잠재 부하 정보와 HTTP 연결의 에이징을 통한 HTTP 연결 스케줄링 알고리즘

(Load Distribution Policy of Web Server using Subsequent Load and HTTP Connection Time)

김 시 연 † 김 성 천 ††
(Si-Yeon Kim) (Sungchun Kim)

요 약 HTTP/1.0과 새로운 HTTP/1.1이 함께 사용됨으로써 단일 HTTP 연결이 단일 사용자 요청을 의미하던 환경에서 연구되었던 웹 서버 부하 분산 정책은 수정이 불가피하게 되었다. HTTP/1.0 환경에서는 사용자의 단일 요청만을 가지고 서버를 할당하였으나, 지속적인 HTTP 연결을 지원하게 되면서 하나의 HTTP 연결을 통해 여러 개의 요청을 서버에게 요구할 수 있으므로, 첫 번째로 도착한 요청 하나만으로는 앞으로 그 HTTP 연결을 통해 전송될 사용자의 요청이 서버의 자원을 얼마나 소비하게 될 것인지 전혀 예상할 수 없게 된다.

본 논문에서는 HTTP/1.1을 효율적으로 지원하는 부하 분산 정책을 제안하고자 한다. 이 정책은 사용자의 첫 번째 요청이 전달되면 그 요청의 내장 객체 정보와 현재 살아있는 HTTP 연결의 에이징(aging)을 고려하여 서버를 선택하는 알고리즘이다. 그리고 디스패처의 잘못된 분산 결정으로 인해 시스템의 성능에 누적되는 악영향을 최소화하기 위한 기법도 제시한다.

키워드 : 에이징, HTTP/1.1, 웹 서버 클러스터

Abstract With HTTP/1.0, a single request means a single HTTP connection so that the granular unit of dispatching is the same as real load. But with persistent HTTP connection, multiple requests may arrive on a single TCP connection. Therefore, a scheme that dispatches load at the granularity of individual requests constrains the feasible dispatching policies.

In this paper we propose a new connection dispatching policy for supporting HTTP/1.1 persistent connections in cluster-based Web servers. When the request of a base html file arrives, the dispatcher gets the subsequent load arriving on that connection using the embedded objects information. After the dispatcher stores the load information in Load Table, the dispatcher employs the connection aging strategy on live persistent connections on the passage of time. The results of simulation show about 1.7%~16.8% improved average response time compared to existing WLC algorithm.

Key words : Aging, HTTP/1.1, web server cluster

1. 서 론

초기에는 약 50%의 웹 페이지가 기껏 1개의 내장 객체를 지니고 있었으나, 2000년대에 들어서면서 10~40개의 객체가 내장되고 있다[1]. 이렇듯, 한 HTML 문서에 여러 개의 내장 객체가 포함되고 구조가 복잡해지면서 지연시간과 네트워크 과부하를 해결하기 위해 기존의 HTTP 프로토콜에도 변화가 생기기 시작하였다. 하나의 웹 페이지에 포함된 각 개체에 대해서 별도의 TCP 연결을 생성하던 방식에서, 하나의 TCP 연결을 토해 여러 개의 객체를 전송할 수 있도록 HTTP/1.1이 발표된 것이다.

하지만, 실제 전체 HTTP 트래픽 중에서 약 40~50%의 데이타가 지속적인 TCP 연결을 통해 이루어지고 있음에도 불구하고, 웹 서버 클러스터에서는 HTTP 프로토콜의 수정된 내용을 전혀 고려하지 않은, 이전의 부하 분산 정책을 그대로 사용하고 있다[1]. 그리하여 이 논문에서는 새로운 TCP 연결 메커니즘을 고려한 웹 서버 부하 분산 정책을 제안하고, 기존의 기법과의 성능

· 이 연구는 서강대학교 산업기술연구소 지원에 의하여 이루어졌음
† 비 회 원 : 서강대학교 컴퓨터학과
si_yeon@hotmail.com
†† 종신회원 : 서강대학교 컴퓨터학과 교수
ksc@arqlab1.sogang.ac.kr
논문접수 : 2003년 2월 3일
심사완료 : 2005년 8월 31일

을 비교 분석한다.

2. 웹 서버 시스템 구조와 기존 알고리즘

2.1 웹 서버 시스템의 구조

부하 분산을 수행하는 개체의 관점에서 본다면 크게 클라이언트 기반, DNS 기반, 디스패처 기반, 서버 기반으로 분류할 수 있다. 이 네 가지 아키텍처 중에서는 디스패처 기반의 클러스터가 서버 효율성과 사용자 응답 시간 측면에서 가장 뛰어난 성능을 보인다[2]. 그러므로 이 논문에서는 웹 서버 클러스터 아키텍처로 디스패처 기반의 아키텍처를 가정한다.

2.2 디스패처 기반의 웹 서버 클러스터

디스패처는 클라이언트로부터 서비스 요청을 받아와서 서버 풀에 있는 서버 중에서 적절한 서버를 선택하여 요청을 처리하도록 할당한다. 클러스터링 기법에 따라서 디스패처는 서버 풀에 있는 서버들에게 스위치나 네트워크 게이트웨이로 나타난다. 서버 풀에 있는 각 서버는 웹 서버 소프트웨어를 실행하고 있다. 클라이언트 요청은 서버의 풀에 균등하게 할당된다. 그러므로 디스패처 기반의 웹 서버 클러스터에서는 사용자의 요청을 어떤 서버에게 할당할 것인지 결정하는 것이 가장 중요한 이슈가 된다.

2.3 기존의 부하 분산 알고리즘

- 라운드 로빈 정책
- 가중치 라운드 로빈 정책
- 최소 연결 정책
- 가중치 최소 연결 정책
- 최소 부하 정책

위의 기법들은 가장 널리 적용되어오던 방법으로서, 서버에게 차례대로 사용자의 연결을 나누어 주거나, 현재 관리하고 있는 연결의 개수가 가장 적은 서버에게 새로운 연결을 할당한다. 그리고 이런 단순한 방법들이 갖고 있는 취약점을 보완하기 위해 이들 기법에 가중치를 적용하여 연결을 할당하기도 한다.

이들 기법들 중에서 가장 정확한 부하를 측정하여 이를 토대로 연결을 할당하는 것은 최소 부하 정책이다. 하지만, 이 정책은 서버 부하를 계산하는 오버로드가 크기 때문에 실시간성을 확보하지 못한다.

2.4 기존의 부하 분산 알고리즘의 문제점

기존의 부하 분산 정책은 HTTP/1.1이 고안되기 전부터 이미 사용되고 있던 분산 정책이므로 HTTP/1.1 버전을 고려하지 않고 제안된 것이다. HTTP/1.1의 지속적인 연결에서 부하 분산이 중요한 이유는, 한번 수립된 연결을 통해 앞으로 부담하게 될 잠재적인 부하(subsequent load)가 존재하기 때문이다.

HTTP/1.0의 비효율적인 면을 개선하기 위해 HTTP/1.1이 고안됨으로써 지속적인 연결을 통해 효율적으로 사용자의 요청을 처리할 수 있게 되었다. 그러나 HTTP/1.1에서는 기본적으로 분산 정책을 각 요청 단위로 적용할 수 없다. 자세히 말하자면 사용자가 보내는 요청 중에 맨 처음에 들어오는 요청만을 이용하여 HTTP 연결을 할당해야 한다. 그리고 일단 수립된 HTTP 연결을 통해 사용자의 요청이 계속해서 들어오면 디스패처는 이를 별도로 구별하지 않기 때문에 이 요청들에 대해서는 부하 분산 처리를 하지 않으므로, 정확한 서버 부하를 가늠하기 힘들게 되어 부하 분산에 문제가 발생한다.

3. 효과적인 HTTP/1.1의 연결 분산 정책

3.1 기본 전략

이 논문에서 제안하는 기법은 웹 페이지에 포함되어 있는 내장 객체의 정보를 이용하여 하나의 HTTP 연결을 통해 완전한 웹 페이지를 클라이언트에게 완성해주기 위해 요청될 잠재적인 부하를 미리 부하 분산에 적용시키는 것이다. 그리고 적용된 부하 정보에 대해서는 서버의 서비스 처리율을 고려하여 시간이 흐름에 따라 부하가 줄어들고 있다는 사실을 반영시키도록 한다. 이렇게 하면 각 서버가 부담하게 될 작업을 예측할 수 있고, 현재 처리되고 있는 상황도 역시 부하 분산에 고려하게 되므로 지속적인 연결을 통한 사용자의 요청에 대해서도 정확한 부하 분산을 할 수 있다.

3.2 디스패처의 역할

디스패처는 사용자로부터 요청을 받게 되면, HTTP 연결을 통해 처리하게 될 전체적인 부하를 알게 된다. 내장 객체에 대한 요청은 연속적으로 요청이 들어오므로, 객체 요청에 대한 딜레이는 전체 응답 시간에 영향이 거의 없으며, 내장 객체들의 전체 서비스량이 결국 그 연결의 부하를 대표할 수 있으며, 디스패처는 내장 객체 정보를 활용하여 잠재적인 부하 정보를 쉽게 알아낼 수 있다. 일단 이렇게 첫 번째 사용자 요청에 의해 알아낸 잠재 부하 정보를 디스패처의 로드 테이블(load table)에 저장한다. 그리고 그 전에 디스패처는 먼저 로드 테이블의 정보를 바탕으로 하여 새로운 HTTP 연결을 할당할 서버를 선택해야 한다. 그래야만 로드 테이블의 적절한 위치에 정보를 추가할 수 있기 때문이다.

그러나 이렇게 초기 예측 부하량을 반영만 시켜놓고 그냥 방치하면 시간이 흐름에 따라 서버가 처리하는 작업에 대해 부하가 줄어드는 것을 적용하지 못하게 된다. 결국 각 HTTP 연결마다 대체로 내장 객체의 데이타양이 비슷하다면 최소 연결 정책과 다른 점이 없게 된다. 따라서 초기에 디스패처가 알아낸 초기 부하 예측량뿐만 아니라, 현재까지 처리된 부하량도 고려해주어야

한다. 그러므로 TCP 연결의 경과 시간을 고려하여 현재까지 처리된 부하량을 예측하고, 이것을 부하 인덱스에 적용시켜야 한다. TCP 연결의 나이를 고려하여 현재까지의 처리량을 계산해내는 기법을 에이징(aging)이라 부르고, 이런 특성에 기인하여 앞으로 이 논문에서 제안하는 분산 정책을 에이징 부하 분산 정책이라 부르기로 한다.

3.3 서버 선택 알고리즘

TCP 연결의 서비스 처리 결과를 예측하여 에이징 분산 정책을 적용하기 위한 알고리즘을 정리하면 다음과 같다.

==========

while(there are more connections on Server(i))

{

if (the first of all connections on Server(i))

$$age(Conn_K) = time_{curr}(Conn_K) - time_{init}(Conn_K)$$

$$load_{proc}(Conn_K) = age(Conn_K) \times \mu$$

$$load_{curr}(Conn_K) = load_{init}(Conn_K) - load_{proc}(Conn_K)$$

else

$$load_{curr}(Conn_K) = load_{init}(Conn_K)$$

$$load(server) = \sum_{K=1}^{n} load_{curr}(Conn_K)$$

}

$$nextserver = Min\{load(server_1), \ldots, load(server_n)\}$$

==========

로드 테이블에 저장된 정보를 통해서 해당 연결이, 현재 활성화되어 있는 연결 중에서 가장 먼저 맺어진 연결이라면 TCP 연결이 성립된 이후의 경과 시간을 계산하여 현재까지의 처리된 부하량을 예측한다. 그리고 그 나머지 연결에 대해서는 초기 부하량을 그대로 적용한다. 이렇게 각 서버가 현재 부담하고 있는 부하량을 모두 계산하여 그 중에서 가장 부하가 적은 서버에게 새로운 요청을 할당하게 된다.

가장 먼저 맺은 연결에 대해서만 서버 처리율을 고려하는 이유는, 일단 하나의 TCP 연결이 감수하는 데이타량은 서버 운영체제의 한 퀀텀보다 작기 때문이다. 디스패처의 서버 처리율 정보는 주기적인 서버 로그 분석 등과 같은 방법을 통한 경험 데이타를 통해 예측할 수 있고, 서버 처리율은 서버 시스템의 성능에 따라 좌우되므로 일단 설정되면 큰 편차없이 적용할 수 있다.

3.4 웹 서버의 작업

웹 서버는 보통 일반적인 웹 서버가 작동되는 방식으로 작동하다가, HTTP 연결을 종료해야 할 때 FIN 패킷을 디스패처를 통해 보내도록 한다. 그러면 디스패처는 FIN 패킷을 보고 자신의 로드 테이블에 있는 해당 연결 정보를 삭제한다. 이렇게 하면 디스패처는 현재 처리가 종료된 HTTP 연결이 생길 때마다 FIN 패킷 정보를 이용하여 로드 테이블을 업데이트하기 때문에 잘못된 부하 분산 결정의 오차가 심화되는 것을 방지할 수 있게 되므로, 디스패처의 잘못된 서버 결정에 대한 영향이 시스템 전체 성능에 누적되지 않게 한다.

4. 시뮬레이션 및 결과 분석

4.1 모델링

디스패처 기반의 웹 클러스터를 구현하는 방법은 디스패처가 사용자 요청에 대한 응답을 클라이언트에게 분산시키는 기법에 따라 크게 세 가지(NAT, 다이렉트 라우팅, IP 터널링)로 분류될 수 있다. 이 논문에서는 응답 데이타는 디스패처를 거쳐 지나지 않는다고 가정하므로 디스패처를 통해 들어오는 요청 데이타의 분산에만 초점을 맞추고 있다. 단, TCP 연결 종료를 위해 전송하는 FIN 패킷은 디스패처를 통해 사용자측에 전송되도록 하는 시스템을 가정하고 있다.

입력 트래픽에 대해서는 1997년에 발표된 SURGE Workload Generator[3]를 활용하였다. 문서의 인기도는 Zipf 분포에 따라 발생시키고, 이렇게 발생된 인기도 분포를 고려하여 해당하는 파일의 크기를 발생시킨다. 이 파일의 분포는 heavy-tailed 또는 Pareto라 불리는 함수로 발생시킨다.

앞으로 실험의 결과에서 성능 지표로 사용하는 평균 응답 시간이라는 것은 어떤 웹 페이지와 그 웹 페이지에 포함된 내장객체까지 모두 다운로드 받는 데 소요되는 시간을 나타낸다.

4.2 에이징 기법의 효과

다음 실험 결과는 초기 예측 부하량을 적용하였을 때의 효과와 에이징 기법의 효과를 보여준다. WLC는 초기의 데이타 양을 연결이 종료될 때까지 가중치로 사용하는 분산 정책이고, AGE는 활성화된 연결에 에이징 기법을 적용하는 분산 정책이다. 즉, AGE는 모든 조건이 동일한 상태에서 WLC 기법에 에이징을 적용하는 것이다.

그림 1의 그래프에서 LC와 WLC를 비교하면 초기의 HTTP 연결 부하량을 가중치로 사용하였을 경우, 가중치가 성능에 미치는 영향을 알 수 있다. WLC는 LC에 비해 최소 35%에서 최대 55%의 성능이 향상된다. 그리고 AGE와 WLC를 비교해보면 HTTP 연결에 에이징 기법을 적용했을 경우, 에이징이 성능에 미치는 영향을 알 수 있다. 그림 1의 실험에서는 서버의 개수를 4 대, 종료 오버헤드는 750 msec, 서버 선택 오버헤드는 30%로 가정한다. 실험 결과, 트래픽이 복잡한 경우에는 평균 1.71%, 트래픽이 양호한 경우에는 평균 10.25%, 트래픽이 한산한 경우에는 평균 16.82%의 성능 향상이 일어난다.

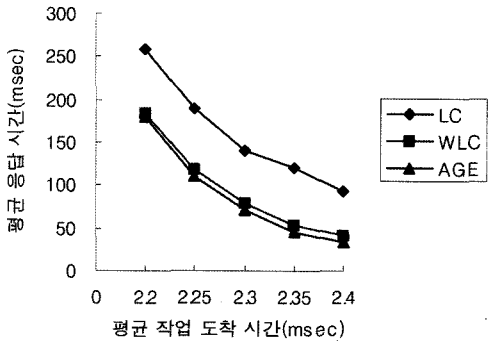

그림 1 초기 예측 부하량과 에이징의 효과

4.3 TCP 연결 종료 오버헤드

지속적인 TCP 연결이 종료될 때에는 해당 연결이 현재 활성화된 연결인지 판단하기 위한 오버헤드가 필요하다[4]. 그리고 연결을 종료한 후에도 실제로 TCP 연결을 종료할 때에는 TIME_WAIT 상태 때문에 오버헤드가 발생한다. 접속자가 아주 많은 사이트의 경우 이 시간동안 몇 천개의 접속을 처리해야 한다. 그림 2를 보면 AGE는 TCP 연결 종료 오버헤드에 대해서 큰 성능의 변화가 없지만 LC는 성능의 차가 매우 크다는 것을 알 수 있다.

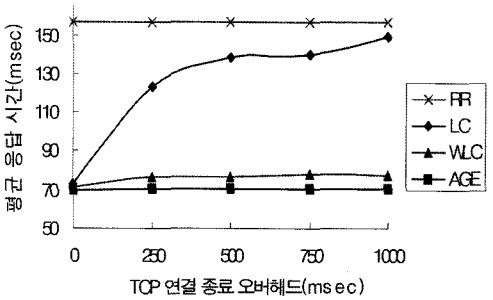

그림 2 TCP 연결 종료 오버헤드에 따른 성능변화

5. 결 론

실험의 결과를 분석한 결과, 일단 정적인 분산 정책(RR)에 비해서는 에이징 분산 정책이 성능의 우위를 보인다. 뿐만 아니라, 정적인 성격과 동적인 성격을 동시에 갖고 있다는 측면에서 에이징 분산 정책과 같은 부류에 속하는 최소 연결 정책(LC)과 비교해 보았을 때에도 뛰어난 성능 향상을 보인다. 같은 맥락으로 최소 연결 정책(LC)과 가중치 최소 연결 정책(WLC)를 비교해 보았다. 여기서 가중치를 주기 위해 사용한 요소는 바로 지속적인 HTTP 연결의 초기에 디스패처가 알아내는 HTTP 연결의 초기 예측 부하량이다. 가중치를 주는 경우는 단순한 최소 연결 정책을 적용하는 경우보다 43.88% 정도의 성능 향상이 일어난다. 그리고 에이징 분산 정책은 이 분산 정책의 지능적인 특성 덕분에 가중치 최소 연결 정책보다 중간 트래픽에서는 평균 10.25%에 이르는 성능 향상을 이룬다. 이 결과를 통해 초기 예측 부하량, 에이징 기법, TCP 연결 종료 시에 일어나는 로드 테이블 업데이트 등의 성격을 지닌 에이징 분산 정책이 매우 좋은 성능 향상을 보임을 입증할 수 있다.

이 논문에서 제안하고 있는 알고리즘은 HTTP 연결의 잠재적인 부하라는 동적인 요소를 가지고 간단한 에이징 기법을 사용하여 정적인 분산 결정을 할 수 있다. 게다가 매번 하나의 HTTP 연결이 끊길 때에는 서버가 FIN 패킷을 디스패처를 통해 클라이언트에게 보내도록 하고 있으므로, HTTP 연결이 끊길 때마다 서버의 상태 정보와 싱크를 맞춰 디스패처의 로드 테이블 정보를 보정한다. 이렇게 함으로써 디스패처의 잘못된 판단이 계속 누적되어 지속적으로 성능에 영향을 끼치는 일을 피할 수 있다.

참 고 문 헌

[1] M. Mikhailov and Craig E.Wills, "Embedded objects in web pages," Tech. Rep. WPI-CS-TR-00-05, Worcester Polytechnic Institute, Worcester, MA, March, 2000.

[2] T. Schroeder, S. Goddard and B. Ramamurthy, "Scalable Web Server Clustering Technologies," IEEE Network, May/June, 2000, pp.38-45.

[3] P. Barford and M. Crovella, "Generating Representative Web Workload for Network and Server Performance Evaluation," Proceedings of the SIGMETRICS International Conference on Measurement and Modeling of Computer System, July, 1998, pp.151-160.

[4] E. Cohen, H. Kaplan and J. D. Oldham. "Policies for managing TCP connections under persistent HTTP," Proceedings of the Eighth International World Wide Web Conference, 1999.

김 시 연

2000년 2월 서강대학교 컴퓨터학과 공학사. 2002년 2월 서강대학교 컴퓨터학과 공학석사

김 성 천

1975년 서울대학교 공과대학 공업교육학(전기전공)학사. 1979년 Wayne State Univ. 컴퓨터공학 공학석사. 1982년 Wayne State Univ. 컴퓨터공학 공학박사. 1982년~1984년 캘리포니아주립대 조교수. 1984년~1985년 금성반도체(주) 책임연구원. 1986년~1989년 서강대학교 공과대학 전자계산소 부소장. 1989년~1991년 서강대학교 공과대학 전자계산학과 학과장. 1985년~현재 서강대학교 공과대학 전자계산학과 교수(1992.9~현재). 1989년~현재 한국정보과학회 병렬처리시스템 연구회 부위원장(1989~1993), 위원장(1994~1997), 대한전자공학회 및 한국통신학회 논문지 편집위원(1991~현재, 1993~현재). 관심분야 병렬처리시스템 (Parallel Computer Architecture, Interconnection Network)