웹 서버 클러스터에서 성능 향상을 위한 차별 큐를 가진 최단 작업 우선 기법

신 재 영, 김 성 천
서강대학교 공과대학 컴퓨터학과
{aquicker, ksc}@arqlab1.sogang.ac.kr

Shortest Job First Technique using Differentiated Queue To Improve Performance On a Web Server Cluster

Jae-Young Shin, Sung-Chun Kim
Department of Computer Science, Sogang University

요 약

일반적으로 부하 분산 정책은 사용자의 서비스 요청 특성에 관계없이 들어오는 그대로 처리하고 있다. 본 논문에서는 이러한 것을 개선하기 위하여 차별 큐를 가진 최단 작업 우선 기법을 제안하였다. 제안한 기법은 부하 분산 서버에 사용자의 서비스 요청이 들어오면 해당 요청의 처리시간을 파악하고 이미 들어온 서비스 요청들과 비교해서 먼저 처리가 완료될 수 있는 서비스 요청을 부하 분산 정책에 할당해주는 전략을 사용함으로써, 사용자들의 평균 응답 시간을 향상시키도록 하였다.

1. 서론

급증하는 웹 서비스 요구에 대하여, 다수의 서버 컴퓨터를 클러스터링 하여 서버의 부하를 분담시키는 웹 서버 클러스터 기술이 최근 효율적인 대안으로 각광을 받고 있다.

웹 서버 클러스터로 유입되는 모든 서비스 요청들이 매번 어느 서버로 서비스 요청이 전송될 것인가를 결정하는 작업을 부하 분산 정책이라고 하는데, 이것이 효율적으로 이루어지지 않으면 과부하(overload) 혹은 저부하(underload) 상황이 발생할 수 있으며 전체 클러스터의 성능을 저하시키는 요인으로 작용한다.

요즈음의 웹 환경에서 중요시되는 지능적인 부하 분산 등을 만족시키기 위해 클라이언트가 요청하는 구체적인 내용을 분석해서 처리하는 내용 기반 부하 분산 기법이 많이 연구되고 있다. 기존의 부하 분산 서버와의 차이점은 TCP/IP의 상위 레벨인 HTTP 레벨에서 부하 분산을 하여 부하 분산 서버로 들어오는 서비스 요청들에 대한 차별화 기법을 수행함으로써 지능적인 부하 분산을 가능하게 한다는 것이다.

본 논문에서는 내용 기반 웹 서버 클러스터 시스템에서 단일 및 다중 차별 큐를 가진 최단 작업 우선 기법을 제안하였다.

2. 웹 서버 클러스터의 성능 향상을 위한 기법들

2.1 DNS-RR의 부하 분산

클라이언트가 원하는 사이트에 접근하기 위해서 DNS에게 IP 주소를 요청하면 DNS-RR은 라운드 로빈(Round Robin) 형식으로 클러스터링된 웹 서버의 IP주소들 중에 하나를 넘겨준다.

이는 실제 웹 서버들의 상태를 알 수 없기 때문에, 웹 서버 가운데 하나가 다운되더라도 해당 웹 서버의 IP 주소를 클라이언트에게 되돌려 주는 경우가 발생할 수 있다[1].

2.2 TCP/IP 레벨의 부하 분산

전용의 부하 분산 서버가 모든 웹 서버들과 하나의 가상 IP 주소를 공유하기 때문에 클라이언트에게 단일 시스템 이미지를 제공할 수 있다. 서비스 요청이 들어오면 부하 분산 서버는 적절한 부하 분산 알고리즘을 이용해서 웹 서버들 중 하나에 TCP 연결을 맺도록 해준다. 그 다음에 해당 클라이언트가 요청하는 모든 서비스들은 TCP 연결을 맺은 웹 서버로 전달한다[2].

2.3 내용 기반 부하 분산

동일한 대상 이름에 대해서는 언제나 같은 서버로 분산하기 때문에, 메인 메모리 캐시의 적중률이 개선되어 웹 서버는 더욱 탤리 응답할 수 있다.

2.4 WWW의 특성

클라이언트가 하는 서비스 요청과 웹 서버에 존재하는 파일들의 특성을 파악하는 것이 중요한데 이 둘의 분포는 Heavy-tailed 특성을 따르는 것으로 알려져 있다[3].

Heavy-tailed 분포의 특성은 쉽게 말해서 "모든 작업 중에서 가장 큰 작업의 1%가 전체 부하의 50%를 차지한다"라는 것이다.

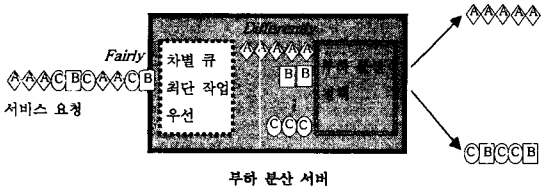
2.5 문제점

기존에는 부하 분산 정책에만 초점을 맞추어서 연구가 되어, 클라이언트들이 보내는 서비스 요청들은 그 특성에 관계없이 들어오는 그대로 부하 분산 정책으로 보내어져 처리되고 있다. 즉, FCFS(First Come Fist Served) 방식을 따르기 때문에 평균 응답 시간은 일반적으로 들어오는 서비스 요청의 순서에 따라서 상당히 큰 폭으로 변할 수 있게 된다.

3. 차별 큐를 가진 최단 작업 우선 기법

3.1 부하 분산 서버

제안 기법은 기존의 부하 분산 정책을 그대로 유지하고 단일 및 다중 차별 큐를 가진 최단 작업 우선 기법을 추가한다.



[그림 3.1] 제안하는 부하 분산 서버

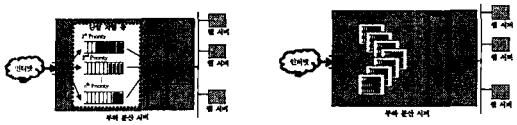
3.2 차별 큐와 최단 작업 우선 알고리즘

차별 큐는 서비스 요청들의 부하를 분석해서 부하 분산 정책으로 가기 전에 차별화 시키는 역할을 한다. i번째 큐는 i-1번째 큐까지가 모두 비어야만 선택될 수 있으며 차별 큐의 분류 기준은 서비스 요청에 대한 부하 정도이다. 차별 큐가 모든 클라이언트들의 서비스 요청을 가지고 있다면 단일 차별 큐가 되는 것이고, 클라이언트마다 독립적인 차별 큐를 가지게 된다면 다중 차별 큐가 되는 것이다.

단일 차별 큐의 경우, 모든 클라이언트의 서비스 요청들은 하나의 차별 큐로 들어가고 여기에서 우선 순위에 맞게 차별적으로 부하 분산 정책을 받아서 적당한 웹 서버로 전달된다.

```
// 6개의 큐를 가지고 차별 큐를 구성한다.
// 단일 차별 큐로 구성되면, 모든 서버 쓰레드가 DQ를 공유할 것이고,
// 다중 차별 큐로 구성되면, 각 서버 쓰레드는 각자의 DQ를 가진 것이다.
Queue[] DQ = new Queue[6];
// 클라이언트의 서비스 요청을 분석해서 차별 큐에 적재한다.
void ServiceRequestAnalysis() {
        while(true) {
        // 클라이언트로부터 서비스 요청을 받는다.
                Socket.Receive(ServiceRequest);
        // ServiceRequest를 분석해서 요청한 파일의 크기를 알아낸다.
                int FileSize = GetFileSize(ServiceRequest);
        // 그리고, 해당 요청을 요청 파일 크기에 따라서 알맞은 큐에 넣는다.
                if(FileSize <= 1024) DQ[0].Enqueue(ServiceRequest);
                else if(FileSize <= 5120) DQ[1].Enqueue(ServiceRequest);
                else if(FileSize <= 10240) DQ[2].Enqueue(ServiceRequest);
                else if(FileSize <= 51200) DQ[3].Enqueue(ServiceRequest);
                else if(FileSize <= 102400) DQ[4].Enqueue(ServiceRequest);
                else DQ[5].Enqueue(ServiceRequest);
        }
}
```

[그림 3.2] 차별 큐와 클라이언트의 서비스 요청을 차별 큐에 적재하는 알고리즘



[그림 3.3] 단일 차별 큐(좌), 다중 차별 큐(우)를 가진 부하 분산 서버

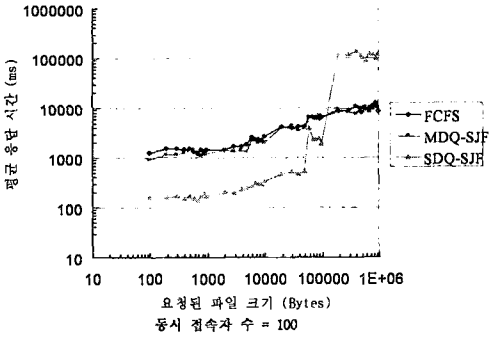
다중 차별 큐는 연결된 클라이언트마다 자기의 차별 큐를 가지게 되어 서비스 요청의 부하에 따라 부분적인 차별적 서비스가 가능하게 되고 전체적인 평균 응답 시간이 줄어들게 된다. 특히, 단일 차별 큐 경우의 문제점인 기아 상태를 막을 수 있다.

4. 프로토타입과 성능 평가

클라이언트에서는 TCP 소켓을 만들고 부하 분산 서버에 연결한 후 서비스 요청을 하게 된다. 하나의 TCP 연결은 멀티 쓰레드 방식으로 다수의 TCP 연결이 동시에 부하 분산 서버로 서비스 요청을 보내게 된다. 부하 분산 서버는 TCP 연결 요청이 들어오면 쓰레드를 생성하고 다시 다른 요청을 기다리는 방식으로 구현하였다. 제안하는 부하 분산 서버는 단일 차별 큐와 다중 차별 큐로 나누어서 각각 구현하였다. 웹 서버는 아파치 웹 서버를 그대로 적용하였으며, 6단계의 큐를 가진 차별

큐를 구현하였다.

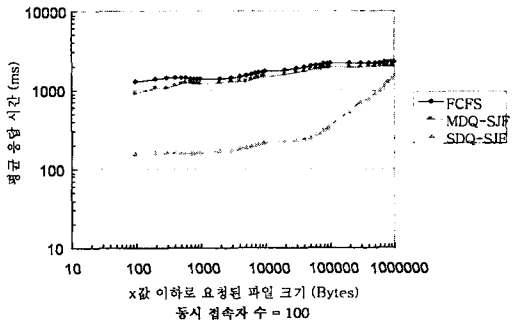
웹 서버 클러스터의 성능은 클라이언트들이 느끼는 빠른 서비스 응답이기 때문에 성능 평가 요소는 동시에 다수의 클라이언트가 다수의 서비스 요청을 하고 서비스를 받을 때까지의 평균 응답 시간으로 정했다. 동시 접속자수를 10명부터 200명까지 범위로 했으며 각 접속자는 100개의 서비스 요청을 하게 된다.



[그림 4.1] 요청된 파일 크기 별 평균 응답 시간

파일 크기 별로 평균 응답 시간을 비교하기 위해서 FCFS, MDQ-SJF, SDQ-SJF를 나타낸 것이고, 기본적으로 파일의 크기가 커짐에 따라서 평균 응답 시간이 늘어나는 것을 볼 수 있다. SDQ-SJF(단일 차별 큐를 가진 부하 분산 서버)는 크기가 작은 파일들에 대해서 평균 응답 시간이 다른 것보다 훨씬 줄어드는 것을 볼 수 있다.

다음은 클라이언트가 요청하는 전체 파일에 대한 평균 응답 시간을 비교하기 위해서 측정한 것이다. 클라이언트들의 서비스 요청들에 대한 평균 응답 시간은 클라이언트가 느끼는 실제 평균 응답 시간이라고 할 수 있다.



[그림 4.2] 클라이언트가 요청한 전체 서비스 요청에 대한 평균 응답 시간

기존의 부하 분산 서버에서 쓰이는 FCFS가 MDQ-SJF와 SDQ-SJF보다 평균 응답 시간이 더 크다는 것을 알 수 있다. 특히, SDQ-SJF는 매우 큰 파일에 대한 나쁜 응답 시간에도 불구하고 전체적인 평균 응답 시간이 매우 감소되는 것을 알 수 있다.

5. 결론

LAN 환경에서의 웹 서버 클러스터 중에 대표적인 내용 기반 부하 분산 기법은 기존의 DNS-RR 기반이나 TCP/IP 레벨의 웹 서버 클러스터의 부하 분산 기법에 비해서 현대적이고 지능적으로 부하 분산을 할 수 있는 장점을 지니고 있다.

본 논문에서 제안한 기법에서는 부하 분산 서버 상에, 서비스 요청의 부하에 따라 서로 다른 큐에 적재하는 차별 큐를 두고 부하 분산 정책에서 우선 순위가 높은 큐에 있는 서비스 요청을 먼저 처리한다. 제안한 기법은 다중 차별 큐를 가진 MDQ-SJF(Multiple Differentiated Queue-Shortest Job First)와 단일 차별 큐를 가진 SDQ-SJF(Single Differentiated Queue-Shortest Job First)이다.

SDQ-SJF는 평균 응답 시간이 매우 많이 감소하는데, 이는 큰 파일에 대한 기아 상태 때문이다. 반면, MDQ-SJF는 많은 성능 향상을 가져오지는 않지만 SDQ-SJF의 단점인 기아 상태를 막을 수 있고, 기존 기법인 FCFS에 비해 성능이 우수하므로 더 효율적이라고 말할 수 있다.

6. 참고 문헌

[1] G.Hunt et al., "Network Dispatcher:A Connection Router for Scalable Internet Services", Proc. 7th Int'l World Wide Web Conf., 1998.

[2] C.S.Yand and M.Y.Luo, "Efficient Support for Content-based Routing in Web Server Clusters," Proceeding of 2nd USENIX/IEEE Symposium on Internet Technologies and Systems(USITS '99), October 11-14, 1999, Boulder, Colorado, USA.
Available at
http://www.usenix.org/events/usits99/full_papers/yang/yang.pdf

[3] Mark E. Crovella, Murad S. Taqqu, and Azer Bestavros, "Heavy-tailed probability distributions in the WWW", In A Practical Guide To Heavy Tails, pages 3-26, Chapman & Hall, New York, 1998.