확장성있는 서버 시스템을 위한 부하 균등화 서비스에 대한 연구

이동훈* , 한영태* , 민덕기*
*건국대학교 컴퓨터 • 정보통신공학과
{dhlee, ythan, dkmin}@cse.konkuk.ac.kr

A Study on Load Balancing Service for Scalable Server Systems

Donghoon Lee*, Youngtae Han*, Dugki Min*
*Dept. of Computer Science and Engineering, Konkuk University

요 약

본 논문은 분산 서버 시스템을 위한 Load Balancing Architecture를 제시한다. 이 구조는 기존에 구현된 클라이언트/서버 시스템을 그대로 사용하면서 규모를 확장할 수 있는 특징을 가지고 있다. Load Balancing Service가 동작하기 위해서는 서버의 성능과 구성상태에 대한 정보를 주고받는 것이 필요하다. 이를 위하여 본 논문에서는 Load Balancing Information Transfer Protocol을 제시한다. 본 논문이 제시하는 구조는 DCOM 및 EJB 등의 Component 서버에서도 사용되어서 확장성을 가지게 할 수 있다.

1. 서론

요즈음 인터넷은 모든 사람에게 있어서 중요한 수단 및 방법이 되고 있다. 대중매체에서도 "인터넷 서바이벌"같은 내용을 방송할 정도로 생활 속에 깊숙이 들어와 있다. 이 정도로 대중화되면서 그전에는 없던 문제가 발생하기 시작하였다. 바로 서버의 용량문제인데 사람들이 많이 찾는 사이트인 경우에 시스템이 느려지는 문제가 많이 발생한다. 일반적으로 사이트가 처음에 구성될 때 얼만큼의 사람들이 접속할지 예측하기는 힘들다. 그래서 보통 적당히 적은 예산을 가지고 서버를 구축하게 된다. 하지만 이렇게 구축해놓은 서버는 나중에 많은 사람들이 몰릴 경우에는 문제가 발생하게 된다. 여기서 필요한 것은 클라이언트의 서비스 요구가 적을 때 구축해 놓은 서버가 나중에 클라이언트의 요구가 많아질 때도 별 어려움 없이 확장이 쉬운 것이다. 즉, 확장성이 좋은 서버로 구축을 해놓게 되면 나중에 규모가 변할 때 쉽게 대응을 할 수 있다.

본 논문은 분산 서버 시스템을 위한 Load Balancing Architecture를 제시한다. 이 구조는 기존에 구현된 클라이언트/서버 시스템을 그대로 사용하여 규모를 확장할 수 있다. Load Balancing Service가 제대로 동작하려면 성능과 서버의 구성상태에 대한 정보를 주고받는 것이 필요하다. 이를 위해서 본 논문에서는 Load Balancing Information Transfer Protocol을 제시한다. 기존 서버는 이 Protocol에 대해서 알지 못하기 때문에 Wrapper를 사용하게된다. 또한 이 프로토콜을 사용하면 클라이언트가 연결된 서버가 결정된 후에라도 클라이언트를 다른 서버로 바꾸어 연결할 수 있다 (Client Rescheduling). 이 구조는 Enterprise Java Beans나 Web Application Server에서 사용하여 좀더 확장성 있고 내고장성 (Fault Tolerance)있는 시스템이 되도록 지원할 수 있다.

다음절에서는 Load Balancing에 대한 관련 연구들을 살펴본다. 3절에서는 Load Balancing에서 고려해야 할 점을 설명한다. 4절에서는 본 논문에서 제시한 Load Balancing Architecture를 설명한다. 5절에서는 Load Balancing Information Transfer Protocol를 설명하고 마지막 6절에서 결론을 내린다.

2. 관련 연구

2.1. Layer 4 Switch

현재 여러 회사에서 Load Balancing Switch라는 이름으로 제품을 만들고 있다[8][9]. 이는 서버 여러 개를 하나의 스위치에 연결해서 같은 IP주소로 접근하도록 하고 스위치에서 여러 개의 서버중의 하나로 연결을 다시 맺어주는 형태이다. 이러한 스위치는 하드웨어적으로 처리하기 때문에 빠르다는 장점이 있으나 아직까지 성능을 판단하는 알고리즘이 부족하다. 주로 사용되는 알고리즘은 Round Robin이나 Random방식으로서 서버의 성능은 그다지 고려하지 않는다. 그리고 클라이언트의 Allocation이 끝나면 그 클라이언트는 다른 서버로 바뀌는 경우를 Rescheduling이라고 하는데 이는 고려하지 않는다.

2.2. Microsoft Load Balancing Service

Microsoft의 Load Balancing은 두 가지가 있는데 첫 번째는 Network Load Balancing Service이다[10]. 이것은 클라이언트의 연결을 여러 대의 서버중의 하나로 할당하는 방식이다. 그러나 여기에서는 Load Balancing Switch에서와 마찬가지로 Rescheduling을 고려하지 않는다.

두 번째는 DCOM에서의 Load Balancing이다[11]. 이것은 Component를 생성할 때 어떤 Component Server에 생성할 것인지를 결정하게 된다. 여기에서도 마찬가지로 Rescheduling은 고려하지 않는다.

2.3. Web Application Server

최근에 많이 나오는 Web Application Server들은 Dispatcher라

는 형태로 Load Balancing을 지원한다. Dispatcher가 클라이언트를 Application Server로 할당한다. 여기에서도 보통은 Rescheduling을 지원하지 않는다. 하지만 여기에 본 논문에서 제안하는 Load Balancing Architecture를 적용하면 Web Application Server도 쉽게 확장성이 좋아진다[5][6][7].

3. Load Balancing Service의 고려사항

최근에는 대용량의 서버 시스템을 구현하기 위해서 여러 대의 서버를 사용하는 경우가 많아졌다. 기존의 커다란 서버 하나를 사용하는 경우에는 Load Balancing이라는 기술이 필요 없었으나 여러 개의 서버를 사용하는 서버 시스템이 생겨나면서 서버처리의 병목현상이 발생하게 되었다. 하나의 서버에 많은 양의 작업이 대기하고 있는데 반해 다른 서버는 할당된 작업이 없어서 비어있는 상태가 된다. 이러한 상태는 전체적인 시스템의 성능이 저하되는 결과를 야기한다. 이때 많은 양의 작업이 대기하고 있는 서버에서 다른 서버로 작업을 넘겨서 처리할 수 있다면 시스템의 성능을 향상시킬 수 있다.

Load Balancing은 기본적으로 여러 개의 컴퓨터 사이에서 작업을 분배하는 기능을 말한다. 여기서 작업을 분배받아서 수행할 수 있는 부분은 서버가 된다. 서버가 하나의 작업을 받아서 수행 중에 다른 서버로 이 작업을 옮기는 작업을 가리켜서 Migration이라고 한다. Migration은 앞에서 언급한 Rescheduling과 실제로 데이터를 옮기는 두 과정으로 이루어져 있다. 이러한 Load Balancing Service는 아래와 같은 고려사항들을 생각해서 구현할 수 있다.

3.1. Dynamic Balancing vs. Static Allocation

Dynamic Load Balancing은 작업이 수행 중에 다른 서버로 Migration이 가능한 것이다. 서버들 사이에서 부하의 차이가 심하다면 옮겨서 전체적인 성능을 높이는 것이 좋다. 하지만 부하를 옮기는 것은 다른 부하를 요구하게 되므로 옮기는데 너무 비용이 많이 든다면 바람직하지 않다. Static Allocation은 처음에 작업을 한번만 할당을 하고 작업이 이미 시작됐다면 그 작업은 끝날 때까지 하나의 서버에서 처리하는 것이다.

본 논문에서 제안하는 구조는 Migration - 즉, Dynamic Balancing - 은 지원하지 않는다. 다만 Client Rescheduling은 지원한다. Data를 옮기는 Migration을 예를 들면 전체적으로 하나의 세계를 구성하고 그 안에서 자유로운 Interaction을 처리하는 게임 같은 구조가 있다[1].

3.2. Transactional vs. Continuous

만약 수행시간이 짧다면 Dynamic Load Balancing을 적용해서 작업을 Migration시킬 필요가 없다. 하지만 Continuous하게 긴 시간을 필요로 한다면 한가한 다른 서버로 옮겨서 전체적인 성능을 높이는 것이 유리할 수 있다. 수행시간이 짧은 Transactional한 작업일 경우에 Session을 유지해야 하는 것이 필요할 수도 있다. 즉, 첫 번째 접속했었던 서버로 두 번째도 접속을 해야할 필요가 있을 수 있다. 이러한 기능은 접속했었던 클라이언트들의 목록을 유지하고 Timeout을 설정하는 형식으로 해결할 수 있다. 본 논문에서 제안하는 구조는 두 가지 경우를 모두 고려하였다.

3.3. Single Session vs. Multi Session

여기서 설명하는 Session은 Transaction 작업의 Session과 의미가 틀리다. 여기의 Session은 여러 Client사이에서 Interaction이 존재하는 범위라고 볼 수 있다. 즉, 대화를 지원하는 서버의 경우에 하나의 대화방에 연결되어 있는 클라이언트들끼리는 대화를 주고받을 수 있어야 한다.

이 Session이 전 시스템에 걸쳐서 모든 사람과의 Interaction이 가능한 경우도 존재할 수 있다. 예를 들면 게임 같은 경우이다[1]. 이를 본 논문의 구조에서 지원하려면 각 서버사이에서 다른 통신 방법을 사용해서 내용을 주고받아야 한다.

3.4. Load Information

Load Information은 서버의 상태를 판단하기 위한 자료이다. 서버의 부하가 높다면 다른 서버를 선택하여야 한다. 이러한 정보는 CPU사용량이나 메모리의 사용량 등의 시스템 정보를 사용할 수도 있고 어플리케이션에서 정보를 제공할 수도 있다. 이때는 본 Framework에서 정한 표준적인 형태로 제공되어야 한다. 이 정보는 Load Balancing을 수행하는 데에 사용된다.

4. Load Balancing Architecture

본 논문의 목적은 기존에 사용하던 서버를 별다른 변경 없이 컴퓨터만을 추가하여 서비스를 확장할 수 있도록 하는 시스템을 구현할 수 있도록 하는 것이다. 이 점은 기존에 사용하던 서버의 재사용성을 높인다는 측면에서 매우 중요하다. 본 논문에서 제시하는 Architecture는 앞에서 설명한 Load Balancing에 대한 고려사항을 반영하고 기존의 서버를 그대로 사용하면서 확장이 가능하도록 설계되었다.

그림1은 Load Balancing Service를 제공하기 위해 본 논문에서 제시한 구조이다. 이 구조는 크게 Concentrator와 Selector가 모여있는 Concentrator Cluster와 Server를 관리하는 Sever Pool과 Wrapper, Server로 구성되어 있는 Server Cluster로 나눌 수 있다. 여기에서 클라이언트와 서버는 이미 구축되어 있는 소프트웨어를 사용할 수 있다.

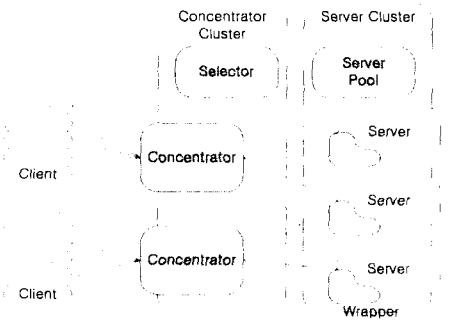

그림 1 - Load Balancing Architecture

▶ Concentrator

Concentrator는 클라이언트에서 오는 연결을 받아서 부하가 적은 서버로 연결시켜 준다. 이 작업을 하려면 어떤 서버가 어떤 포트로 서비스하고 있는지에 대한 정보를 알아야 하고 여러 서버들의 상태가 어떤지 알아야 한다. 이 정보는 Selector를 통해서 얻어온다. Concentrator가 존재하는 가장 큰 이유는 Client Rescheduling 때문이다. 클라이언트는 실제로 Concentrator와 연결되어 있기 때문에 서버가 바뀌어도 그 사실을 알 수 없다.

▶ Selector

Selector는 Concentrator Cluster와 서버의 포트 사용 상태를 관리한다. Concentrator들의 상태를 관리하고 서버의 정보가 변할 경우 Concentrator들에게 알려준다. 클라이언트로부터 Concentrator 선택 요청이 들어오면 적당한 Concentrator를 클라이언트에게 알려준다. Selector는 서버에 대한 정보를 서버로부터

직접 받거나 서버의 밖의 Wrapper에서 받는다. Selector는 정책에 따라서 정보를 적극적으로 수집할 수도 있고 서버 쪽에서 정보를 Push할 수도 있다.

▶ Wrapper

Load Balancing 서비스를 사용하기 위해서 서버는 Concentrator나 Selector에게 여러 가지 정보를 주어야 한다. 그러나 기존의 서버는 Concentrator와 Selector의 존재를 모르고 구현되어 있기 때문에 그러한 기능을 대신해줄 부분이 필요하다. Wrapper는 기존에 구현되어 있는 서버를 대신해서 Selector와 통신하는 기능을 가지고 있다. 통신할 때는 Load Balancing Information Transfer Protocol을 사용해서 정보를 주고받는다.

▶ Server Pool

System의 부하에 좀더 Dynamic하게 반응하는 방법은 성능이 떨어질 때 자동으로 서버가 추가되어 서비스를 하고 필요 없어지면 자동으로 제거되는 것이다. 이를 위해서 Server Pool을 유지하고 있다가 필요할 때 System에 자동으로 추가하는 방법을 사용할 수 있다. 이 Server Pool은 서버들을 관리하면서 Selector를 통해서 전체 System의 성능을 감시하고 있다가 판단을 내리게 된다.

5. Load Balancing Information Transfer Protocol

앞에서 설명한 것처럼 서버를 대신해서 Selector와 통신하는 Wrapper나 Wrapper의 기능을 포함하게되는 새로운 서버는 Selector와 통신하는 Protocol이 필요하다. Protocol에서 필요로 하는 기능은 아래와 같이 4가지로 볼 수 있다.

▶ Registration / Unregistration

서버가 Selector에게 자신이 서비스를 제공한다고 알리거나 서비스를 중단한다고 알린다. Selector는 이 정보를 사용해서 클라이언트로부터의 접속을 이 서버로 할당할지를 결정한다.

▶ Performance Information

서버가 Selector에게 자신의 성능 정보를 보낸다. 이 정보는 Selector를 통해서 Concentrator로 전달된다. Concentrator는 이 정보를 사용해서 서버의 상태를 판단해서 클라이언트를 할당할지 결정할 수 있다.

① System Information
운영체제의 Performance Information을 얻어낼 수 있는 방법을 사용해서 시스템의 성능 정보를 얻어낸다. 수치의 표준화가 잘 되어 있고 일반적으로 상태를 잘 반영한다. 하지만 운영체제마다 사용방법이 틀리다. 예를 들면 CPU 활용률, 남은 메모리 양등이 될 수 있다.

② Application-Provided Information
서버에서 자체적으로 자신의 상태를 보낸다. 시스템의 성능정보보다 이 정보가 좀더 민감한 정보가 될 수 있다. 하지만 시스템의 정보보다 비현실적일 수 있다. 이 정보를 예를 들면 세션관리서버에서 세션의 수같은 것이 정보로 사용될 수 있다.

▶ Client Rescheduling

서버에서 클라이언트를 다른 서버로 옮길 때 사용한다. 클라이언트를 옮기는 이유는 자신의 성능이 너무 떨어져서 버틸 수 없을 때 클라이언트를 다른 서버로 옮김으로서 이 상태를 해소하려고 하는 것이다. 그러나 이렇게 클라이언트를 다른 서버로 옮기는 작업은 그 자체로 시스템에게 많은 부하를 주기 때문에 주의 깊게 사용되어야 한다.

Client Rescheduling도 정보를 얻어내는 주체가 누구인가에 대한 문제가 존재하듯이 마찬가지로 누가 Client Rescheduling을 원하게 되는지에 따라서 두 가지 방법이 존재한다. 첫째로 과부하가 걸려있는 서버 쪽에서 과부하를 탐지해서 다른 여유 있는 서버로 옮기고 싶어하는 경우(Sender Initiated)가 있다. 두 번째로 전체 System의 구성상태가 바뀌는 경우, 즉 새로운 서버가 추가됐을 경우나 부하가 적은 서버에서 부하를 나누어 받기 원하는 경우(Receiver Initiated)이다[3][4].

▶ Configuration Information

서버들의 구성상태들에 대한 정보를 찾는 기능이다. 현재 서비스하고 있는 서버들의 상태와 설정 등을 알아볼 수 있다. 이것은 Server Pool과 서버에서 사용할 수 있다.

6. 결론

본 논문은 분산 서버 시스템을 위한 Load Balancing Architecture와 Load Balancing Information Transfer Protocol을 제시한다. 이 구조는 Wrapper를 사용해서 기존에 구현된 클라이언트/서버 시스템을 그대로 이용하면서 규모를 확장할 수 있게 된다. 또한 본 논문이 제시하는 구조는 DCOM 및 EJB 등의 Component 서버와 Web Application Server에서도 사용되어서 확장성을 가지게 할 수 있다. 추후에는 Migration Framework과 결합하여 총괄적인 Load Balancing Framework을 구성될 것이다.

7. 참고문헌

[1] Dugki Min, Eunmi Choi, Donghoon Lee, Byungseok Park. A Load Balancing Algorithm For a Distributed Multimedia Game Server Architecture. In Proc. of IEEE Conf. Multimedia Computing and Systems, 1999.
[2] Mukesh Singhal, Niranjan G. Shivaratri. Advanced Concepts in Operating Systems. McGraw-Hill, 1994.
[3] Randy Chow, Theodore Johnson. Distributed Operating Systems & Algorithm. Addison Wesley, 1997.
[4] Dirk Slama, Jason Garbis, Perry Russell. Enterprise CORBA. Prentice Hall PTR, 1999.
[5] SilverStream Application Server, http://www.silverstream.com/website/SilverStream/Pages/products_f.html
[6] IBM WebSphere Application Server, http://www-4.ibm.com/software/webservers/appserv/
[7] BEA WebLogic Application Servers, http://www.beasys.com/products/weblogic/index.html
[8] Alteon Web Switching, http://www.alteonwebsystems.com/products/
[9] Foundry Networks ServerIron Internet Traffic Management Switch, http://www.foundrynet.com/serverironspec.html
[10] MSDN Library, Platform SDK / Component Services / COM+ / Services Provided By COM+ / Load Balancing
[11] MSDN Library, Backgrounders / Windows Platform / Windows NT / Microsoft Windows NT Load Balancing Service