2000년도 대한전기학회 하계학술대회 논문집 2000.7.17-20

클러스트링 웹서버에서의 QoS 구현에 관한 연구

박종규, 이도영, 장휘, 김학배
연세대학교 전기컴퓨터공학부

QoS Implementation on a Clustering Web Server

*Jonggyu Park, *Doyoung Lee, **Whie Chang, *Hagbae Kim
* Dept. Electrical & Computer Engineering, Yonsei University.
** C-EISA

Abstract - 본 연구에서는 리눅스 기반의 클러스터링 웹서버를 구성하고, 이 클러스터링 웹서버를 하나의 서버인 것처럼 관리하는 개발툴을 만들었다. 그리고 커널 패치를 통하여 로드밸런서가 다양한 시스템 정보를 밸런싱에 이용하도록 하였다. 각 리얼서버에는 응답 데이터의 양, 혹은 각 패킷의 길이에 따라 전송의 순서를 결정하는 QoS를 구현하였다.

1. 서 론

클러스터링 웹서버는 서버 내부의 네트웍을 구성하여 실제 서버에 걸리는 부하를 분산하고, 이보다 예측 가능하면서 최대의 수율 및 dependability를 보장하고, 성능과 고가용성을 높이는 방안으로 이용되어 지고 있다. 이때 클러스터링 웹서버는 단일서버로 동작하기 위하여 서로의 상태 및 내부 네트웍의 선로의 상태를 끊임없이 감시해야 한다. 이런 일련의 작업들은 TC(traffic control)와 성능 향상을 위해서, 웹서버의 특성상 사용자에게 양질의 서비스를 공급하는데 목적을 두고 있다.

현재 상품화되고 있는 리눅스를 이용한 클러스터링 웹서버들은 각각의 패키지를 하나로 묶어 단일서버를 구성하는 단계에 그치고 있는데, 이는 단순히 직관적인 기법에 의한 알고리즘으로만 설명되는 확장성 및 고가용성 보장만을 구현하고 있어, 해석적으로 검증된 구체적 알고리즘이 부족하다. 또한 단순히 PC에 리눅스용 가상서버(virtual server)용 응용프로그램을 활용하고 단순 구현한 상태일 뿐, H/W 모듈, 전체 시스템 아키텍쳐, 설치될 O/S 커널 및 하드디스크상의 프로그램 파일들, 관리용 프로그램에 대한 자체 솔루션 모델이 전무하고, 고가용성(HA) 보장을 위한 알고리즘 및 specification, 고속파일/데이터 전송을 위한 최적의 작업할당 및 스케줄링 기법의 부재, 그리고 구체적인 웹캐싱 또는 웹가속기 알고리즘 및 솔루션 부재 등을 기본적인 문제로 내포하고 있다.

본 연구에서는 기존의 리눅스용 클러스터링 웹서버의 상황을 알아보고, QoS(Quality of Service)를 높이기 위하여 클러스터링 웹서버 내부의 각 트래픽을 여러 집단으로 분류하고, Priority를 설정하여 Priority가 높은 패킷을 먼저 서비스 할 수 있는 리눅스 클러스터링 서버구조를 제안한다. 또한 전체적인 서비스를 질적으로 향상시키고 각 서버들이 공유자원을 균등히 분배하여 전체적인 서버성능 향상을 할 수 있는 서버의 구조를 구현하여 실제 클러스터링서버에서 각 리얼 서버의 상태를 정확하게 로드밸런싱의 정보로 이용하는 몇 가지 방법을 연구하였다.

2. 본 론

산업기술개발

2.1 서버의 구조

일반적으로 클러스터링 서버에서의 부하 분산 구조로는 NAT(Network address translation), IP Tunneling, Direct Routing 등 세 가지 방법이 쓰이고 있다. 각각 장단점이 있지만 같은 서브넷에서 구성되는 클러스터링 서버는 Direct Routing이 가장 효율적이다. 실제 본 연구에서도 Direct Routing을 이용해서 클러스터링 서버를 구성하였다.

2.1.1 Direct Routing

Direct Routing은 실제서버와 부하분산 서버에서 가상 IP 주소를 공유한다. 부하분산서버에서와 마찬가지로 가상 IP 주소를 설정한 인터페이스가 있어야하며, 이 인터페이스를 이용, 요청 패킷을 받아들이고 선택한 서버에 직접 라우팅 할 수 있다. 모든 실제 서버는 가상 IP주소로 설정한 non-arp alias 인터페이스가 있거나 가상 IP 주소로 향하는 패킷을 지역 소켓으로 재 지향한다. 그래서 실제 서버에서 패킷을 지역적으로 처리할 수 있다. 부하분산 서버와 실제 서버는 허브/스위치를 이용 물리적으로 링크된 그들만의 인터페이스를 가지고 있어야한다. 구조는 <그림 1>과 같다.

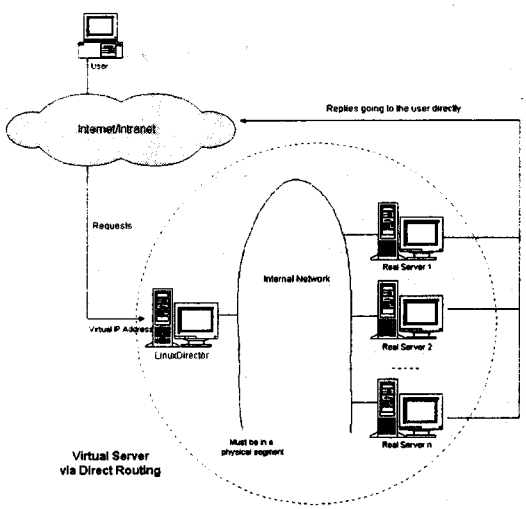

그림 1 Direct Routing 클러스트링 서버

사용자가 서버 클러스터에서 제공하는 서비스에 접근할 때, 가상 IP 주소(가상 서버의 IP 주소)로 향하는 요구 패킷이 부하분산서버로 간다. 부하분산서버(Linux Director)에서 패킷의 목적지 주소와 포트 번호를 검사

한다. 그 내용이 가상서버 서비스와 일치하면 스케쥴링 알고리즘에 따라 클러스터에서 실제서버를 선택하고, 접속을 기록하는 해쉬 테이블에 새로운 접속을 추가한다. 그러고 나서 부하분산서버에서 선택한 서버로 직접 패킷을 전송한다. 들어오는 패킷이 이러한 접속에 해당하고 해쉬 테이블에서 선택한 서버를 찾을 수 있으면 패킷은 다시 서버로 직접 전송된다. 서버에서 전송(포워딩)된 패킷을 받으면, 서버는 패킷에서 알리아스 인터페이스나 지역 소켓의 주소를 찾아서 요청을 처리하고 결과를 사용자에게 직접 전송한다. 접속이 해제되거나 시간을 초과하면, 해쉬 테이블에서 연결 기록을 제거한다. 작업의 흐름은 <그림 2>와 같다.

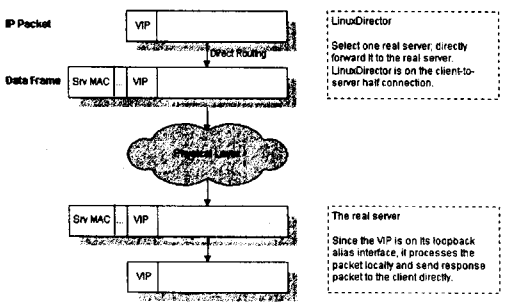

그림 2 Direct Routing 작업흐름

부하분산서버는 단순히 데이터 프레임의 MAC 주소만 선택한 서버로 바꾸며 이를 다시 LAN에 재전송한다. 이런 이유 때문에 부하분산 서버와 각 서버가 단일한 물리적 세그먼트안에서 연결되어야한다.

2.2 Scheduling 알고리즘

2.2.1 Round-Robin Scheduling

라운드-로빈 방식을 이용해 네트웍 연결을 서로 다른 서버에 연결하는 것을 말한다. 이 경우 실제서버의 연결갯수나 반응시간등은 고려를 하지 않는다. 그렇지만 약간의 차이가 있다. 라운드-로빈 DNS는 단일한 도메인을 서로 다른 IP로 해석을 하지만, 스케쥴링의 기초는 호스트 기반이며 캐싱때문에 알고리즘을 효율적으로 사용하기 힘들다. 그래서 실제 서버사이에 동적인 부하 불균형이 심각해 질 수 있다. 가상 서버의 스케쥴링 기초는 네트웍 기반이며 라운드로빈 DNS에 비해 훨씬 더 훌륭하다.

2.2.2 Weighted Round-Robin Scheduling

가중치기반 라운드-로빈 스케쥴링은 실제 서버에 서로 다른 처리 용량을 지정할 수 있다. 각 서버에 가중치를 부여할 수 있으며, 여기서 지정한 정수값을 통해 처리 용량을 정한다. 기본 가중치는 1이다. 예를 들어 실제 서버가 A,B,C 이고 각각의 가중치가 4,3,2 일 경우 스케쥴링 순서는 ABCABCABA 가 된다. 가중치가 있는 라운드 로빈 스케쥴링을 사용하면 실제 서버에서 네트웍 접속을 셀 필요가 없고 동적 스케쥴링 알고리즘보다 스케쥴링의 과부하가 적으므로 더 많은 실제 서버를 운영할 수 있다. 그러나 요청에 대한 부하가 매우 많을 경우 실제 서버사이에 동적인 부하 불균형 상태가 생길 수 있다.

2.2.3 Least-Connection Scheduling

최소 접속 스케쥴링은 가장 접속이 적은 서버로 직접 연결하는 방식을 말한다. 각 서버에서 동적으로 실제 접속한 숫자를 세어야하므로 동적인 스케쥴링 알고리즘중의 하나이다. 비슷한 성능의 서버로 구성된 가상 서버는 아주 큰 요구가 한 서버로만 집중되지 않기 때문에, 접속 부하가 매우 큰 경우에도 아주 효과적으로 분산을 한다. 가장 빠른 서버에서 더 많은 네트웍 접속을 처리할 수 있다. 그러므로 다양한 처리 용량을 지닌 서버로 구성했을 경우에도 훌륭하게 작동한다는 것을 한눈에 알 수 있을 것이다. 그렇지만 실제로는 TCP의 TIME_WAIT 상태 때문에 아주 좋은 성능을 낼 수는 없다. TCP의 TIME_WAIT는 보통 2분이다. 그런데 접속자가 아주 많은 웹사이트는 2분 동안에 몇 천 개의 접속을 처리해야 할 경우가 있다. 서버 A는 서버 B보다 처리용량이 두 배일 경우 서버 A는 수 천 개의 요청을 처리하고 TCP의 TIME_WAIT 상황에 직면하게 된다. 그렇지만 서버 B는 몇 천 개의 요청이 처리되기만을 기다리게 된다. 그래서 최소 접속 스케쥴링을 이용할 경우 다양한 처리용량을 지닌 서버로 구성되었을 경우 부하분산이 효율적으로 되지 못할 수 있는 것이다.

2.2.4 Weighted Least-Connection Scheduling

가중치 기반 최소 접속 스케쥴링은 최소 접속 스케쥴링의 한 부분으로서 각각의 실제 서버에 성능 가중치를 부여할 수 있다. 언제라도 가중치가 높은 서버에서 더 많은 요청을 받을 수 있다. 가상 서버의 관리자는 각각의 실제 서버에 가중치를 부여할 수 있다. 가중치의 비율인 실제 접속자수에 따라 네트웍 접속이 할당된다. 기본 가중치는 1이다.
가중치가 있는 최소 접속 스케쥴링은 다음과 같이 작동한다:
n개의 실제 서버가 있는 경우 각 서버 i는 가중치 Wi (i = 1, ... , n)를 가진다고 가정하자. 서버 i의 활동 접속(active connection)은 Ci (i = 1, ... , n)이고 모든_접속 은 Ci (i = 1, ... , n)의 합이다. 서버 j로 가는 넷트웍 접속은 아래와 같다.

$$(Cj/ALL\_CONNECTIONS)/Wj = \min \{ (Ci/ALL\_CONNECTIONS)/Wi \} \ (i=1,..,n)$$

이 비교에서 ALL_CONNECTIONS는 상수이므로 Ci를 모든 접속으로 나눠줄 필요가 없다. 그러면 다음과 같이 최적화 될 것이다.

$$Cj/Wj = \min \{ Ci/Wi \} \ (i=1,..,n)$$

가중치가 있는 최소접속 스케쥴링 알고리즘은 최소접속 스케쥴링 알고리즘에 비해 부가적인 배분작업이 필요하다. 서버들이 같은 처리용량을 가졌을 때는 작업할당의 간접비용을 최소화하기 위해 최소 접속 스케쥴링과 가중치가 있는 최소 접속 스케쥴링 알고리즘 둘 다 사용할 수 있다.

2.3 QoS 보장

서버의 성능은 실제 정보를 공급받는 사용자의 느낌에 의해서 평가될 수 있다. 이것은 정량적인 어떤 형태의 측정이 불가능한데, 소량의 정보를 제공받는 사용자는 미세한 응답시간을 요구하게 되고, 많은 정보를 제공받는 사용자는 충분히 기다려 줄 준비가 되어 있다.
<그림 3>에서 보여지는 것처럼 각각의 서비스는 데이터를 TCP단으로 넘김으로서 데이터는 각 단계를 거쳐 인터넷으로 전송되게 된다. 그럼으로 우리가 구현 하고자 하는 QoS는 이 패킷의 특성에 맞게 데이터의 전송을 제어하는 스케쥴링의 문제로 바뀌게 된다.
본 논문에서는 이를 위해서 리눅스의 커널을 패치 하여 각 데이터의 길이가 어떤 Variable한 값에 미치지 못하면 먼저 전송을 하는 간단한 규약을 제안하였다.
또한 이 QoS 전략이 바뀌어도 기본의 구조가 계속 유

지 될 수 있도록 Service Data에 Priority를 부여하고 스케줄링에 응용하는 리얼타임 스케줄링 개념을 도입하였다.

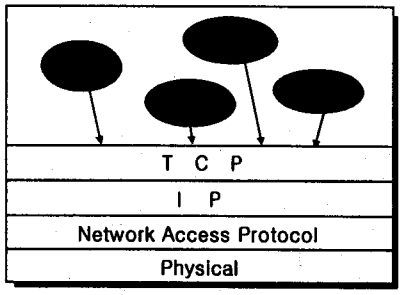

그림 3 Service Data의 전송

이런 일련의 작업들은 Linux의 커널레벨에서 일어나는 것으로 스케줄링 부분을 패치하여 QoS의 확장성을 보장하는 Linux커널로 재구성하였다.

2.4 클러스터링 서버 관리

클러스터링 서버에서 중요한 한 부분은 여러대의 Computer들이 모여서 단일서버처럼 동작 할 수 있도록 하는 관리프로그램이다. 이 관리프로그램은 각 서버의 상태에 따라서 균등한 로드밸런싱이 될 수 있도록 서버들의 정확한 상태를 알아내는 Mon 부분과 이 Mon에서 얻어지는 정보에 의해서 리얼서버를 지능적으로 등록하고 제거하는 작업과 로드밸런서의 상태를 파악하여 Fake를 수행하여 서버의 H/A를 보장하는 부분등으로 나뉘어 진다. 본 논문에서 Red-Hat의 피라냐를 기본 소스로 하여 각 리얼 서버의 선로상태와 리얼서버의 상태(메모리 사용량, CPU 사용률, 버퍼 사용률 등)를 전송받아 로드밸런싱의 정보로 사용하고 전체 서버를 GUI 환경에서 관리할 수 있는 서버관리 프로그램을 구성하였다.

3. 결 론

끊임없이 증가하는 서버의 부하를 해결하는 대안을 찾고자 하는 연구는 계속되고 있지만, 일반적으로 클러스터링 가상서버에 의한 방식이 가장 향상된 성능을 낼 수 있는 방법중의 하나이다. 클러스터링 서버는 클라이언트에게는 투명성을, 서버관리자에게는 고가용성과 확장성을 제공한다. 본 논문에서는 이 확장성과 고가용성의 클러스트링 서버의 접속자에게 최고의 만족을 주는 QoS를 보장하고 이를 통하여 서비스의 질을 향상시키는 방안을 제시한다. 또한, 현재 일반적으로 사용하고 있는 단순한 로드밸런싱 스케줄링 알고리즘에 비해 서버들에서 추출된 다양한 상태에 대한 정보를 이용하여 로드밸런서의 가중치 부하분배 스케줄링을 구현하였다.
그러나 사람의 만족도는 정량적이 어떤 크기로 나타내어지는 것이 아니라 각기 상황에 따라 서로 다른 관계로 모델링이 불가능하여 개선된 결과를 보여줄 수 없었다.
리얼서버들에서 얻은 많은 입력을 로드밸런스라는 출력으로 형성되는 시스템과 QoS를 위한 스케줄링 과정을 모델링을 하여 파라미터 동정과정을 통하여 성능을 개선시키고 개선의 결과를 정량화 할 것이다.

[참 고 문 헌]

[1] William Stallings, "High-Speed Networks : TCP/IP and ATM Design Principles", Prentice-Hall International, Inc.

[2] Michael Beck외, "Linux Kernel Internals", Addison-Wesley, Second Edition,

[3] V. Cardellinu, M. Colajanni and P. Yu, "Redirection Algorithms for Load Sharing in Distributed Web-Server Systems", *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*, pp.528-535, 1999.

[4] A. Mourad and H. Liu, "Scalable Web Server Architectures", *Computers and Communications, 1997. Proceedings., Second IEEE Symposium on*, pp.12-16, 1997.

[5] B. Narendran, S. Rangarajan and S. Yajnik, "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access", *Reliable Distributed Systems, 1997.* pp.97-106, 1997.

[6] http://www.linux-vs.org

[7] M. Dias, W. Kish, R. Mukherjee and R. Tewari, " A scalable and highly available web server," *IEEE Proceedings of COMPCON '96*, pp.85-92, 1996.

[8] N. Budhiraja, K. Marzullo, "High-available services using the primary-backup approach," management of replicated data, pp.47-50, 1992.

[9] K. Shin and X. Cui, "Effects of computing time delay on real-time control systems," *Proc. of 1988 ACC*, pp.1071-1076, 1988.