 bound > A일 때는 시간 지연으로 인하여 요청이 처리되지 않기 때문에 miss가 발생한다. 또한 프록시의 큐에 저장되어 있는 데이터를 요청하면 히트, 그렇지 않을 경우에는 miss 처리된다. 그리고 히트율은 서비스가 제공된 히트수를 서비스를 요청한 object의 수로 나눈 값의 백분율로 표시된다.

시뮬레이션 처리순서를 살펴보면, 먼저 Test.pl에서 여러 성능 옵션들을 설정하고, 모듈 Web에서 object를 입력받아 MD5[15]를 이용하여 해싱 하고 consistent hashing 방법에 의해 선택된 프록시에 해쉬된 object를 저장한다. 그리고 공유 프록시에 요청에 대한 로드 분산을 수행하기 위하여 각 프록시의 로드상태를 30분 단위로 파악한다.

본 실험에서는 consistent hashing을 적용하기 위해 1000개의 가상 캐쉬를 생성하였다. 그리고 전체 프록시의 수와, 각 프록시에 대한 카피 프록시 수를 1000개의 가상 캐쉬에 할당하기 위해 MD5를 이용하였다. 카피 프록시를 사용한 이유는 각 프록시에서 처리하는 요청량의 비율을 균등하게 하기 위해서다.

모듈 Virtual의 서부루틴 run은 메인 함수의 역할을 담당하며, 모든 object가 이곳에서 처리된다. 먼저 모듈 Dns에서 object를 MD5에 의해 해싱한 다음, 각 프록시가 담당하고 있는 1000개의 가상 캐쉬에 할당한다. 그러면 해쉬된 object의 프록시 ID가 결정된다. 요청에 대한 로드밸런싱을 실시하기 위하여 현재까지 분산시킨 hot spot의 위치를 저장해야 한다. 왜냐하면 향후 hot spot이 요청되었을 때 부하가 적게 걸리는 순서로 공유된 프록시에 할당시킬 수 있기 때문이다. 그리고 hot spot이 아닌 object가 요청될 경우는 MD5에 의해서 해쉬된 가상 캐쉬 값으로 할당시킨다. 모듈 Proxy는 시뮬레이션을 통해서 처리된 object를 각 프록시에 저장하는 기능을 수행한다. 또한 일정 시간마다 각 프록시의 부하 상태를 측정하여 로드밸런싱 과정에서 프록시의 로드 상태를 사용할 수 있도록 자료를 제공한다.

Ⅵ. 성능평가

1. 로드밸런싱 성능평가

먼저 성능 평가 방법으로 consistent hashing 기반 한 DNS를 이용한 라운드 로빈 방법과 본 논문에서 제안한 hot spot 예측을 이용한 로드밸런싱 방법에 대해서 각 프록시의 부하 불균형을 비교해 보았다.

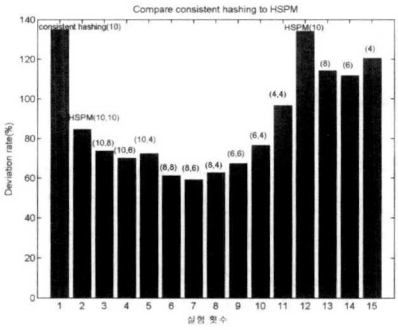

그림 5. 분산 캐시에 부과되는 요청의 편차도

그림 5는 공유된 여러 프록시에 hot spot을 분산했을 때, 프록시들에 요청된 부하의 표준편차를 나

타낸 것이다. 시뮬레이션을 실시할 때 매개변수로는 분산시키는 hot spot의 수와 프리패치(prefetch) 되는 프록시 수를 고려하였다. hot spot의 분산을 실시할 때, 가장 요청수가 큰 hot spot을 로드가 가장 적게 부과된 다수의 프록시에 분산 프리패치 하는 방법과, 한 개 이상의 hot spot을 다수의 프록시에 분산 프리패치 하는 방법 등 다양한 경우에 대한 실험을 실시하였다. 그림에서 x-축은 실험의 다양한 방법 즉 특정한 분산 방법을 이용한 시뮬레이션을 실시한 횟수를 의미하고, y-축은 프록시에 부과되는 작업량의 표준 편차를 나타낸다. 즉, 15 가지 경우의 다양한 분산 방법에 대해서 시뮬레이션을 실시하였다. 그림 5에서 consistent hashing(10)은 consistent hashing 방법에서 가장 빈도수가 높은 hot spot을 10개의 프록시에 분산하는 것을 의미하며, HSPM(10,10)은 HSPM을 이용하여 빈도수가 가장 높은 두개의 hot spot을 각각 10개의 프록시에 분산하는 것을 의미한다. 즉 가장 빈도수가 높은 hot spot을 10개의 프록시에 두 번째로 빈도수가 높은 hot spot을 또한 10개의 프록시에 분산한 것이다. 본 논문에서 빈도수가 가장 높은 hot spot을 HS(1), 두 번째로 빈도수가 높은 hot spot를 HS(2)로 표시한다. 시뮬레이션 결과 HS(1)인 hot spot 1개를 분산시켰을 때 보다, HS(2)까지, 즉 2개까지 로드 분산을 실시했을 때 로드 분산이 원활히 이루어지는 것을 알 수 있었다. 또한 두개의 hot spot을 분산시킬 경우에도, HS(1)은 8개의 프록시에, HS(2)는 6개의 프록시에 로드를 분산 시켰을 때 가장 좋은 성능을 보여주었다. 따라서 여러 개의 hot spot을 분산시킬 때, 분산시키는 프록시의 수와 hot spot의 수를 조합하는 방법에 따라 전체 시스템의 성능에 많은 영향을 미치는 것을 알 수 있었다.

2. 히트율 성능평가

앞 절에서는 기존의 consistent hashing 방법과 HSPM 방법의 로드밸런싱에 대한 성능을 비교해 보았다. 웹 캐싱 시스템의 성능에서 매우 중요한 히트 율에 대한 성능을 알아보기 위하여 다양한 서버의 처리속도와 캐시 사이즈의 조합으로 한 개의 hot spot과 두개의 hot spot의 경우에 대한 실험을 실시하였다.

1) 한 개의 hot spot 로드밸런싱

그림 6, 7은 consistent hashing 방법을 이용하여 10개의 프록시에 분산시킨 것과 제안한 HSPM 방법을 시뮬레이션을 통해 얻은 다양한 결과 중에서 히트 율이 우수한 2가지 경우에 대해서만 표시한 것이다.

HSPM 방법은 consistent hashing 방법과 같이 hot spot을 처음에는 10개의 프록시에 패치 하였으며, 두개씩 감소시켜 최저 4개까지 설정하는 실험을 실시하였다. 이때 또한 큐 사이즈가 시스템에 미치는 영향을 살펴보기 위하여, 큐 사이즈를 10에서 60까지 변화시키며 실험을 실시하였다. 여기에서 큐 사이즈는 한 프록시에 저장할 수 있는 object의 수를 의미한다. 그래프에서 x-축은 큐 사이즈를 y-축은 히트 율을 나타낸다. 그림 6은 처리속도가 빠른 경우에 대한 시뮬레이션 결과이며, 이 경우 큐 사이즈가 10인 경우에 4개의 프록시에 분산하면 가장 높은 히트 율을 보이며, 큐 사이즈가 증가함에 따라 분산 프록시의 수를 6으로 설정했을 때 가장 높은 히트 율을 보여주었다. 또한 그림 7은 처리속도가

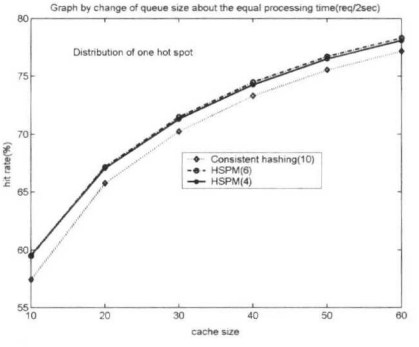

그림 6. 처리속도(req/2sec)일 때 히트 율 비교

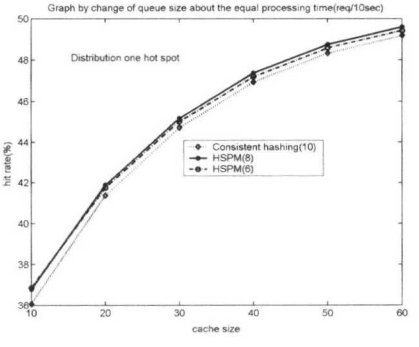

그림 7. 처리속도(req/10sec)일 때 히트 율 비교

느린 경우에 대한 실험이며, 큐 사이즈가 10일 경우 프록시의 수를 6으로 설정할 때 히트 율이 가장 높았고, 큐 사이즈가 증가함에 따라 분산 프록시 수를 8개로 설정했을 때 가장 높은 히트 율을 보여주었다. 그림 8에서는 큐 사이즈를 30으로 설정한 후 각 프록시에서 한 개의 object를 처리하는 속도가 감소할 때 히트 율의 변화를 나타낸 그래프이다. 그림에서 보는 바와 같이 제안한 HSPM이 기존의 consistent hashing 방법 보다 좋은 히트 율을 보여주고 있다.

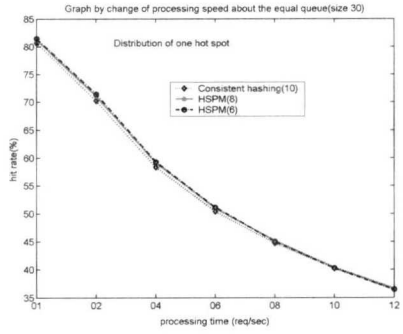

그림 8. 동일한 큐 사이즈(30)의 히트 율 비교

처리속도가 빠를 경우에는 hot spot을 6개의 프록시에만 분산시키고, 속도가 느려지는 경우에는 8개의 프록시에 hot spot을 분산 처리한 부분이 좋은 히트 율을 보였다. 이와 같이 처리속도가 빠를 경우에는 분산 웹 캐싱 시스템의 전체 작업량을 고려하여 hot spot을 분산시키는 프록시의 수를 적게 하고, 처리속도가 느려지는 경우에는 많은 프록시에 분산시켜 사용자의 요청에 대하여 좋은 히트 율을 얻을 수 있다.

2) 두 개의 hot spot 로드밸런싱

그림 9, 10은 두 개의 hot spot을 공유 프록시에 분산하여 시뮬레이션을 실시한 결과에 대한 그래프로 x-축은 캐시 사이즈, y-축은 히트 율을 나타낸다.

그림 9는 프록시 서버가 서비스를 처리하는 시간이 빠른 경우의 실험결과를 나타낸 것이다. 이때 두 개의 hot spot을 공유 프록시에 부하를 분산하는 방법이 한 개의 hot spot을 부하 분산하여 로드밸런싱을 수행한 방법보다 히트 율이 높게 나타난다. 또한 기존의 consistent hashing 방법과 제안한 HSPM 방법을 비교했을 때, HSPM 방법에서 평균적으로 히

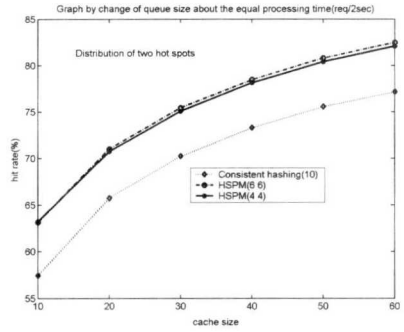

그림 9. 처리속도(req/2sec)일 때 히트 율 비교

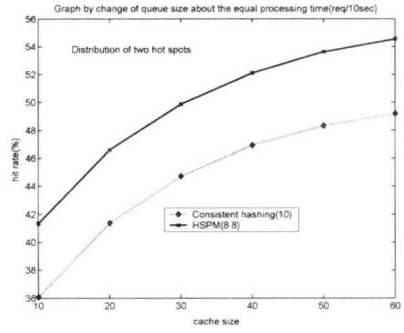

그림 10. 처리속도(req/10sec)일 때 히트 율 비교

트 율이 5% 더 높게 나타났다. 이때 가장 히트 율이 우수한 결과는, HS(1)은 6개의 프록시에 HS(2) 또한 6개의 프록시에 분산 패치한 경우이다. 그림 10은 처리속도가 느린 경우에 그림 9와 동일한 조건으로 실시한 결과를 표시한 그림이다. 이 경우에 HS(1)은 8개 프록시에, HS(2) 또한 8개의 프록시에 분산 패치한 경우에 가장 좋은 히트 율을 보여주었다. 기존의 consistent hashing 방법보다 본 연구에서 제안한 HSPM이 평균적으로 4%의 향상된 히트 율을 보였다.

그림 11은 그림 8과 동일한 방법으로 큐 사이즈를 30으로 설정하고 처리속도가 느려질 때 히트 율의 변화를 나타낸 그래프이다. HSPM 방법의 처리속도가 빠른 부분에서는 HS(1)을 분산시키는 프록시 서버를 4개로 설정하고, HS(2)를 분산시키는 프록시도 역시 4개로 설정 할 경우에 가장 우수한 히트 율을 보였다. 또한 서버의 처리 속도가 느려질 때 각 HS(1)과 HS(2)의 hot spot에 대하여 분산에 참여하는 프록시 서버의 수를 HS(1)은 8, HS(2) 또한 8로 설정했을 경우에 좋은 히트 율을 보였다.

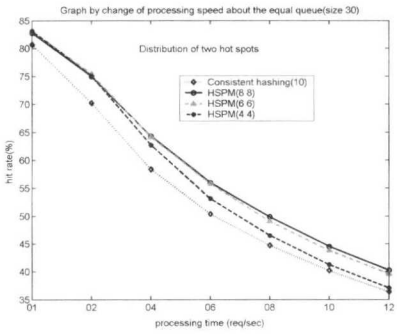

그림 11. 동일한 큐 사이즈(30)의 히트 율 비교

이처럼 프록시의 분산 수를 결정하기 위해서는 분산 웹 캐싱 시스템의 전체 작업량을 고려해야 한다. 작업량이 적고 처리속도가 빠르면 프록시의 개수를 줄여 분산시키고, 작업량이 많고 처리속도가 늦어질수록 많은 프록시에 hot spot을 분산시켜야 한다.

Ⅶ. 결 론

본 연구를 통하여 분산 웹 캐싱 시스템의 구조와 분산 웹 캐싱 시스템의 로드밸런싱에 대하여 살펴보았다. 분산 웹 캐싱 시스템의 부하 조절은 대부분 사용자의 요청에 따른 통계학적 방법이나 현재까지의 요청 특성에 기반 하여 이루어졌다. 이러한 방법은 갑자기 요청량이 증가할 경우에 특정 hot spot에 의해서 시스템의 로드의 불균형이 발생하며 이로 인해 전체 시스템의 성능이 저하된다. 이러한 문제점을 해결하는 방법으로 본 연구에서 부하 불균형을 조절하는 방식으로 hot spot을 미리 예측하며, 부하가 적게 부과되는 공유 프록시에 hot spot을 패치함으로써 로드밸런싱을 이루고, 분산 공유 시스템의 성능을 향상시키는 방법을 제안하였다.

제안한 방법의 성능을 평가하기 위하여 실제 분산 웹 캐싱 시스템의 작동과 유사한 시뮬레이터를 구성하였으며, PERL 프로그램 언어를 이용하여 작성하였다. 각 프록시에 부과되는 부하 정보는 access logs 파일을 분석하여 입수하고 이 자료를 이용하여 향후 발생할 hot spot을 예측하였다. 예측된 hot spot을 부하가 적게 부과되는 공유 프록시에 패치함으로써 부하를 분산하는 효과를 가져왔다. hot spot의 순위가 HS(1)인 한 개의 hot spot 을 부하 분산에 참여시킨 경우와 HS(1)과 HS(2) 모두를 부하 분산에 참여시킨 경우에 대하여 시뮬레이션을 실시하였으며, 두 경우 모두 기존의 consistent hashing 보다 히트 율이 우수하고 각 공유 프록시에 부과된 요청이 균등하게 이루어짐을 알 수 있었다. 따라서 본 연구를 통하여 제안한 HSPM은 분산 시스템에 적용할 경우 히트 율의 증가와 공유 프록시의 활용을 증대시킬 것이다.

참 고 문 헌

[1] M. Arlitt and C. Williamson, "Web Server Workload Characterization : The Search for Invariants," ACM SIGMETRICS 96, 1996.

[2] Lee Breslau et al, "Web Caching and Zipf-like Distributions : Evidence and Implications", IEEE INFOCOM, 1999

[3] The Harvest Group. (1994) Harvest Information Discovery and Access System. [Online]. Available: http://excalibur.usc.edu/science from University of Wisconsin-Madison in

[4] S.Glassman. A Caching Relay for the World Wide Web I. In Proceedings of the Third International World Wide Web Conference, May 1994. pp.69-76

[5] A. Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In Proc. 12th IEEE Intl. Conf. on Data Engineering, New Orleans, Louisiana, Mar. 1996.

[6] D. Wessels and K. Claffy, "Application of Internet Caching Protocol(ICP), version 2",Internet Draft:draft-wessels-icp-v2-appl-00. Work in Progress., Internet Engineering Task Force, May, 1997.

[7] David Karger et al, "Web Caching with Consistent Hashing", In Proceedings of the 8th International World Wide Web Conference, 1999.

[8] V. Valloppillil and K. Ross. " Cache array routing protocol v1.1", February, Internet Draft, http://www.globecom.net/ietf/draft/draft-vinod-carp-v1-03.html, 1998.

[9] Kang Woo Lee et al., "Desing of Cache

Mechanism in Distributed Directory Environment", The Journal of the Korean Institute of Communication Sciences, pp.205-214, vol.22, Feb. 1997

[10] Kyong-Hoon Min et al., "A Web Ccahe Algorithm for Small Organizations",The Journal of the Korean Institute of Communication Sciences, pp.1115-1123, vol.25, Aug. 2000

[11] Kun-Lung Wu et al., "Replication for Load Balancing and Hot-Spot Relief on Proxy Web Caches with Hash Routing", 2003 Kluwer Academic Publishers. Distributed and Parallel Databases Vol. 13, pp.203-220, 2003

[12] P. Barford and M.E. Crovella, "Generating representative web workloads for network and server performance evaluation", In Proceedings of ACM SIGMETRICS Conference, 1998, pp.151-160

[13] Leighton et al. "Global hosting system", united states patent & trademark office, august 22, 2000

[14] Woei-Luen Shyu and Cheng-Shong Wu et al., "Efficiency analyses on routing cache replacement algorithms" Communications, ICC 2002. IEEE International Conference, pp. 2232 -2236, vol.4, 2002

[15] R. Rivest, "The MD5 Message-Digest Algorithm", RFC 1321, Network Working Group, April 1992

정 성 칠(Sung-Chil Jung) 정회원

2002년 2월 : 전북대학교 기계공학과 졸업

2004년 2월 : 전북대학교 제어계측공학과 석사

<관심분야> 웹 캐싱 시스템

정 길 도(Kil-To Chong) 정회원

1984년 6월 : 미국 Oregon State University 기계공학학사

1986년 12월 : 미국 Georgia Institute of Technology 기계공학석사

1992년 12월 : 미국 Texas A&M University 기계공학 박사

1993.9 - 1995. 2 영남대학교 전임강사

1995.3 ~ 현재 전북대학교 전임강사, 조교수, 부교수

<관심분야> 통신 네트워크 성능 분석, Time-Delay, 실시간 멀티미디어 전송, Web 기술