TCP/IP 부하 분산을 위한 동적 스케줄러의 설계

이정석[0]*, 신용욱*, 변태영**, 이선우*, 한기준*
경북대학교 컴퓨터공학과

Design of Dynamic Scheduler for TCP/IP Load-Balancing

Jeong-suk Lee[0]*, Yong-Wook Shin*, Tae-Young Byun**, Sun-Woo Lee*, Ki-Jun Han*

* Dept. of Computer Engineering, Kyungpook National University

longlong@netopia.ce.knu.ac.kr , ywshin@comeng.ce.knu.ac.kr , sunwlee@netlab.ce.knu.ac.kr

kjhan@bh.knu.ac.kr

** School of Computer & Electronic Engineering, Kyungju University

tybyun@comeng.ce.knu.ac.kr

요 약

본 논문에서는 3-way 디스패치(dispatch) 기반의 대표적인 기법들인 Resonate 사의 Central Dispatch 와 IBM사의 Network Dispatcher 에 대해 비교, 분석하고 보다 나은 부하 분산 기법을 구현하기 위해 두 가지 기법의 장점들을 선택하여 다양한 트래픽 상태에 대해 부하 균등(Load-Balancing)과 부하 분산(Load-Sharing)기법을 동적으로 선택하여 동작할 수 있는 부하 분산 스케줄러를 설계한다.

1. 서론

최근 몇 년간 급속한 인터넷 혹은 인트라넷의 증가로 인해 많은 사이트들이 사용자들에게 서비스 해줄 수 없는 상황이 종종 발생한다. 이것은 전자 상거래, 인터넷을 통한 최신 뉴스, 유행에 따른 정보 등을 얻기 위해 사용자들의 서비스 수요가 급증하고 있기 때문이다.

이러한 수요의 급증은 일반적으로 특정한 웹사이트로 집중되는 경향이 있다. 만약 웹사이트에서 과부하(Over-load)가 발생하여 사용자의 대기 시간이 길어지거나, 서비스 요구에 대한 응답을 수행해 주지 못한다면, 결국 사용자는 그 사이트를 회피하는 경향이 있다. 이를 방지하기 위해 TCP/IP 기반 서비스를 제공해 주고 있는 기업들은 부하 분산 기법의 도입을 선택할 필요가 있다. [1][4][5]

일반적으로 사용자의 서비스 요구가 많은 사이트들은 서버를 여러 대를 두어서 서비스를 제공해 주고 있다. 이러한 환경에서 확장성(scalability)이 있고, 각 서버의 효율적인 사용을 위해 다양한 트래픽 상황에 대처할 수 있는 동적인 부하 분산 기법의 필요성이 제기된다.

2. 기존 연구 사례

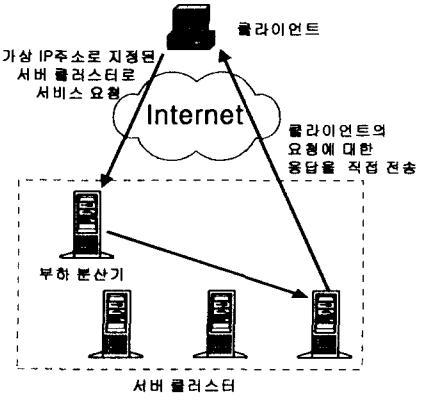
2.1. 라운드-로빈(Round-Robin) DNS

라운드-로빈 DNS는 고전적인 부하 분산 기법의 대표적인 방법으로서 특정한 서비스에 대한 서버들의 IP 주소 리스트를 도메인 네임 서버에 미리 정의해 놓고 서비스 요구가 수신될 때 마다 서버들이 순서대로 서비스해준다.

이 방법은 서버의 주소를 클라이언트가 캐싱 하기 때문에 일정 기간동안 같은 서버로 요청을 하게 되므로 동적으로 변하는 망 상황에서 대처하기가 힘들다.

2.2. 디스패치 기반의 부하 분산 기법

디스패치 기반 기법은 앞에서 살펴본 고전적인 접근법에서 나타난 한계점들을 극복하기 위해 제안된 기법으로 복수개의 서버에 하나의 IP주소를 이용하여 접근할 수 있도록 하는 가상 IP 주소(VIPA) 개념과 사용자 요구 트래픽과 서버의 응답 트래픽의 비대칭적인 특성[2]을 이용하는



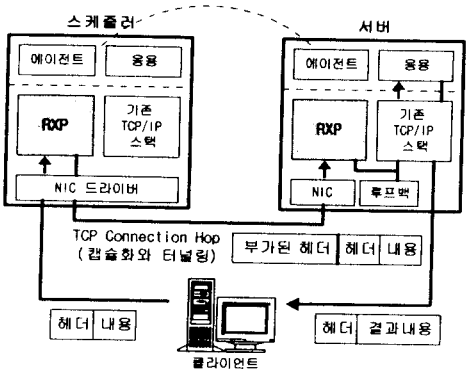
[그림 1] 디스패치 기반 부하 분산 기법

상각 구도의 연결구조를 지원한다. 이 기법은 항상 스케줄러가 각 서버의 현재 상태를 파악하고 있기 때문에 라운드-로빈 DNS가 가지는 단점을 해결할 수 있다는 장점을 가진다. 그림 1은 디스패치 기반 부하 분산 기법의 개요를 보여주고 있다.

2.2.1. Resonate Central Dispatch

Central Dispatch는 패킷의 URL을 분석하여 신뢰성 있는 서비스를 클라이언트에게 제공해 준다. 그림 2는 Central Dispatch 의 동작을 보여준다. RXP(Resonate Exchange Protocol)는 실질적인 스케줄링과 수신되는 트래픽 관리를 담당하고 있으며, 에이전트는 현재 서버의 상태정보(CPU의 부하, 가용 한 자원 정보, 망 지연시간 등)를 파악하여 주 부하 분산기와 다른 서버들의 에이전트들에게 제공해 주며, 한 사이트에서 각 호스트들을 유일하게 나타낸다. 메인 스케줄러는 가상 IP주소로 지정되어 있다.

Central Dispatch는 URL을 분석함으로써 지속적인 트래픽이 있을 때 서버들의 부하를 동일하게(equalizing) 하는 부하균등(Load-Balancing)방법을 사용하고 있다.



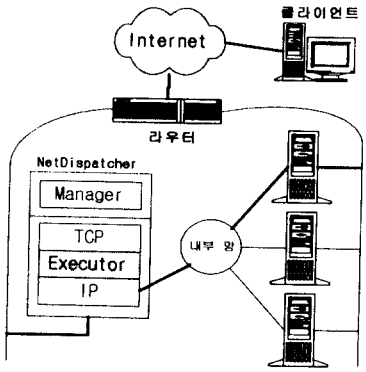
[그림 2] Central Dispatch의 동작

2.2.2. IBM Network Dispatcher

Network Dispatcher는 포트를 기준으로 서비스를 분류하여 해당 서버 클러스터로 패킷을 포워딩 한다. 새로운 연결 요청이 수신될 때 적합한 서버 클러스터를 선택하기 위해 CPU의 작업부하에 대해 반비례하도록 연결을 할당하는 정성적 분배 기법인 부하 분산(Load-Sharing)방법을 사용한다.

이 기법은 사용자의 요구가 다량으로 발생하는 최대 트래픽 집중 기간이 되면 일부 서버로 편향되는 부하를 부드럽게(smoothing) 해서 효율적으로 동작하는 것을 보여준다.

Network Dispatcher는 각 패킷의 헤더를 읽어서 포워딩과 새로운 연결할당을 담당하는 Executor와 사용자-레벨상에서 동작하는 응용 프로그램인 Manager로 구현이 된다. 그림 3은 Network Dispatcher의 일반적 구조이다.



[그림 3] Network Dispatcher의 일반적인 구조

3. 부하 분산 스케줄러의 설계

3.1. 기존 연구의 한계점

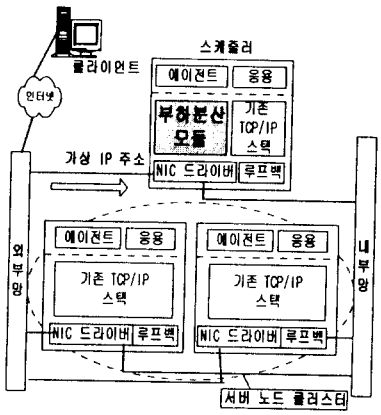
앞에서 살펴본 두 가지 디스패치 기반 기법들은 많은 이점들을 지니고 있지만, 자원의 소비량과 비용면에서 대조적인 측면을 가진다. Central Dispatch는 패킷을 서버로 전송할 때 헤더 변환 작업이 필요하고, 서버 노드에서도 패킷에 대한 처리를 위해 커널-레벨에 모듈을 추가 시켜야 하며, 서비스 요구의 급속한 증가가 있을 때에는 URL 분석에 많은 오버헤드가 발생하기 때문에 효율적으로 대처하지 못하는 경향이 있다. Network Dispatcher는 기존의 TCP/IP 스택을 변경하기 위해 복잡한 구현과정을 필요로 하게 된다.

이런 문제점들을 해결하기 위해서는 트래픽 상태의 변화에 따라 동적인 부하 분산이 가능하도록 해야 한다.

3.2. 스케줄러의 구조 정의

본 논문에서 제안하고 있는 기법은 서버에 대한 추가적인 모듈 설치를 최대한 지양하면서 부하 균등 방법을 응용한 디스패치 기반의 기법으로 부하 분산 스케줄러를 설계한다.

제안된 모델에서는 서버의 추가 모듈과 불필요한 헤더 변환 과정을 없애기 위해, 망의 구성을 외부 망(External Network)과 내부 망(Internal Network)으로 분리하여 부하 분산을 한다.



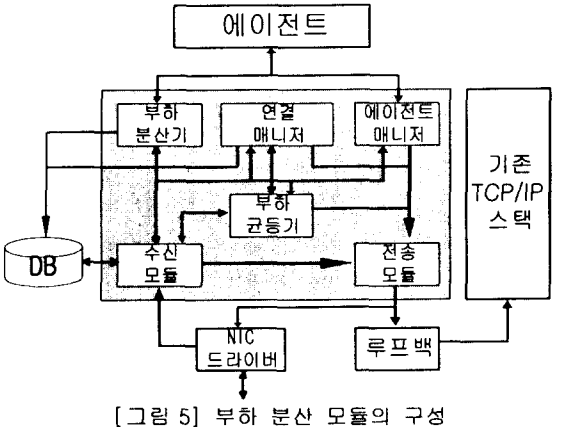
[그림 4] 제안된 망의 구성도

그림 4는 망의 구성에 대한 간략한 설계를 보여 준다. 만약 클라이언트가 가상 IP 주소로 지정된 스케줄러에게 서비스 요청을 시도한다면 스케줄러의 부하 분산 모듈은 디바이스 드라이버로부터 수신되는 패킷들을 가지고 온다. 부하 분산 모듈 내부에서 구동 되는 하위 모듈들은 패킷의 내용을 분석하고 서비스의 종류를 분류하며 에이전트에 의해 수집된 서버들의 상태정보를 바탕으로 서버를 선택하여 패킷을 포워딩 시킨다. 이 때 패킷은 내부 망을 따라 서버에게 전달이 되기 때문에 부가적인 헤더 변환 작업이 필요가 없고, 단지 목적지 서버의 MAC주소만을 알고 있으면 된다. 패킷을 전달 받은 서버는 요청에 대한 결과를 스케줄러를 거치지 않고 바로 외부 망을 통해 클라이언트로 전송하게 된다. 이러한 동작과정을 수행하는 데에는 다음과 같은 모듈들이 필요하다.

- 수신 모듈 : NIC 드라이버에서 수신되는 패킷들을 받아들여 트래픽의 특징에 따라 해당 모듈로 전송한다. 이 모듈은 패킷 도착 시간들의 간격을 기록하는 타이머를 사용하여 동적인 트래픽 변화에 대응할 수 있도록 부하 분산 과정을 조절한다
- 부하 균등기 모듈 : 수신모듈에서 받은 패킷 내용의 URL을 분석한다. 요청 서비스에 해당하는 서버들의 상태정보를 에이전트 매니저 모듈로부터 받아 적합한 서버를 선택하는 기능을 가진다.
- 연결 매니저 모듈 : 부하 분산기 모듈에서 선택된 서버와 TCP 연결을 설정하고 그 정보를 테이블 형태로 저장한다.

- 부하 분산기 모듈 : 수신 모듈로부터 전달 받은 패킷들의 헤더에서 포트 번호만으로 해당 서비스를 결정하고, 에이전트로부터 서버들의 상태정보를 받아 가장 적합한 서버를 선택한다.
- 전송 모듈 : 포워딩 할 패킷을 하위 물리계층으로 전송하는 담당을 한다
- 에이전트 매니저 모듈 : 각 서버와 스케줄러의 응용계층에서 동작하고 있는 에이전트로부터 상태정보를 동적 모니터링을 통해 수집하여 부하 균등기 모듈과 부하 분산기 모듈에게 제공해 준다.

그림 5는 부하 분산 모듈을 구성하는 하위 모듈들의 구조이다.



[그림 5] 부하 분산 모듈의 구성

정상적인 동작을 수행하는 부하 분산 모듈은 클라이언트에게 신뢰성 있는 서비스를 제공해 주기 위해 수신되는 패킷들을 부하 균등기로 보내어 URL을 분석하고 서버를 할당한다. 하지만, 서비스 요구 트래픽이 갑자기 폭주하는 상황을 고려해 보자. 그렇다면, 어느 정도는 서비스 제공이 가능하겠지만 이런 상황이 지속된다면 스케줄러에서는 URL 분석에 대한 오버헤드로 인해 서비스 결과가 클라이언트까지 도달하는데 지연이 발생하고 심할 경우에는 병목현상이 발생하여 한 동안 서비스 제공이 불가능해질 수도 있다.

본 논문에서 제안된 모델은 이러한 문제점을 해결하기 위하여 부하의 분배과정을 트래픽의 특성에 따라 동적으로 수행한다. 일반적으로 트래픽이 스케줄러로 들어 올 때에는 수신 모듈에서 패킷을 부하 균등기 모듈로 전달하여 각 서버의 트래픽 상태를 균등하게 유지한다. 하지만, 수신 모듈에서 클라이언트로부터 수신되는 패킷들의 도착 시간 간격이 일정한 한계값(threshold)을 초과하여 조밀해질 경우에는 폭주 기간으로 판단하고 URL분석과 같은 복잡한 부하 균등 과정을 수행하지 않도록 하기 위해 부하 분산기 모듈로 패킷을 전달한다. 패킷을 받은 부하 분산기 모듈은 패킷의 헤더 부분에 있는 목적지 포트 번호를 이용해서 해당 서비스 서버 클러스터를 결정하고, 에이전트 매니저와 연결 매니저에 의해 제공된 정보를 참조하여 목적지 서버의 위치를 전송 모듈에게 알려 준다. 전송 모듈에서는 단지 목적지 서버의 위치 정보를 이용하여 포워딩만을 수행한다.

수신 모듈은 패킷 도착 시간을 조사해서 트래픽의 양이 줄어 들면 폭주 기간이 종료되었다고 판단하여 패킷을 다시 부하 균등기 모듈로 보내어 부하 분산 과정을 정상적인 트래픽 상태에 맞게 변화 시킨다.

이와 같이 동작하는 동적 부하 분산 스케줄러를 설계함으로써 얻을 수 있는 이점은 다음과 같다.

- 서버에 추가 모듈이 필요 없으므로 확장성을 제공해줄 수 있다.
- 스케줄러는 단지 커널-레벨에 모듈들을 추가만 시키면 되므로 기존 TCP/IP 스택의 수정이 필요 없다.
- 트래픽의 상태 변화에 따라 동적으로 부하 분산 기법의 선택이 가능하므로 사용자에게 신뢰성 있는 서비스를 제공해줄 수 있다.

실제적으로 서비스를 제공하는 서버들은 단지 응용-레벨 상의 프로그램인 에이전트를 통해 스케줄러에게 정보를 제공해줄 수 있다.

4. 결론 및 향후 연구 과제

앞에서도 언급한 바와 같이 본 논문의 주된 목적은 보다 효율적인 동적 부하 분산 스케줄러를 구현하는 것이다. 그러기 위해 기존에 제안되었던 부하 분산 기법들을 소개하고 장단점을 파악하여 각 기법들의 수행과정에서 나타날 수 있는 여러 가지 한계점을 살펴 보았다.

본 논문에서 제안된 부하 분산 기법은 서버에 추가 모듈 설치를 최대한 배제함으로써 확장성(scalability) 있는 서버의 운용을 기대할 수 있고, 이에 따르는 추가 비용도 감소될 수 있다. 또한, 동적인 트래픽 상태에 대해 부하 분산 기법과 부하 균등 기법을 적절하게 사용함으로써 클라이언트에게 신뢰성 있는 서비스를 제공해줄 수 있다.

5. 참고 문헌

[1] Andrew Reback, " Resonate Central Dispatch : Powerful Distributed Traffic Management for IP - Based Networks " , Resonate Technical Overview, Jan. 1999

[2] Glen Kosaka, " Resonate Central Dispatch TCP Connection Hop " , Resonate Technical Overview, Jan. 1999

[3] Resonate Central Dispatch Application Note, " Scaling Application Performance on Servers Using Port Load Balancing" , Jan. 1999

[4] G. Goldszmidt, G. Hunt, " NetDispatcher : A TCP Connection Router " , IBM Research Report, May. 1997

[5] Chris Gage, " Scaleability, Availability and Load - balancing for TCP/IP applications " , IBM Research White Paper, April 1999

[6] Valeria Cardellini, Michele Colajanni, Philip S. Yu, " Dynamic Load Balancing on Web -Server Systems" , *IEEE Internet Computing*, May.June 1999