리눅스에서 레이어-7 웹 클러스터링 시스템의 구현 및 사용자 요청률 차이의 인식에 기반한 성능 개선

(Implementation of a Layer-7 Web Clustering System on Linux with Performance Enhancements via Recognition of User Request Rate Variations)

홍 일 구 † 노 삼 혁 ††
(Il-gu Hong) (Sam H. Noh)

요 약 인터넷을 통해 웹이 보편화되면서 폭발적으로 증가하는 사용자의 서비스 요구를 수용하는 것은 점점 더 어려운 문제가 되고 있다. 사용자의 증가를 예측할 수 없는 상황에서 매번 고성능의 시스템을 구입하는 것은 좋은 해결책이 될 수 없다. 즉, 필요할 때에 시스템을 적절히, 간단하게 확장할 수 있는 방법이 있어야 한다. 웹 클러스터링 시스템은 이러한 요구를 수용할 수 있는 기술로서 주목 받고 있다. 본 논문은 웹 클러스터링 시스템 연구에서 두 가지 점에서 기여를 하고 있다. 우선, FreeBSD상에서 구현되었던 Layer-7 스위치 기법 기반의 웹 클러스터링 시스템을, 많은 사용자에 의해 선호되고 있는 Linux 운영체제에 구현하였다. 이 두 운영체제 사이에는 상당한 차이가 있으며 본 논문에서는 Linux상의 구현에 대해 상세히 언급한다. 두번째는 Zipf-like한 웹 요청의 특성을 반영하여 각 요청에 따라 자원을 클러스터 상에서 효과적으로 할당할 수 있는 DS (Dual Scheduling) 부하 분산 기법을 제안하였다. 실험을 통해 이 기법이 시스템 성능을 향상시키는 사실을 보인다.

키워드 : 웹, 클러스터링, Linux, 네트워크

Abstract The popularity of Web service is ever increasing. As the number of services and clients continue to increase, the problem of providing a system that scales with this increase is becoming more difficult. A costly and ineffective method is to buy a new system that is more powerful every time the load becomes unbearable. A more cost effective solution is to expand the system as the need arises. This is the approach taken in Web cluster systems. However, providing effective scalability in a Web cluster system is still an open issue. In this study, we implement a Web cluster system based on Layer 7 switching technique on Linux. The implementation is based on a design proposed and implemented by Aron et al., but on the FreeBSD. Though the design is the same, due to the vast difference between the FreeBSD and Linux, the implementation presented in this paper is totally new. We also propose the Dual Scheduling (DS) load distribution algorithm that distributes the requests to the system resources by observing the variations in the request rate. We show through measurement on our implementation that the DS algorithm performs considerably better than previous algorithms.

Key words : Web, Clustering, Linux, Network

1. 서 론

인터넷이 보편화 되면서 사용자 수는 매년 꾸준히 증가하고 있다. 또한 많은 인터넷 서비스들이 웹으로 통합되면서, 웹 요청은 기하 급수적으로 증가하고 있다. 이러한 상황은 시스템의 선택에 있어 확장성을 중요한 문제로 부각시키고 있다. 이에 발 맞추어 웹 클러스터링은 확장성과 가격대 성능비에서 우수한 시스템 구성 방법으로 주목 받고 있다. 클러스터는 스위치 기술을 바탕으로 저가의 서버 시스템들을 하나의 시스템으로 통합하는 기술로서 사용자의 요청을 부하분산 알고리즘에 따라 클러스터에 포함된 각 서버에 분산시켜 처리한다. 결국 단일 시스템에 가해지던 부하는 클러스터링 시스템

· 이 논문은 2003학년도 홍익대학교 교내연구비에 의하여 지원되었음
† 비 회 원 : XIMETA
ighong@ximeta.com
†† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
samhnoh@hongik.ac.kr
논문접수 : 2004년 1월 29일
심사완료 : 2004년 11월 3일

을 구성하는 각 서버 노드로 나뉘어진다. 따라서 시스템의 부하가 증가함에 따라 발생하는 문제를 클러스터의 노드 수를 증가시킴으로써 어느 정도 해결할 수 있다. 본 연구는 웹 클러스터 환경에 관한 연구로서 다음과 같이 크게 두 가지 기여를 한다. 우선, Aron등[1]이 FreeBSD 운영체제에 구현하였던 확장성이 뛰어난 클러스터링 시스템을 많은 사용자에 의해 선호되고 있는 Linux 운영체제에 구현하였다. 이 시스템은 TCP-handoff 기반의 Layer-7 스위치 기술을 사용하고 있으며 시스템의 확장성을 위하여 전위 서버가 가지는 부하의 일부분을 각 후위 서버로 분산시키는 방법을 사용하고 있다. 비록 같은 기술을 구현하는 내용이지만 FreeBSD와 Linux의 네트워크 코드가 완전히 상이한 관계로 새로운 설계와 구현을 하게 되었다. 두 번째 기여는 새로운 부하 분산 알고리즘의 제안이다. Aron등의 논문에서는 Pei등이 제안한 LARD(Locality Aware Request Distribution)[1,2] 알고리즘을 사용하고 있다. LARD 알고리즘은 지역성을 이용하여 디스크 I/O를 줄임으로써 시스템의 성능을 향상시키는 것을 목표로 하고 있다. 하지만 단순한 지역성만으로는 각 사용자 요청의 패턴을 정확히 반영하기 어렵다. 본 연구에서는 지역성뿐만 아니라 각 사용자 요청의 특성에 따라 시스템 자원을 할당할 수 있는 Dual Scheduling(DS) 알고리즘을 제안한다. 구현된 시스템상에서 실험해 본 결과 LARD 알고리즘에 비해 약 35% 이상의 성능향상을 확인할 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 지금까지 제안되었던 클러스터링 시스템의 여러 가지 형태에 관하여 간단하게 다룬다. 3장과 4장에 걸쳐 TCP-handoff 기반의 클러스터링 시스템을 Linux에서 어떻게 구현하였는지 상세히 다루고, 본 연구에서 제안하는 Dual Scheduling 알고리즘에 대해 설명한다. 5장에서는 실험과 그 결과를 살펴보고, 마지막으로 6장에서 결론을 내리고 차후 연구과제에 대한 언급을 한다.

2. 관련연구

웹 클러스터 기법이 사용되기 이전, DNS 서버를 이용한 간단한 부하 분산 기법이 소개되었다. 라운드로빈(Round Robin) DNS[3]기법은 DNS 서버에서 특정 URL에 대해 다수의 IP 주소를 매핑시키는 방법으로 특정 URL에 대한 요청을 여러 서버에 분산시킨다. 그러나 이 기법은 서버의 상태(부하)를 적절히 반영할 수 없고 URL에 대한 IP 주소가 사용자 시스템과 프록시 서버에서 캐싱되어 사용되는 문제가 있다. 라운드로빈 DNS 기법이 적용된 시스템으로 NCSA(National Center for Supercomputing Application) 사이트가 있다[4].

웹 클러스터링 시스템의 일반적인 구조는 그림 1과 같이 전위 서버와 후위 서버로 구성된다. 전위 서버는 사용자의 요청을 분산시키며 각 노드의 부하를 관리하는 역할을 담당한다. 후위 서버는 사용자에게 실제 서비스를 제공하는 역할을 담당한다. 웹 클러스터링 시스템은 부하 분산시 이용되는 정보에 따라 Layer-4 방식과 Layer-7 방식으로 나눌 수 있다.

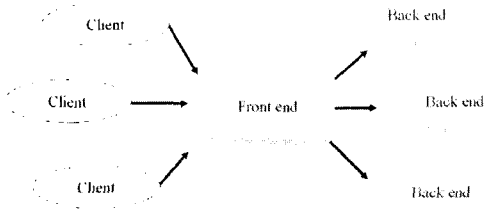


그림 1 일반적인 웹 클러스터링 시스템의 구조

Layer-4 방식은 사용자 요청 내용과 무관하게 각 후위 서버의 부하에 따라 커넥션을 할당한다. 이에 비해 Layer-7 방식은 사용자의 요청에 따라 부하를 분산한다. 즉, 사용자 요청의 내용을 보고 부하를 분산하게 된다. 이하에서는 Layer-4와 Layer-7 웹 클러스터링 기법에 대해 더 자세히 서술한다.

2.1 Layer-4 웹 클러스터링

사용자 요청을 고려하지 않는 Layer-4 기반의 시스템은 전위 서버가 후위 서버의 상태를 능동적으로 반영하는데 한계가 있다. 결국 정적(static) 부하 분산 알고리즘, 혹은 제한적인 능동(dynamic) 알고리즘만을 지원한다. 대표적인 부하 분산 알고리즘은 WRR(Weighted Round Robin)과 Least Connection 등이 사용된다[5].

전위 서버의 역할은 사용자와 후위 서버 사이의 단순한 중계를 담당하는 것이다. 사용자가 웹 서버와 커넥션을 형성하려면 TCP three-way handshaking 과정을 거쳐야 한다. 이 과정 중에 syn 메시지가 포함된 패킷이 전위 서버에 전송된다. 이때 전위 서버는 부하 분산 알고리즘에 따라 후위 서버를 결정하고 패킷 자체를 해당 후위 서버로 전달하게 된다. 위의 과정을 통해서 실제 후위 서버와 사용자 사이에 TCP 커넥션이 형성된다. 전위 서버는 **사용자-후위서버**의 매핑(mapping)정보와 후위 각 서버의 부하 정보를 관리한다. 전위 서버가 후위 서버에 패킷을 전송하는 방식에 따라 NAT(Network Address Translation)[6], IP 터널링(tunneling)[7], 그리고 One IP 기법[8] 등이 사용된다.

2.1.1 NAT(Network Address Translation) 방식

그림 2에서 볼 수 있듯이 NAT(Network Address Translation) 방식에서는 전위 서버가 사용자와 후위 서

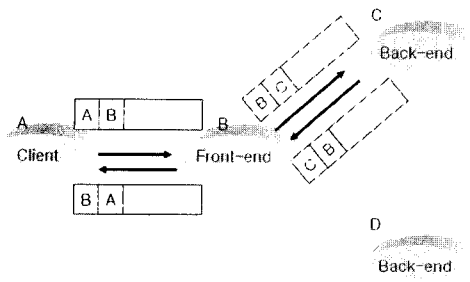


그림 2 NAT 기법을 사용하는 Layer-4 시스템

버 사이에 전송되는 패킷들을 직접 중계하게 된다. 이때 전위 서버는 패킷의 IP 헤더 주소 정보를 변경하여 사용자와 후위 서버 사이의 투명성을 제공한다. 즉, 전위 서버로 전달된 사용자 패킷은 소스 주소를 전위 서버의 주소로, 목적지 주소는 후위 서버의 주소로 바꾸고, 반대로 후위 서버에서 전달된 패킷의 목적지 주소는 사용자의 주소로, 소스 주소는 전위 서버의 주소로 변경하여 전달된다. NAT 방식을 이용한 시스템으로University of California, Berkeley에서 개발된 Magicrouter[9]와 CISCO에서 개발된 LocalDirector[10]를 들 수 있다.

2.1.2 IP 터널링(tunneling) 방식

IP 터널링 방식은 NAT 방식에서 전위 서버가 갖는 부하 즉, 모든 패킷들이 전위 서버를 통해 전달할 때 발생되는 부하를 줄이기 위해 소개 되었다[7]. IP 터널링 기법을 사용하는 클러스터링 시스템의 모든 서버 노드들은 가상 주소와 실제 주소라는 2가지 주소를 갖는다. 가상 주소는 클러스터를 대표하는 주소로써 클러스터에 포함된 모든 서버들이 동일한 값을 갖는다. 사용자는 이 주소를 통해 클러스터에 요청을 보내게 된다. 단, 후위 서버들은 가상 주소에 대한 ARP(Address Resolution Protocol) 요청을 무시하며, 전위 서버만이 이 요청을 처리하게 된다. 결국 전위 서버만이 사용자에게 노출된다. 전위 서버로 전달된 패킷은 후위 서버의 실제 주소를 이용하여 IP 터널링 헤더를 구성, 캡슐화(encapsulation)된 상태로 후위 서버로 전달된다. 후위 서버에서는 IP 터널링헤더를 제거하고 자신에게 전달된 패킷을 처리하게 된다. 이때 후위 서버에서 생성된 패킷은 전위 서버를 거치지 않고 서비스 요청자에게 직접 전달된다. 그림 3이 IP 터널링 방식에서 시스템간의 관계를 보여주고 있다. Linux Virtual Server Project[11,12]에서는 IP 터널링을 비롯한 Layer-4기반의 클러스터링 툴들을 공개하고 있다.

2.1.3 One IP 방식

그림 4는 One IP 기법을 보여준다. One IP 기법은 IP 터널링 기법과 동일한 가상주소 기법을 사용한다. 단지 IP 터널링 헤더를 사용하지 않고 MAC(Media

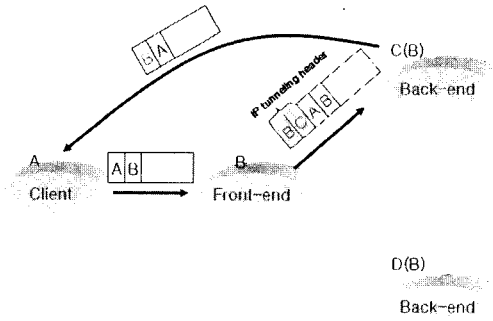


그림 3 IP 터널링 기법을 사용하는 Layer-4 시스템

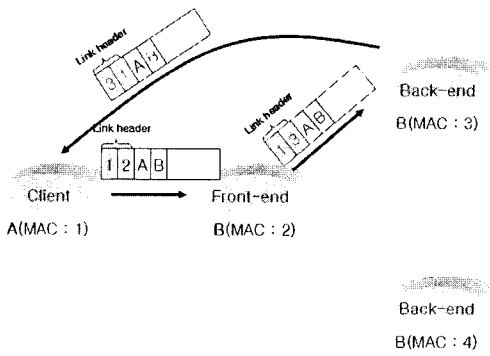


그림 4 One IP 기법을 사용하는 Layer-4 시스템

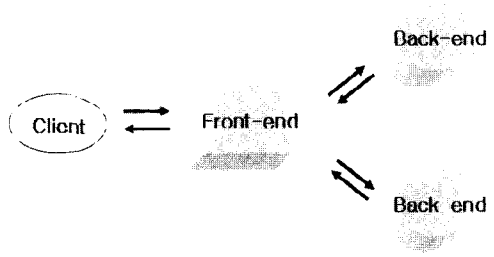
Access Control) 주소를 변경하여 패킷을 전달한다는 차이점이 있다. 이 기법은 IP 터널링 헤더를 생성하고 처리하는데 사용되는 부하는 없지만 클러스터 시스템이 동일한 네트워크(스위치)안에 구성되어야 하는 제약을 갖게 된다[8]. One IP 기법을 이용한 대표적인 클러스터링 시스템으로 IBM의 NetDispatcher가 있다[13]. 이 시스템은 실제로 1996년 하계 올림픽 게임과 1998년 동계 올림픽 게임 웹 사이트에 사용되었다.

2.2 Layer-7 웹 클러스터링

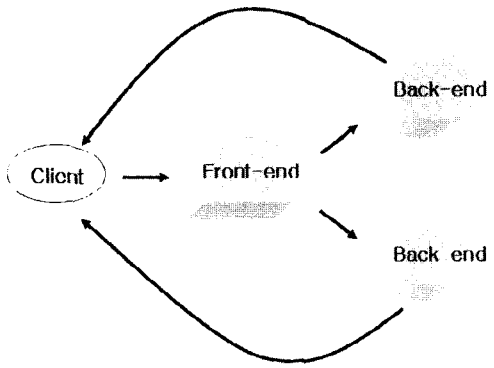
웹을 통한 서비스의 다양화와 콘텐트(content)의 양적 증가는 새로운 서비스 환경을 요구하고 있다. 클러스터링 시스템에서 다양한 서비스를 지원하기 위해서는 서비스에 따른 부하 분산이 필요하다. 또한 증가하는 콘텐트를 쉽게 관리할 수 있는 방법을 요구하고 있다. 이러한 웹 환경의 변화는 사용자 요청 내용에 따라 부하를 분산시킬 수 있는 Layer-7 기반 시스템의 필요성을 증가시키고 있다.

Layer-7 기반의 클러스터 시스템은 사용자와 전위 서버 사이에 이미 커넥션이 설정된 상황에서 사용자의 요청 내용에 따라 부하를 분산한다. 즉, CARD (Content Aware Request Distribution) 기반의 부하 분산 알고리즘을 사용한다. Layer-4 기반 기법과 마찬가지로

Layer-7 기반 기법은 전위 서버에서 후위 서버로 데이타를 전송하는 방식에 따라 TCP-splicing과TCP-handoff로 나뉠 수 있다. 그림 5는 Layer-7 기반 클러스터링 기법을 보여준다.



(a) TCP-splicing



(b) TCP-handoff

그림 5 Layer-7 기반 웹 클러스터링 기법

2.2.1 TCP-splicing 기법

TCP-splicing 기법은 NAT 방식과 마찬가지로 사용자와 후위 서버 사이에 데이타 전송을 전위 서버가 중계하게 된다. 이들 두 기법의 차이점은 패킷을 전달하는 과정에서 NAT 방식과는 다르게 TCP-splicing기법에서는 모든 패킷을 TCP/IP 네트워크 스택을 통하여 전달해야 한다. 따라서, 전달되는 데이타의 양이 많으면 시스템의 부하가 증가하게 되어 확장성이 현저히 떨어지는 문제가 있다[2,14].

TCP splicing에 관련된 연구로는 다음과 같은 것들이 있다. Cohen등은 TCP-splicing 기법을 프록시 서버에 적용하였다[14]. Yang과 Luo는 TCP splicing 기법을 이용한 Linux 웹 클러스터링 시스템을 구축하였다[15,16]. Zhang등은 HACC(Harvard Array of Clustered Computers) 프로젝트에서 Windows NT를 사용하여 TCP-splicing을 기반한 클러스터링 시스템을 구성하였다[17].

2.2.2 TCP-handoff 기법

TCP-handoff 기법은 TCP-splicing 기법이 전위 서버에서 모든 커넥션을 관리하는 것과 다르게 전위 서버와 사용자 사이에 생성된 커넥션을 핸드오프 프로토콜을 통하여 후위 서버로 전달한다. 후위 서버는 핸드오프 프로토콜을 통해 전달된 커넥션 정보를 이용하여 가상 소켓을 생성하고 그것을 사용자와 후위 서버 사이에 직접 연결된 커넥션으로 인식하게 된다. 핸드오프가 완료되면 사용자로부터 전위 서버로 전달되는 모든 패킷은 NAT 기법을 통하여 후위 서버로 전송되고 후위 서버에서 처리되어 전위 서버를 거치지 않고 직접 사용자에게 전달된다. TCP-handoff 기법이 갖게 되는 부하는 핸드오프 프로토콜 과정에서 발생되는 추가적인 패킷 교환과 새로운 가상 커넥션의 생성에 사용되는 부하로 볼 수 있다. 결국 TCP-handoff 기법의 부하는 커넥션 수에 비례한다. 또 한가지 Layer-4 기반 기법에 비해 추가적으로 고려해야 하는 것은 부하 분산 방법이다. TCP-handoff 기법을 포함한 Layer-7 기반 기법에서는 사용자의 요청을 부하 분산 이전에 알아낼 수 있다. 따라서 이를 효율적으로 활용할 수 있는 기법이 필요하다. 이러한 맥락에서 Pai와Aron등은 지역성을 고려한 LARD(Locality Aware Request Distribution) 부하 분산 알고리즘을 제안하였고 이를 이용한 클러스터링 시스템을 구현하였다[2,18]. 그러나 이러한 경우 사용자 요청을 확인하기 위한 부가적인 연산과 커넥션 관리, 그리고 부하 분산을 위한 복잡한 연산에 의해 전위 서버의 부하를 높일 수 있다는 문제점을 갖고 있다.

Aron[1]등은 전위 서버의 부하로 확장성이 떨어지는 문제를 해결하기 위하여 확장성이 높은 시스템을 위한 새로운 클러스터링 구성 기법을 제시하였다. 이 방식에서는 기존에 전위 서버가 가지고 있던 부하를 크게 두 부분으로 나눈다. 첫째는 후위 서버를 결정하는 부하 분산 부분이고 둘째는 전위 서버와 후위 서버의 커넥션을 관리하는 부분이다. 이 두 부분에서 상대적으로 부하가 큰 후자를 후위 서버에서 처리하도록 하여 시스템의 확장성을 높였다. 그림 6은 이러한 확장성이 향상된 웹 클러스터링 시스템을 도식적으로 보여준다. 클러스터링 시스템 전위에는 라운드로빈 DNS 서버 혹은 Layer-4 스위치를 두고 사용자 요청을 서버들 사이에 분산시킨다. 사용자 요청은 각 서버로 전달되고 서버는 사용자의 요청을 패킷에서 읽어 서버 ID와 함께 부하 분산을 담당하는 서버에 전달한다. 부하 분산 서버는 각 서버의 상태와 지역성을 고려하여 이를 처리할 서버를 결정한다. 이 서버가 결정된 후 최초 사용자의 요청을 받았던 서버는 핸드오프 프로토콜을 이용하여 해당 서버로 커넥션을 전달한다.

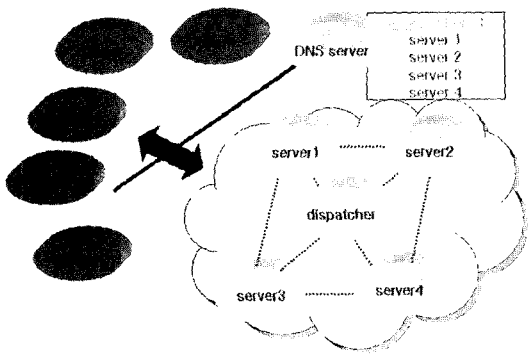


그림 6 확장성이 향상된 Layer-7 기반 클러스터링 시스템 구성

지금까지 웹 클러스터링 기법에 관하여 간단하게 살펴보았다. 본 논문에서는 이들 중 Aron등의 확장성이 향상된 Layer-7 기반의 클러스터 시스템을 Linux에 구현하였다. 다음 3장과 4장에서는 이 구현에 대해 자세히 살펴본다.

3. Layer-7 기반 Linux 웹 클러스터링 구현

본 논문에서는 Aron등[1]의 연구에서 FreeBSD 시스템에 구현하였던 클러스터링 시스템을 Linux 커널 2.4.16 버전에 구현하였다. 본 연구에서 기존 FreeBSD에 구현된 시스템을 사용하지 않고 Linux에서 재차 구현하게된 이유는 크게 두 가지이다. 첫째는 FreeBSD에 비해 Linux가 공개 소프트웨어로서의 활용도가 높고 많은 사용자를 갖고 있다는 점이다. 따라서 본 연구의 결과가 활용된다면 그 파급효과가 더 클 것으로 예상했다. 두 번째는 새로이 제안하는 알고리즘을 실험해 볼 테스트베드가 필요했다는 점이다. Aron등에 의한 구현은 소스코드가 공개되어 있지 않아 새로운 부하 분산 알고리즘에 대한 검증을 할 수 없는 실정이었다. 본 연구는 이 분야의 지속적인 연구를 위해서는 이러한 새로운 구현이 반드시 필요한 것으로 판단하였다. 본 연구는 이러한 점들을 고려하여 Linux 기반의 테스트베드를 구축하였다. 특히, FreeBSD에서 구현한 Aron등의 시스템과는 달리 Linux의 네트워크 코드는 상이한 점이 많아 새로운 설계와 구현을 하게 되었다.

Linux 네트워크 코드는 2개의 소켓 인터페이스 즉, BSD와 INET 소켓 인터페이스를 지원한다. BSD 소켓 인터페이스는 사용자 프로그램과 커널에 소켓 API를 제공한다. INET 소켓 인터페이스는 Linux 커널 내부에서 사용되며 BSD 인터페이스와 네트워크 스택 사이에 존재한다. 본 구현에서는 INET 소켓 인터페이스 계층만을 이용한다. TCP/IP 네트워크 스택에서 BSD 소켓 인

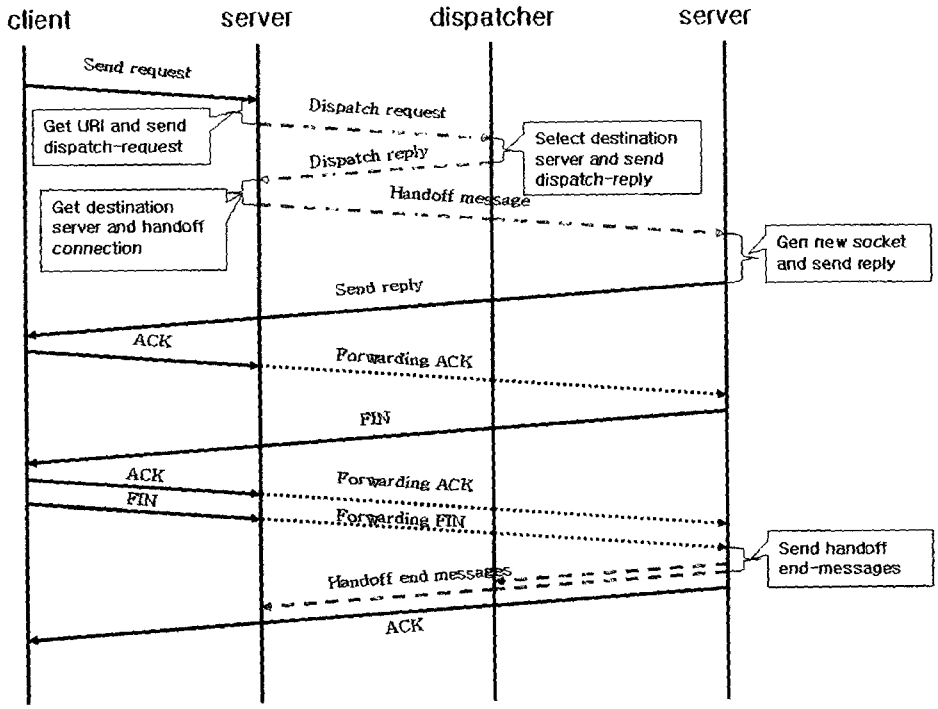


그림 7 패킷 흐름도

터페이스로 전달되는 메시지를 가로채고 이를 처리할 새로운 핸들러를 정의하였다. 또한 전송 메커니즘을 지원하기 위한 코드를 IP 스택과 TCP 스택 사이에 삽입하였다.

그림 7은 클러스터 시스템에서 시간에 따른 패킷의 흐름을 보여준다. 사용자는 DNS 쿼리를 통해서 클러스터링 시스템에 포함된 서버를 선택하고 요청을 전달한다. 사용자 요청을 받은 서버는 URI(Uniformed Resource Identifier)를 읽어 들이고 제어 커넥션을 통해서 부하 분산기에 전달한다. 부하 분산기에서는 이 요청을 처리할 서버를 결정하여 알려준다. 사용자의 요청을 받은 서버는 선택된 서버로 핸드오프 프로토콜을 통해 커넥션을 전달한다. 결국 사용자 요청은 부하 분산기에 의해 선택된 서버에서 처리된다. 그림 7에서 굵은 점선은 제어 커넥션을 의미하고 엷은 점선은 전송 메커니즘을 통해 전달되는 패킷을 의미한다.

3.1 제어 커넥션의 초기화와 새로운 소켓 핸들러 설정

시스템의 초기화 단계에 제어 커넥션이 형성된다. 제어 커넥션은 클러스터를 구성하는 모든 노드 사이에 생성되며 시스템 콜을 통해서 새로운 *sock* 핸들러가 제어 커넥션을 구성하는 *sock* 구조에 설정된다. *sock* 구조는 커넥션 종류를 구별하기 위한 *status* 필드와 핸드오프와 관계된 **주소 정보**(핸드오프를 일으킨 서버의 주소, 핸드오프를 통해 새로운 소켓을 형성한 서버의 주소)를 저장하기 위한 필드가 추가 되었다. 그림 8은 Linux 커널 네트워크 스택 인터페이스를 보여준다. 네트워크를 통해 전달된 패킷은 각 프로토콜 스택의 처리가 완료되면 새로 정의된 *sock->data_ready* 핸들러를 호출한다.

앞에서 살펴 보았듯이 제어 커넥션은 클러스터를 구성하는 각 서버 사이에 메시지를 전달하기 위해 사용된다. 사용자 요청을 처리할 서버를 결정할 때 부하 분산기와 서버 사이에 주고 받는 메시지들과 핸드오프 과정 중 서버들 사이에 주고 받는 메시지들은 이 제어 커넥션을 통해서 전달된다.

3.2 핸드오프 프로토콜

TCP-handoff 기법은 사용자와 최초 사용자 요청을 받아들인 서버 사이에 생성된 커넥션을 부하 분산 서버에 의해 결정된 서버로 전달하는 메커니즘이다. 이 기법을 이용한 사용자와 서버간의 교류는 다음 네 과정을 거치면서 실행된다. 첫째, 사용자와 연결된 커넥션의 정보를 실제 처리할 서버로 전달하여 가상의 소켓 구조를 생성한다. 둘째, 사용자가 전달한 패킷을 핸드오프 프로토콜을 통해 생성된 가상 소켓으로 전송한다. 셋째, 실제 사용자의 요청을 처리한 서버는 응답을 사용자에게 직접 전달한다. 넷째, 커넥션이 종료되면 이를 기존의 소켓에 종료 사실을 알려준다. 아래에서 각 과정에 대한 구현을 구체적으로 설명한다.

첫째, 새로운 가상 소켓 구조를 생성하기 위해서 사용자와 서버 사이에 생성된 소켓의 정보를 부하 분산기에 의해 결정된 서버로 보내야 한다. 이 메세지에는 사용자

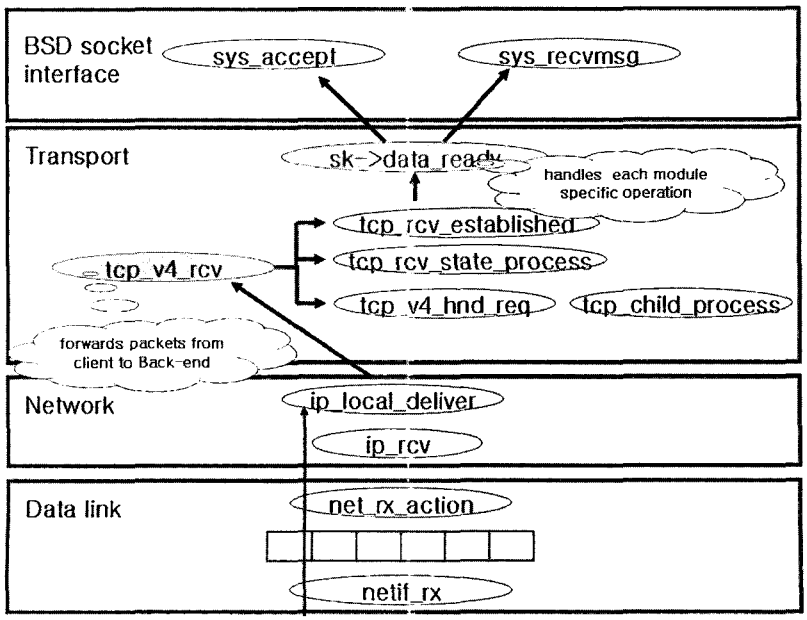


그림 8 Linux 네트워크 스택

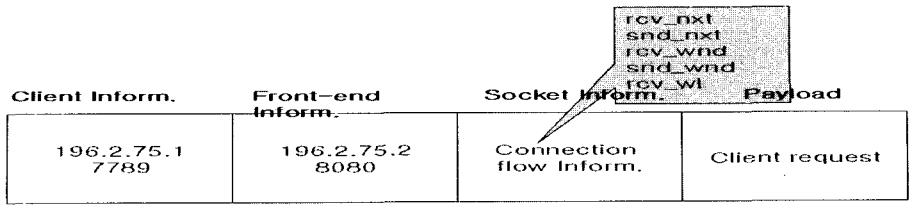


그림 9 핸드오프 프로토콜 메시지 형식

의 주소 정보, 메시지를 보내는 서버의 주소 정보, TCP 흐름 제어를 위한 정보, 그리고 사용자 요청의 내용이 포함된다. 실제 사용자 요청을 처리할 서버는 이 정보를 이용하여 가상 소켓을 생성하고 웹 서버 listen 소켓의 *accept_queue*에 삽입시킨다. 그림 9는 핸드오프 프로토콜 메시지 형식을 보여준다. 사용자의 주소와 TCP 흐름제어 정보는 가상 소켓을 생성할 때 사용된다. 흐름 정보와 관계된 필드의 의미는 RFC 0793에 정의되어 있다[19]. Payload정보는 사용자 요청의 내용을 포함하고 있다. 사용자의 요청은 *sk_buff* 구조에 실려 가상 소켓 생성시 소켓의 *receive_buff*에 전달된다. 소켓이 생성되고 나면 실제 서비스를 제공하는 서버는 이 소켓을 사용자와 서버 사이에 직접 연결된 커넥션으로 인식하게 된다. 핸드오프를 일으킨 서버의 주소는 사용자 요청에 대한 응답을 보낼 때 사용된다. 이것은 아래에서 살펴본다.

둘째, 사용자의 패킷이 처음 사용자 요청을 받았던 서버에 도착하면 소켓의 *status* 정보를 통해 핸드오프가 발생되었는지를 살펴보게 된다. 이미 핸드오프가 발생되었다면 패킷을 가상 소켓으로 전달하기 위해서 주소 정보를 변화시키게 된다. 이 때 목적지 주소는 실제 이 패킷을 처리할 서버의 주소로 바뀌게 된다. 주소 변환이 완료되면 패킷의 checksum 값이 재계산되어 해당 서버로 전달된다.

셋째, 사용자 요청을 처리하고 응답을 보낼 때, 서버는 *status* 필드를 통해 소켓이 핸드오프 프로토콜을 통해 생성된 가상 소켓인지를 확인한다. 가상 소켓인 경우 소켓의 소스 주소는 핸드오프를 일으켰던 서버의 주소로 바꾸어 전달되게 된다. 이 과정을 통하여 사용자에게 클러스터 시스템의 투명성을 제공하게 된다.

넷째, 가상 소켓은 fin/reset 설정된 패킷을 받게 되면 제어 커넥션을 통해 가상 커넥션이 종료되었음을 핸드오프를 일으킨 서버와 부하 분산기에 알려주게 된다. 부하 분산기에서는 이 메시지를 받으면 사용자 요청을 처리한 서버의 부하를 감소시키게 된다. 핸드오프를 일으킨 서버에서는 사용자와 서버 사이에 생성된 소켓을 반환하게 된다. 그림 10은 핸드오프 메커니즘을 통한 패킷

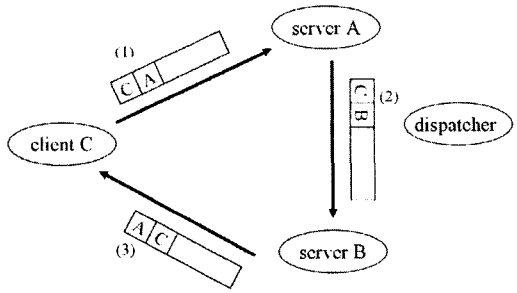


그림 10 핸드오프 프로토콜을 통한 패킷의 전송 메커니즘

의 전송과정을 보여준다.

3.3 부하 분산기

사용자와 최초 연결을 맺은 서버에 사용자로부터 요청이 전달되면 새로 정의된 핸들러는 패킷에서 요청을 읽고 4-byte의 해쉬 값을 만든다. 이 해쉬 값과 서버의 ID를 제어 커넥션을 통해서 부하 분산기에 전달한다. 부하 분산기는 **사용자 요청-실행 서버** 매핑 정보와 각 **서버의 부하** 정보를 이용하여 이 요청을 처리할 서버를 결정한 후 제어 커넥션을 통하여 전달한다. 처음 사용자 요청을 받은 서버는 선택된 서버가 자신이라면 웹 서버 listen 소켓의 *accept_queue*에 사용자와 연결된 상태를 가지는 소켓을 삽입시킨다. 만약 선택된 서버가 자신이 아니라면 핸드오프 프로토콜을 통해 사용자 요청의 처리를 넘겨주게 된다.

앞에서 살펴본 것과 같이 부하 분산기는 다음 두 가지 정보를 관리하게 된다. 첫째, **사용자 요청-실행 서버** 정보로서 사용자 요청의 해쉬 값을 인덱스로 하는 테이블로 구성된다. 이를 통해 부하 분산기는 시스템의 특성, 예를 들어 지역성을 이용한 부하 분산이 가능하게 된다. 둘째, 각 **서버의 부하** 정보로서 전체 클러스터링 시스템 자원을 효율적으로 분배하는데 사용된다. 이러한 정보를 이용하여 효율적인 부하 분산 정책을 펼치게 되는데 4장에서 부하 분산 알고리즘에 대해 좀 더 자세히 다루도록 한다.

4. 부하 분산 알고리즘

일반적으로 웹 사용자 요청의 패턴이 Zipf 분포를 따른다는 사실은 잘 알려져 있다[20-22]. 즉, 적은 수의 특정 웹 요청이 전체 부하의 상당 부분을 차지한다. 다시 말하면 각각의 요청은 상이한 참조 확률과 빈도를 보여준다. 예를 들어 특정 호스트의 "index.htm" 파일은 다른 파일들에 비해 더 높은 확률과 빈도로 참조된다. 또 다른 예로 9.11 테러 시 특정 파일에 대한 요청이 순간적으로 폭주하면서 시스템을 다운시키는 현상을 관찰할 수 있었다. 결국 각 요청의 상이한 패턴을 시스템에 반영하기 위해서는 자원의 할당과 부하 분산 과정에서 각 요청의 성격을 적극적으로 수용할 수 있어야 한다.

이러한 웹의 성격을 시스템에 반영하기 위하여 일부 웹 서버의 경우 캐싱 메커니즘을 이용하여 자주 참조되는 페이지를 메모리에 상주시켜 지역성을 활용하고 있다. Layer-7 기반의 웹 클러스터링 시스템에 적용된 대표적인 부하 분산 알고리즘인 LARD는 지역성에 바탕을 두고 부하 분산을 한다. 하지만 지역성만으로 웹 요청의 패턴을 정확히 반영하는 것에는 어려움이 따른다. 본 논문에서는 이러한 문제를 해결하기 위하여 DS (Dual Scheduling) 알고리즘을 제안하였고 기존의 LARD 알고리즘과 성능을 비교하였다.

4.1 LARD 알고리즘

LARD(Location Aware Request Distribution) 알고리즘은 지역성에 바탕을 둔 부하 분산 알고리즘이다. 이 알고리즘은 가장 최근에 특정 요청을 처리한 서버에 동일 요청을 할당하여 불필요한 디스크 I/O를 줄임으로써 전체 시스템의 성능을 향상시키는 것을 목적으로 한다. 이때 서버의 부하가 집중되어 병목현상이 발생하는 것을 막기 위하여 서버의 부하가 특정 값 이상으로 증가하지 못하도록 제약을 둔다. LARD 알고리즘은 각 서버의 부하 정보와 사용자 요청-실행 서버 매핑 테이블을 분산기에 두고 활용한다.

그림 11은 LARD 부하 분산 알고리즘을 보여준다. 사용자의 요청이 들어오면 각 서버 노드에 부하들이 재계산된다. 각 서버의 부하 계산시 두 가지 점을 고려한다. 첫째 사용자 요청의 지역성을 고려하게 된다. 즉, 전달된 사용자의 요청을 과거에 이미 처리했던 서버 노드가 있다면 버퍼 캐쉬에 이 요청에 대한 응답이 있을 확률이 높을 것이라는 점을 이용한다. 둘째 각 서버의 현재 부하 상태를 고려하게 된다. 각 서버의 부하는 사용자 요청이 스케줄링될 때마다 증가되고 서비스가 완료되면 감소된다. 테이블을 통해 관리되는 각 서버의 부하 정보는 특정 서버에 현재 부하가 서버의 성능을 저하시킬 수 있는 범위 이상으로 선택되지 않도록 하는데 사용된다. 본 논문에서는 이를 위해 최대 부하를 Pai[2] 논문에서 실험시 사용한 80으로 설정하였다. 전달된 사용자의 요청에 대해 그림 11의 cost 함수를 이용하여 노드의 부하를 계산한 후 가장 적은 부하를 가지는 노드를 선택한다.

```
int get_node(request)
{
        min_cost = cost(0,request);

        for(i=0 ; i < node_num; i++)
        {
                temp = cost(i,request);
                if(temp < min_cost) {
                        min_cost = temp;
                        temp_id = i;
                }
        }

        addload(&node_load[temp_id]);
        return temp_id;
}
int cost(node_id, request)
{       cost;
        if( file[request].node_id == node_id) cost += hit_cost;
        else cost += miss_cost;
        if( load(node_id) < MAX_LOAD) cost += bal_cost;
        else cost += inval_cost;
        return cost;
}
```

그림 11 LARD 알고리즘

4.2 사용자요청을 감안한 DS(Dual Scheduling) 알고리즘

앞에서 살펴본 것과 같이 LARD 알고리즘은 지역성을 바탕으로 웹 요청을 분산시킨다. 모든 웹 요청이 동일한 성격(빈도, 확률, 패턴)을 갖는 상황에서 LARD 알고리즘은 효율적으로 동작한다. 불행히도 앞에서 언급한 것과 같이 각각의 웹 요청은 매우 상이한 성격을 보여주고 있으며 결국 LARD 알고리즘처럼 웹 요청의 지역성과 부하만으로 시스템 자원을 분배하는 방법은 한계를 갖게 된다. 예를 들어 매우 짧은 시간동안 특정 요청이 서버로 폭주하는 경우를 생각해 보자. LARD 알고리즘에서 각 요청은 부하 분산기를 통해 처음 노드 A에 할당될 것이다. 웹 요청이 노드 A의 한계-실험에서는 80으로 설정되었다-를 초과하기 전에는 오직 노드 A로 요청이 할당될 것이다. 노드 A가 한계점에 도달하게 된 후에야 새로운 노드 B가 할당되고 요청은 B로 향하게 된다. 이제 웹 요청들은 노드 B가 한계 지점에 도달할 때까지 노드 B를 통해 서비스를 받게 될 것이다. 결국 폭주하는 요청에 대하여 가용한 자원에 비해 실제 자원의 할당이 지연되고, 자원의 분할이 적절하게 이루어 지지 않는 문제가 발생하게 된다. 결과적으로 자원의 할당이 특정 노드의 부하와 지역성에 의해서만 결정됨으로써 특정 요청의 성격에 민감하지 못한 한계를 갖게 된다. 위의 예에서 특정 요청이 빈번하게 발생되는 경우

이 요청들을 처리할 서버의 수를 신속하게 증가시키고 요청의 빈도가 감소하면 즉시 자원을 회수 할 수 있다면 좀더 효과적으로 시스템 자원을 관리할 수 있을 것이다. 문제는 이러한 일련의 작업이 지역성과 서버의 부하에 따라 자원을 할당하는 기존의 메커니즘을 보존하면서 이루어져야 한다는 점에 있다. 이러한 문제를 해결하기 위하여 본 논문에서는 Dual Scheduling(DS) 기법을 제안한다.

DS 알고리즘의 기본적인 아이디어는 요청을 처리할 자원의 할당 시, 지역성과 부하 외에 추가로 단위 시간당 요청의 빈도를 고려하는데 있다. 예를 들어, 특정 요청이 폭주하는 상황에서, 시스템은 단위 시간당 빈도가 증가하게 되면 이를 처리할 서버들을 증가시키고 요청의 단위 시간당 빈도가 줄어들면 자동적으로 할당된 서버들을 줄인다. 요청의 단위 시간당 빈도가 높지 않을 경우 부하 분산기는 LARD 알고리즘과 동일한 메커니즘에서 동작하게 된다. 또한 특정 요청의 단위 시간당 빈도가 증가한 상황에서, 웹 요청이 직접 전달된 서버에서 처리할 수 있을 경우 부하 분산기를 거치지 않고 직접 처리하여 패킷교환을 줄임으로써 시스템의 성능을 향상시킨다.

```
[Dispatcher module]
int schedule(node ID, request, access_tm)
{
        if( is_hot(request, access_tm) )  // 요청의 성격을 판단
                return alloc_new_node(request); // 새 서버 할당
        else return get_node(request);  // 지역성에 의존한 할당
}

int alloc_new_node(request)
{
        return reschedule(file[request],node);
}

[Real server module]
void check_schedule(request, access_tm)
{
        if( is_hot(request, access_tm) )  // 요청의 성격을 판단
        {
                serve(request);         // 로컬 서버에서 처리
        }else {
                clear_request(request); // 기존의 할당 정보 무효화
                send_request_to_disp(request); // 부하 분산기에 요청
        }
}
```

그림 12 Dual Scheduling 알고리즘

그림 12는 DS 알고리즘을 보여준다. 이 알고리즘은 크게 두 부분으로 나뉜다. 첫번째 모듈은 부하 분산기에 존재하며(그림 12의 [Dispatcher module]), 나머지는 각각의 서버에 존재한다(그림 12의 [Real server module]). 부하 분산기는 일반적인 경우 get_node 함수를 통하여 기존의 LARD 알고리즘을 이용하여 지역성과 부하를 고려해서 사용자 요청을 스케줄링 하게 된다. 하지만 특정 요청의 단위 시간당 빈도가 증가하게 되면 alloc_new_node 함수를 통하여 새로운 노드를 할당하게 된다. 위의 메커니즘을 지원하기 위하여 부하 분산기는 각 요청의 요청-시간정보를 저장한다.

반면 각 서버의 [Real server module]에서 적용되는 알고리즘은 다음과 같다. 중앙의 부하 분산기에서 특정 웹 요청이 할당되면, 서버는 이 요청이 할당 되었음을 기록한다. 이때 부하 분산기와 마찬가지로 요청에 대한 요청-시간 정보를 저정한다. 다음 특정 웹 요청이 사용자로부터 서버에 전달되면, 서버는 우선 이 요청이 부하 분산기로부터 서버에 이미 할당된 요청인지를 조사하고, 이 요청에 대한 단위 시간당 빈도를 측정한다. 만약 단위 시간당 빈도가 충분히 크고 현재 서비스 노드의 부하가 한계를 넘지 않았다면, 부하 분산기를 거치지 않고 serve 함수를 통해 로컬에서 처리하게 된다. 만약 단위 시간당 요청 빈도가 크지 않거나 부하 분산기에 의해 스케줄링된 적이 없다면 send_request_to_disp 함수를 호출하여 부하 분산기로 스케줄링을 요청한다. 이때 할당된 요청의 정보는 무효화된다. 시스템 전체의 노드 정보를 유지하기 위하여 각 서비스 노드들은 자신의 노드 정보를 유지 관리하고, 부하 분산기로 전달되는 요청에 추가적으로 이 정보를 보낸다.

DS 알고리즘을 지원하기 위해서 각 클러스터 서버는 부하 분산기에서 사용된 사용자 요청-실행 서버 테이블과 동일한 해쉬 테이블을 관리하게 된다. 또한 기존에 중앙에서 관리되던 서버의 부하정보를 직접 관리하며, 중앙의 부하 분산기는 서버와 정보를 주고 받을 때마다 이 정보를 갱신한다. 단위 시간당 빈도는 저장된 요청-시간 정보와 현재 도착한 웹요청의 시간차에 의해 계산된다. 요청-시간 정보는 서버에 의해 웹 요청이 처리될 때마다 갱신되게 된다.

그림 13은 변형된 패킷 흐름도를 보여준다. 그림 13에서 알 수 있듯이 이 기법은 부가적으로 케쉉 메커니즘을 사용하여 불필요한 패킷 교환을 줄이는 효과를 얻을 수 있다.

5. 실험 및 결과

실험 환경을 구성하기 위하여 클러스터 서버 노드로 Pentium III 800MHz, 128MB RAM을 갖춘 시스템 16대를 사용하였고 사용자 요청을 생성하기 위하여 Pentium III 800Mhz, 128MB RAM의 시스템 4대를 사용하였다. 각각의 시스템은 100Mbps Ethernet 카드를 사용하고 있으며 100Mbps 스위칭 헙에 연결되어 있다. 클러스터링 시스템은 Linux 커널 2.4.16에 구현했으며 사용자 요청을 생성하기 위해서는 기존 연구에서 많이 활용되어 온 SURGE 웹 요청 생성 프로그램[23]을 사용하였다.

실험은 클러스터 노드를 4의 배수로 증가시키고 각각의 실험 환경에 대해 프로세스의 수는 80개로 고정시키

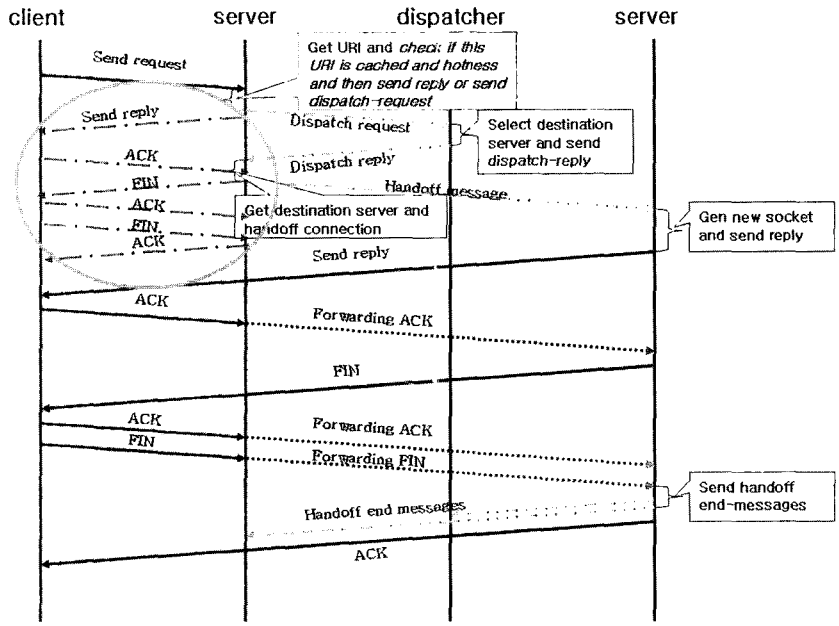


그림 13 변형된 패킷 흐름도

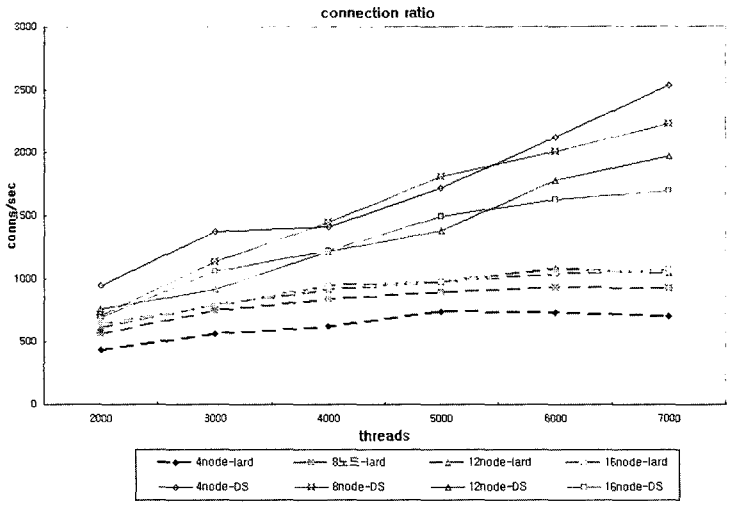


그림 14 DS 시스템과 LARD 시스템의 connection 비율

고 스레드(thread)의 수를 2000에서 7000까지 증가시키면서 수행하였다. SURGE의 스레드는 사용자의 접근 패턴에 따라 서버의 요청을 생성하여 전달하게 된다. SURGE에 의해 요청되는 파일의 수는 2000개로 한정시켰으며 이들은 약 50MB의 데이타를 구성한다. 파일의 크기는 최소 77바이트에서 최대 3,119,822바이트로 구성되었다. 파일은 세 가지 모델로 구성된다. Base 모델은 이미지를 포함한 파일이고 embed 모델은 이미지 자체를 가리키며, 마지막으로 loner 모델은 이미지를 가지지 않는 모델이다. SURGE 프로그램은 각 모델에 해당되는 요청을 이용하여 Zipf 분포에 맞는 웹 요청을 생성해 낸다. 각 환경에서의 실험은 5회씩 진행되었으며 이 중 최대값과 최소값을 제외한 나머지들의 평균값을 사용하였다. 또, DS 알고리즘에서 인자로 활용하는 웹 요청 빈도 임계값을 초당 200개까지의 요청을 수용할 수 도록 설정하였다.

5.1 처리율

첫 번째 결과는 각 알고리즘을 이용한 시스템의 처리

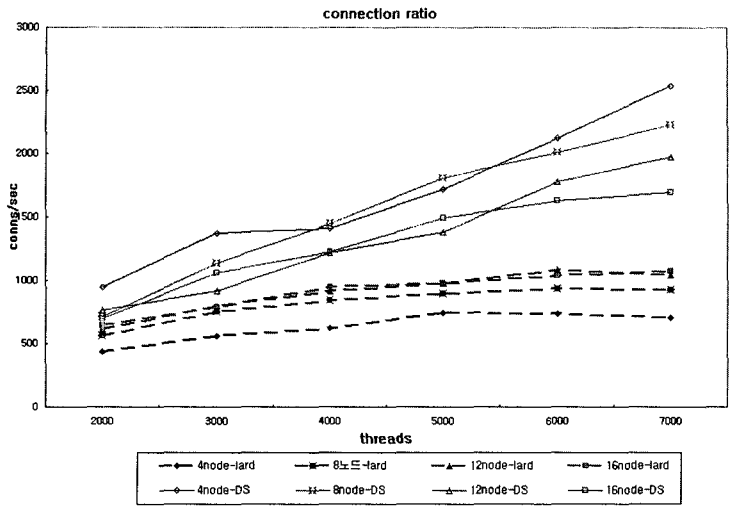


그림 15 DS 시스템과 LARD 시스템의 응답 시간

율을 보여주고 있다. 그림 14는 초당 커넥션 수를 관측한 결과이다. 실험 결과에서 x축은 스레드의 수를 표시하고 있다. 각 스레드는 하나의 사용자와 동일한 역할을 한다고 볼 수 있으며 여기서 보여지는 수치는 4대의 사용자 요청 생성 서버에서 생성된 스레드의 합이다. Y축은 초당 처리되는 커넥션 수를 의미한다.

이 실험의 결과를 다음과 같이 요약할 수 있다. 첫째, DS 알고리즘의 처리율이 LARD에 비해 높다는 것을 알 수 있다. 4노드의 경우 7000 스레드에서 4배 이상의 성능향상을 보여주고 있다. 실험에 의하면 전체적으로 약 35% 이상의 성능 향상을 보여줌을 알 수 있다. 둘째, LARD 실험 결과가 완만한 상승곡선을 이루고 있는 것에 비해 DS 알고리즘은 가파른 상승곡선을 이루고 있다. 셋째, LARD 실험에서는 노드 수가 증가함에 따라 처리율이 증가함에 비해 DS 알고리즘은 반대의 모습을 보여주고 있다. 클러스터 서버에서 서버 노드 수가 증가하게 되면 그 처리율도 같이 증가하는게 일반적이다. 그러나 이 실험에서 DS 알고리즘의 경우 그 반대의 결과를 보이고 있다. 이러한 결과를 보이는 첫번째 이유는 DS 알고리즘의 특성에서 찾을 수 있다. DS 알고리즘은 특정 웹 요청의 빈도가 증가함에 따라 시스템 자원을 적극적으로 할당하여 요청을 분산 처리한다. 따라서 서버에 할당되는 웹 요청의 수가 증가함에 따라 시스템 자원을 적극적으로 활용할 가능성이 높아진다. 본 실험에서는 같은 양의 요청이 각 서버로 나뉘어 전달되므로 적은 수의 서버를 가진 실험이 DS 알고리즘에 의해 시스템 자원을 보다 적극적으로 활용하게 되어 보다 높은 성능을 가져오고 있다고 설명할 수 있다. 이러한 결과를 보이는 두번째 이유는 실험의 한계에서 찾을 수 있다. 즉, 지금까지의 설명대로라면 더 높은 부하를 주면서 실험을 할 경우 다시 노드 수의 증가에 따라 성능이 뒤바뀌는 시점이 존재해야 할 것으로 예상한다. 그러나 본 결과에서는 그러한 시점을 보이지 못하고 있는데 이는 노드의 수나 스레드의 수를 더 이상 늘릴 수 없는 실험환경의 한계 때문이다. 본 실험에서 제시한 인자들에 따른 실험은 본 실험환경에서 안정적인 실험을 위한 최대치들이다. 이러한 한계를 극복하기 위해서는 새로운 실험환경의 구축이 필요하여 이 점에 대한 검증은 차후 연구 과제로 넘기기로 한다.

5.2 응답 시간

그림 15는 두 알고리즘의 응답 시간을 보여주고 있다. LARD 시스템의 경우 응답시간이 거의 균등하게 나타나는 반면에 DS 알고리즘을 적용한 시스템은 노드수에 따라 매우 큰 차이를 보여주고 있다. 4.2절에서 살펴 보았듯이 DS 알고리즘이 적용되어 부하 분산기를 거치지 않고 처리된 요청은 불필요한 패킷 교환과 핸드오프를 처리하는데 사용되는 시간을 줄이게 된다. 결국 DS 알고리즘은 각 요청의 빈도에 따라 5.1절에 살펴보았듯이 시스템 자원의 할당을 조절하게 되고 이를 통해 부가적으로 패킷의 처리 시간을 줄이는 효과를 얻게된다.

6. 결론 및 향후 추진과제

본 논문은 Aron등[1]이 FreeBSD에 구현한 시스템을 Linux에 구현하였고 사용자 요청의 특성을 반영한 Dual Scheduling(DS) 알고리즘을 제안하였다. DS 알고리즘은 각 요청의 접근 빈도와 확률이 상이하다는 점을 고려하여 요청의 빈도에 따라 자원을 할당할 수 있는 메커니즘을 LARD 알고리즘에 추가하였다.

차후 연구 과제로 사용자 요청의 부하 빈도를 판단하는 적절한 단위 시간을 결정하는 작업을 진행하고 있다. 단위 시간과 시스템 성능 관계, 그리고 각 사용자 요청의 워크로드와의 관계를 밝혀냄으로써 적절한 단위 시간을 결정할 수 있을 것으로 보여진다. 또한 현재 http 프로토콜 1.0만을 지원하는 시스템을 http 1.1의 persistent 커넥션을 지원하는 시스템으로 확장할 뿐만 아니라 동적 콘텐트 환경에서 부하를 효율적으로 분배하는 부하 분산 방법에 대한 연구를 진행하고자 한다.

참 고 문 헌

[1] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, "Scalable Content-aware Request Distribution in Cluster-based Network Servers," In Proceedings of the USENIX 2000 Annual Technical Conference, 2000.

[2] V.S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-aware Request Distribution in Cluster-based Network Servers," In Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII), 1998.

[3] T. Brisco, "DNS support for load balancing," RFC 1794, April, 1995.

[4] T.T. Kwan, R.E. McGrath, and D.A. Reed. "NCSA's World Wide Web server: Design and performance," IEEE Computer, vol. 28, no. 11, pp 68-74, Nov. 1995.

[5] R. Shah, High Performance Cluster Computing vol. 1, pp 340 363, Prentice Hall, New Jersey, 1999.

[6] K. Egevang and P. Francis, "The IP Network Address Translator (NAT)," RFC1631, May 1994.

[7] W. Simpson, "IP in IP Tunneling," RFC 1853, October, 1995.

[8] O.P. Damani, P.Y. Chung, Y. Huang, C. Kintala, and Y.M Wang, "ONE-IP: Techniques for Hosting a Service on a Cluster of Machines," Journal of Computer Networks and ISDN Systems, v.29, pp. 1019-1027, Sept. 1997.

[9] E. Anderson, D. Patterson, and E. Brewer, "The Magicrouter, an application of fast packet interposing," University of California, Berkeley, May, 1996.

[10] Cisco's LocalDirector (http://www.cisco.com/)

[11] Linux virtual server project (http://www.Linux-virtualserver.org/).

[12] W. Zhang, "Linux Virtual Servers for Scalable Network Services," Ottawa Linux Symposium, July 19-22, 2000.

[13] G.D. Hunt, G.S. Goldxzmidt, R.P. King, and R. Mukherjee. "Network Dispatcher: A Connection Router for Scalable Internet Services," In Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, Apr. 1998.

[14] A. Cohen, S. Rangarajan, and H. Slye, "On the Performance of TCP splicing for URL-aware Redirection," In Proceedings of the 2nd USENIX Symposium on Internet& Systems, 1999.

[15] C. S. Yang and M. Y. Luo, "Efficient Support for Content-based Routing in Web Server Cluster," In Proceedings of the 2nd USENIX Symposium on Internet Technologies & Systems, 1999.

[16] C. S. Yang and M. Y. Luo, "Design and Implementation of an Administration System for Distributed Web server," In Proceedings of the 12nd USENIX System Administration Conference, December, 1998.

[17] X. Zhang, M. Barrientos, B. Chen, and M. Seltzer, "HACC: An Architecture for Cluster-Based Web Server," In Proceedings of the 3rd USENIX Windows NT Symposium, 1999.

[18] M. Aron, P. Druschel, and W. Zwaenepoel, "Efficient Support for P-HTTP in Cluster-Based Web Servers," In Proceedings of the USENIX 1999 Annual Technical Conference, Monterey, CA, June 1999.

[19] Transmission Control Protocol (TCP), RFC 0793, September, 1981.

[20] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of WWW Client-based Traces," Technical Report BUGS95-010, April 1995.

[21] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," In the Proceedings of Infocom '99, April 1999.

[22] M. Arlitt and C. Williamson, "Web server workload characteristics: The search for invariants," In the Proceedings of the ACM SIGMETRICS '96, May 1996.

[23] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," In the Proceedings of the ACM SIGMETRICS '98, June 1998.

홍 일 구

2001년 홍익대학교 정보컴퓨터공학부 학사. 2003년 홍익대학교 컴퓨터공학과 석사. 2003년~현재 미국 IRVINE, CA 소재 XIMETA 근무. 관심분야는 인터넷 네트워킹 시스템, 임베디드 시스템, 스토리지 시스템

노 삼 혁

1986년 서울대학교 컴퓨터공학과(공학사). 1993년 메릴랜드대학교 컴퓨터과학과(박사). 1994년~현재 홍익대학교 정보컴퓨터공학부 부교수. 관심분야는 시스템 소프트웨어, 병렬처리 시스템, 실시간 시스템 등