웹 서버 클러스터를 위한 효율적인 부하 분배 알고리즘

(An Efficient Load Balancing Algorithm for Web Server Cluster)

김 성 수 † 정 지 영 ††
(Sungsoo Kim) (Ji Yung Chung)

요 약 최근 인터넷과 웹은 널리 사용되는 미디어로 인식되고 있으나 상대적으로 빈약한 성능과 낮은 가용도를 제공한다. 클러스터 구조는 고가용도와 고성능 그리고 확장성을 요구하는 웹 서비스나 정보시스템 같은 응용 분야에서 저 비용으로 유용하게 사용 가능하다. 본 논문에서는 고가용도 및 확장성을 제공하는 클러스터링 웹 서버를 대상으로 부하 분배기의 구조를 제안하고 문서 접근 확률과 문서 크기 정보를 이용한 부하 분배 알고리즘을 개발하여 성능을 최대화할 수 있도록 하였다. 특히 제안된 알고리즘은 각 서버 노드가 동일한 운영체제로 구성되기 않아도 되고 처리 용량이 서로 달라도 되며 기존의 알고리즘에 비해 캐쉬 적중률을 향상시킨다.

Abstract Although the web is becoming a widely accepted medium, it provides relatively poor performance and low availability. A cluster consists of a collection of interconnected stand-alone computers working together and provides a high-availability solution in application area such as web services or information systems. In this paper, we propose a load balancer architecture under the web server cluster that provides high-availability, high performance and scalability. In addtion, we propose an efficient load balancing algorithm that considers access rate and size of documents. Specially, our algorithm does not require the nodes running under similar configurations and the same operating system. Also, server cache can be utilized more efficiently.

1. 서 론

최근 인터넷의 사용이 크게 증가하면서 웹을 이용한 상품 및 서비스가 크게 보급, 확산되고 있고 이러한 서비스의 예로 기업의 광고 및 상품의 검색, 전자상거래나 정보 관리 등을 들 수 있다. 현재 인터넷과 웹은 성공적으로 인식되고 있으며 앞으로도 많은 성장 가능성이 있지만 제한된 대역폭과 서버 측 용량의 한계로 인한 여러 문제를 야기하고 있다. 인터넷을 이용하는 사용자가 많아질수록 사용자 모두가 제한된 대역폭을 효율적으로 이용하기는 더욱 어려워지며 따라서 웹 서버의 성능과 확장성(scalability)에 관한 연구가 중요시 된다.

웹은 모든 종류의 데이타와 서비스를 분배하는 미디어로 받아들여지고 있지만 상대적으로 빈약한 성능과 신뢰도를 제공한다. 특히 확장성을 염두에 두고 디자인되지 않았고 사용자나 서비스 제공자의 실수를 고려하여 디자인되지도 않았다. 또한 HTTP는 네트워크에 문제가 발생했을 경우 이를 복구하는 기능을 제공하지 않으므로 사용자 요구는 빈번하게 단일 오류로 인해 이용불가능하게 된다. 따라서 현재의 인터넷을 더욱 성숙시키기 위해서는 웹 서버의 중복에 대한 연구가 필요하다.

한편 대화형 웹 서비스의 중단은 서비스 제공자와 소비자에게 큰 손실을 가져올 수 있기 때문에 웹 서버는 항상 서비스가 가능하도록 결함의 발생을 예측하여 설계하여야 한다. 일정 수준의 신뢰도와 가용도를 제공하기 위하여 많은 서비스 제공자들은 결함 허용 컴퓨터를 이용하여 그들의 서버를 구성해 왔다[1]. 이러한 결함 허용 컴퓨터들은 하드웨어를 중복시키거나 자기 검사(self-checking)를 수행함으로써 결함을 방지하기 때문

· 이 논문은 2000년도 한국학술진흥재단의 연구비에 의하여 연구되었음 (KRF-2000-041-E00289)
† 종신회원 : 아주대학교 정보통신전문대학원 교수
sskim@madang.ajou.ac.kr
†† 학생회원 : 아주대학교 정보통신전문대학원 정보통신공학과
abback@madang.ajou.co.kr
논문접수 : 2000년 11월 13일
심사완료 : 2001년 8월 2일

에 시스템을 구축하기 위해 많은 비용을 필요로 한다. 이에 비해 저가의 PC나 워크스테이션을 이용하는 클러스터링 기법은 비용 측면에서 매우 유리하며 특히 웹 서비스나 정보 시스템과 같이 고성능과 고가용도를 요구하는 응용 분야에서 결함 허용 컴퓨터의 대안으로 등장하고 있다[2, 3, 4].

클러스터의 또 다른 장점은 확장성이다. 독립된 컴퓨터에서는 업그레이드를 위해 하드웨어 및 소프트웨어의 거의 모든 것을 교환하거나 확장하여야 하며, 이 작업은 매우 번거롭고 비용 또한 크다. 그러나 클러스터에서는 단순히 새로운 컴퓨터를 클러스터에 추가하는 것만으로 성능이 향상될 수 있다.

현재 인기있는 웹 싸이트에서는 매일 수백만의 사용자 요구를 수용하기 위해 다수의 서버를 사용하고 있으며 하나의 가상 URL 인터페이스를 이용하여 사용자에게 투명성을 제공하는 방식을 일반적으로 채택하고 있다[5]. 이러한 클러스터링을 이용한 웹 서버 구축 시 중요하게 고려해야할 사항은 각 노드에 적절한 부하 분산이 이루어지도록 하는 것이며 노드에 결함이 발생할 경우에 대비하여 적절한 서비스 시스템 전이가 이루어지도록 해야한다. 이러한 알고리즘은 시스템 성능에 가능한 한 영향을 적게 주어야 하며 결함에 따른 전이 과정에 있어서도 신속히 이루어져야 한다. 부하 분배기(load balancer)는 이러한 중재 알고리즘을 통해 사용자의 요구가 들어올 때마다 각 노드들에게 최적의 부하 분배를 하며 특정 노드에 결함이 발생했을 경우에도 나머지 노드들에게 적절한 분배가 이루어지도록 하는 역할을 한다. 그러므로 부하 분배기의 서비스 정책은 시스템 전체 성능에 중요한 영향을 미치게 된다.

본 논문에서는 고가용도 및 확장성을 제공하는 클러스터링 웹 서버를 대상으로 부하 분배기의 구조를 제안하고 문서 접근 확률(access rate)과 문서 크기 정보를 이용한 부하 분배 알고리즘을 개발하여 성능을 최대화할 수 있도록 하였다. 이러한 연구는 인터넷 기반 비즈니스 시대에 중요한 기반 기술의 하나라 할 수 있다.

논문의 구성으로 2장에서는 관련 연구를 알아보고 3장에서는 클러스터링 웹 서버의 구조 및 제안된 부하 분배기에 대하여 알아본다. 또한 4장에서는 본 논문에서 개발된 부하 분배 알고리즘과 시뮬레이션 결과를 설명하고 마지막으로 5장에서 결론을 맺는다.

2. 관련연구

클러스터 시스템은 대량생산 되는 COTS(commercially off-the-shelf)를 사용하기 때문에 무엇보다 가격대 성능비가 우수하며 리눅스 같은 무료 소프트웨어를 사용할 경우 기존의 동급 고성능 컴퓨터에 비해 십분의 일 이하로 가격을 낮출 수 있다[6]. 특히 Myrinet, Fast Ethernet, Gigabit Ethernet과 같은 고속 네트워크 장비의 개발로 인해 속도 문제가 해결되었기 때문에 현실적으로 실현 가능하다는 장점을 지니고 있다. 널리 알려진 클러스터로는 미국 항공우주국의 Beowulf[7]와 미국 버클리 대학의 NOW (Network of Workstation)를 들 수 있다.

이러한 클러스터링 시스템은 현재 고성능과 고가용성 분야에서 널리 연구되고 있으며 최근에는 웹 서비스와 연계하여 많은 연구가 진행되고 있다[8, 9]. 클러스터링 웹 서버의 부하 분배기는 클러스터 내에 있는 특정 서버에 부하를 집중시키지 않기 위해 적절한 스케줄링 알고리즘을 통해 부하를 분산시키는 역할을 하며 서버 클러스터가 하나의 가상서버로 보이게 하는 역할을 한다. 즉 서비스 요구 패킷이 부하 분배기에 들어오면 스케줄링 알고리즘에 의해 실제 서버가 선택되고 선택된 서버로 그 패킷이 전달된다. 이 때 사용되는 스케줄링 알고리즘에 따라 부하 분배가 이루어지므로 분배 알고리즘은 서버의 성능에 많은 영향을 미치며 부하분배에 이용되는 기준(metric)은 간단하고 빨리 계산될 수 있어야 한다[10].

부하 분배 스케줄링에 이용되는 방법 중 라운드 로빈 스케줄링은 모든 서버 노드들이 동등하게 한번씩 돌아가며 서비스하는 방식으로 각 노드의 서비스 처리 용량과 관계 없이 서비스되므로 부하 불균형이 발생할 수 있는 단점이 있다. 이러한 문제점을 고려하여 라운드 로빈 스케줄링과 유사하나 클러스터에 속한 서버들의 처리 용량에 따라 가중치를 둔 뒤 연결하는 가중 라운드 로빈 스케줄링 방법이 있으며 연결된 사용자 요구 수가 제일 적은 곳을 우선적으로 할당하는 최소 연결 스케줄링 방법 등이 있다.

이러한 부하 분산 기법들을 이용할 경우 하나의 IP 주소로 서비스가 가능하며 서버 노드의 결함 발생 시 나머지 노드들에 부하를 분산시킴으로써 결함을 허용할 수 있는 장점이 있다. 부하 분배기에서 일어날 수 있는 병목 현상은 IP 터널링이나 직접(direct) 라우팅을 통해 실행 서버와 클라이언트가 직접 통신함으로써 해결될 수 있다[11].

대표적인 웹 서버 프로젝트의 하나인 SWEB은 워크스테이션들의 네트워크와 분산 메모리를 사용하는 웹 서버로서 서버 노드 각각이 모든 문서를 가지고 있지 않으며 LAN을 통해 다른 서버 노드의 문서를 가져온다[12]. 이 때 가장 적은 시간이 걸릴 것 같은 서버를 선택하기 위한 기준으로 CPU 처리시간과 디스크 대역폭,

네트워크 지연의 합이 최소인 것을 고려한다. 이 방법은 비교적 정확하게 동적으로 변하는 서버 노드들의 부하를 측정하였으나 자신의 서버에 존재하지 않는 문서를 가져오기 위한 네트워크 오버헤드를 고려해야 하는 단점이 있다.

또한 각 서버 노드의 평균 휴지 시간(idle time)을 이용하여 부하를 분배시키는 방식이 존재한다[13]. 이 방법은 부하 불균형이 발생하더라도 다음 주기가 될 때까지 기다려야 하며, 부하 불균형이 발생할 때마다 매번 부하 균형에 충실하면 너무 많은 오버헤드가 발생하게 되는 단점을 지니고 있다[14]. 이 외에도 CPU 부하나 큐 길이 등을 이용한 부하 분산 기법이 연구되었다[15].

RobustWeb은 미리 정의된 문서 접근 확률에 따라 부하 분배기가 사용자의 요구를 분배한다[16]. 즉 사용자의 요구에 대해 각각의 문서는 접근 확률을 고려하여 서비스 될 서버 노드가 결정된다. 이 방식은 문서들의 크기가 대체로 일정한 경우에 잘 동작하며 하나 이상의 서버들이 다운되었을 경우에 네트워크 플로우를 기반으로 하는 알고리즘을 사용하여 분배 확률을 재 계산함으로써 노드의 결함을 허용한다.

각 서버 노드의 CPU 부하 정보나 휴지 시간을 이용하는 부하 분배의 경우 클러스터 시스템의 부하 분배기는 모든 서버 노드들을 주기적으로 모니터링 하게 되는데, 이 때 부하 모니터링 주기에 대한 결정과 이에 따르는 오버헤드를 최소화하는 것은 매우 중요하다. 이와 같은 동적 부하 분배에서 시스템 부하를 결정하는 방법은 경우에 따라 주관적일 수 있으며 각 서버 노드에서 부하 모니터링 프로그램이 실행되어야 하기 때문에 대부분 동일한 운영체제가 실행되는 것을 전제로 한다[4]. 또한 서버 노드에서 정보를 받을 때와 요구 할당을 위해 정보를 사용할 때의 지연 때문에 실제로는 정확한 부하 분배를 이룰 수 없다.

앞에서 설명한 라운드 로빈 방식이나 최소 커넥션 방식은 정적 처리 방식으로 문서 크기가 유사할 경우에는 잘 동작하나 그렇지 않을 경우에는 문제가 될 수 있다[17]. 또한 웹 문서의 접근 확률에 따라 부하를 분배하는 것은 문서의 크기나 전송시간이 다를 경우 부하 불균형을 초래할 수 있다.

이러한 문제점들을 최소화하기 위해 본 연구에서는 평균 문서 접근 확률과 문서 크기 정보를 동시에 고려하여 부하 분산을 수행하는 효율적인 알고리즘을 개발하고 이를 수용할 수 있는 웹 서버 구조를 제안하였다. 이와 같이 서비스될 문서마다 확률과 함께 서버의 위치가 결정되는 부하 분산 기법을 이용할 경우, 동적 부하 분배에서 발생할 수 있는 서버 노드 모니터링에 대한 오버헤드가 최소화 될 수 있으며 문서 크기 정보를 고려하기 때문에 텍스트 문서와 멀티미디어 문서가 공존한다 할지라도 각 노드들 간에 부하 균형을 이룰 수 있게 된다. 또한 각 서버 노드가 동일한 운영 체제로 구성되지 않아도 되고 처리 용량이 서로 달라도 된다. 특히 성능면에서도 각 노드에서 서비스할 문서 수를 제한하여 캐쉬 적중률(hit rate)을 극대화 할 수 있도록 하였다.

3. 클러스터링 웹 서버 구조

본 연구에서 고려하는 웹 서버 구조는 요구 분배를 위한 부하 분배기와 요구된 문서를 서비스 하는 다수의 서버 노드로 구성되어 있으며 그림 1은 이러한 구조와 함께 본 논문에서 제안하는 부하 분배기 모듈의 구조를 보여주고 있다. 사용자의 요구는 인터넷을 통해 부하 분배기에 전달되며 부하 분배기는 4장에서 제안되는 분배 알고리즘에 의해 주어진 확률에 따라 요구된 패킷을 문서 서버에 전송하는 역할을 한다.

이 구조에서 부하 분배기로부터 요구를 받은 문서 서버는 부하 분배기를 거치지 않고 바로 클라이언트에게 응답을 보내는 직접 라우팅 방식을 이용한다. 이 방식은 부하 분배기를 거쳐 클라이언트에 응답하는 NAT (Network Address Translation) 방식과 달리 부하 분배기가 요구 패킷만 처리하고 서버 노드들이 직접 사용자들에게 응답을 보내기 때문에 병목이 발생하지 않고 100개가 넘는 서버 노드들을 스케줄링할 수 있는 장점을 가지고 있다. NAT 방식의 경우 요구와 응답 패킷을 재 가공해야 하기 때문에 서버 숫자가 20개 이상이면 전체 시스템에 병목을 가져올 수 있다[11].

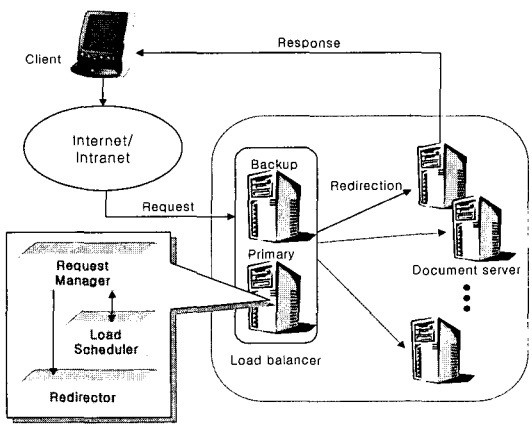

그림 1 고가용도 웹 서버 클러스터 구조

위의 구조에서 각각의 문서 서버는 주기적으로 heartbeat 메시지를 통해 자신의 존재를 부하 분배기에 알리며 문서 서버가 다운되면 부하 분배기는 이를 고려하여 알고리즘을 재 수행함으로써 나머지 노드들에 부하를 분산시킨다. 특히 부하 분배기에서 주 수행 서버와 백업 서버를 hot-standby나 cold-standby 방식으로 운영함으로써 부하 분배기 자체의 결함을 허용할 수 있다.

각각의 문서 서버는 네트워크 부하를 피하기 위해 자신의 기억 장치에 모든 웹 문서를 포함할 수도 있고 모두 하나의 안정된 기억 장치(stable storage)를 통해 서비스할 수도 있다. 일반적으로 웹 문서의 경우 상대적으로 적은 용량의 문서가 대부분이고 하드디스크도 GMR 헤드의 등장으로 기록 밀도가 대폭 향상되어 가격과 용량 면에서 급격한 발전을 보이고 있으므로 용량에 대한 문제는 감소하는 추세이나 데이타 일관성(consistency)에 대한 고려가 전제되어야 한다.

특히 부하 분배기는 사용자 요구를 처리하는 부분(Request Manager), 주어진 정보를 이용하여 각 문서에 따라 요구될 서버를 확률과 함께 결정하는 부분(Load Scheduler), 그리고 결정된 계획에 따라 실제 분배하는 부분(Redirector) 등으로 구성된다. Request Manager는 분배 테이블(Redirection Table)을 참조하여 문서 서버를 결정하며 문서에 대한 사용자 요구 수를 관리하여 분배 테이블에 기록한다. Load Scheduler는 이 정보를 이용하여 주기적으로 알고리즘을 수행함으로써 분배 확률을 갱신하며 각 문서 서버의 heartbeat 메시지를 받는다.

문서 서버의 결함 발생 시 대처할 수 있는 방법으로는 노드가 수리 복구될 때까지 큐에 작업들을 대기시키는 방법, 여분의 노드로 이동하여 작업하는 방법, 정상적으로 수행되는 다른 노드들에 재 분배하는 방법 등이 사용될 수 있다[19]. 첫 번째 방법은 사용자 응답 시간이 너무 길어지게 되고 두 번째 방법은 추가적인 하드웨어 비용이 필요하므로 본 논문에서는 세 번째 방법을 대상으로 하였다.

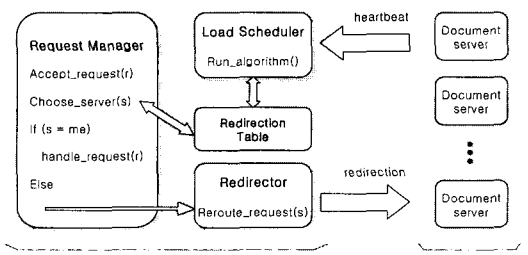

그림 2 부하 분배기 모듈간의 관계

그러므로 문서 서버 노드 중 하나가 다운되었다는 이벤트가 발생하면 부하 분배기는 문서의 새로운 분배 확률을 계산하기 위해 그 문서 서버를 제외하고 알고리즘을 재 수행함으로써 분배 테이블을 갱신하게 된다. 그림 2는 이러한 부하 분배기 모듈들의 기능과 관계를 보여주고 있다.

4. 부하 분배

4.1 부하분배 알고리즘

클러스터를 구성하는 방법은 RR-DNS(Round-Robin DNS) 방식과 부하 분배기를 이용하는 방법이 있는데 RR-DNS는 하나의 도메인 이름에 다른 IP 주소를 맵핑하는 방식이다. 이는 서버가 다운됐을 경우 DNS에서 다운된 서버를 직접 삭제해야 하는 단점이 있어 고가용도를 제공하기에는 문제가 많다. 이에 비해 부하 분배기를 이용하게 되면 서버 클로스터가 하나의 IP를 가진 가상 서버로 보이게 된다.

이 절에서는 각 문서 서버의 부하를 일정하게 유지하기 위해 부하 분배기에서 사용될 새로운 알고리즘을 제안한다. 현재 부하 분배에 관한 대부분의 연구에서 부하의 균형을 맞추기 위한 기준으로 CPU 큐 길이나 디스크 사용률 등을 조합하여 사용하고 있다. 그러나 사용자의 요구가 웹 문서에 한정되는 전용의 웹 서버 시스템일 경우에는 이러한 기준이 자연스러운 방법이 될 수 없다[16]. 따라서 웹 문서의 접근 확률 같은 기준이 사용된 연구가 있으나 문서의 크기가 서로 다르면 처리 시간이 달라지므로 이를 고려한 알고리즘이 사용되어야 한다.

또한 기존의 부하 분배 알고리즘들은 각각의 서버 노드에서 서비스 가능한 모든 문서가 서비스될 확률이 대체로 동일하다. 이에 비해 제안된 방법은 문서 접근 확률과 문서 크기 정보를 이용하여 부하 분산을 이루는 동시에 가능한 범위 내에서 문서 전용 노드를 이용할 수 있도록 함으로써 하나의 서버에 대해 서비스될 문서의 수를 줄이는 방법으로 캐쉬의 적중률을 극대화 할 수 있게 하였다. 이것은 부하 분배 알고리즘의 가장 근본적 목적인 사용자 응답 시간의 단축에 부합한다.

제안된 알고리즘을 설명하기 위해 웹 서버에 존재하는 M개의 문서 각각이 사용자로부터 요구될 확률 R_1, R_2 ,…, R_M($R_1+R_2+\cdots+R_M=1$)과 함께 주어지고 각각의 문서 크기는 D_1, D_2, …, D_M과 같이 주어진다고 가정하자. 문서의 요구확률 R_i를 결정하기 위한 방법으로 사용자의 특정 문서에 대한 요구는 Request Manager에 의해 해당 문서의 요구 수를 증가시키고 문서들에 대한 총

요구수를 해당 문서에 대한 요구 수로 나누게 된다[20].

이 때 사용자로부터 문서가 요청되면 부하 분배기는 주어진 확률에 따라 N개의 문서 서버 S_1, S_2, …, S_N 중에서 서비스 받을 서버를 결정하고 패킷을 전달하게 된다. 또한 C_1, C_2, …, C_N은 시스템 전체에서 각 문서 서버가 처리할 수 있는 용량의 비율을 의미하며 $C_1+C_2+\cdots+C_N=1$이 된다. 이것은 제안된 알고리즘이 다양한 처리 능력을 가진 서버 노드로 구성된 클러스터링 시스템에서도 동작할 수 있다는 것을 의미한다.

실제 알고리즘에서는 R_i와 D_i를 이용하여 문서 크기 정보를 고려한 가중 문서 접근 확률 Reweight_R_i를 구하여 이를 각 서버가 처리할 수 있는 용량인 C_i에 적절히 할당하게 되는데 구체적으로 각 문서가 각 서버에 요구될 확률을 결정하는 알고리즘은 다음과 같다.

알고리즘이 동작하는 과정을 예를 들어 설명하면 다음과 같다. 5개의 문서 각각이 0.35, 0.15, 0.4, 0.05, 0.05의 접근 확률을 가지고 있으며 각각의 문서 크기가 10KB, 30KB, 3KB, 10KB, 6KB라고 하자. 또한 클러스터를 구성하는 서버들간의 처리 용량 비율인 C 값은 각각 0.4, 0.3, 0.3의 값을 갖는다고 가정하자. 이 때 문서 크기 정보를 고려한 Reweight_R_i는 알고리즘에 의해 0.35, 0.45, 0.12, 0.05, 0.03으로 결정되어지며 문서 중 최대 값을 가진 D_2가 선택된다. 따라서 D_2는 최대 C값을 갖는 S_1에 할당되는데 S_1은 0.4의 C 값을 가지고 있으므로 0.45중 0.4만 S_1에 할당되고 나머지 0.05는 S_2에 할당된다. C 값이 같을 때는 임의의 서버가 선택된다고 가정한다.

문서 D_2의 Reweight_R_i가 모두 할당되면 그 다음으로 높은 값을 갖는 D_1이 선택된다. 이 때 각 문서 서버에 남아 있는 C 값은 각각 0, 0.25, 0.3 이므로 D_1은 S_3에 우선적으로 할당된다. 이와 같이 모든 문서와 서버에 대해 수행되어 바깥쪽 While 문을 완전히 빠져 나온 후의 결과는 표 1과 같고 최종적으로 알고리즘의 수행이 끝나면 분배 테이블은 비로소 표 2와 같은 결과를 갖게 된다.

표 1 While 문 수행 후의 결과

	D_1	D_2	D_3	D_4	D_5
S_1	0	0.4	0	0	0
S_2	0.05	0.05	0.12	0.05	0.03
S_3	0.3	0	0	0	0

표 2 알고리즘 수행 결과

	D_1	D_2	D_3	D_4	D_5
S_1	0	0.89	0	0	0
S_2	0.14	0.11	1	1	1
S_3	0.86	0	0	0	0

Redirection Algorithm

```
Total ← 0
For i ← 1 to M {
   For j ← 1 to N {
      do Redirection_table[i][j] ← 0
   }
}
For i ← 1 to M {
   do Reweight_Ri ← Ri × Di
}
For i ← 1 to M {
   do Total = Total + Reweight_Ri
}
For i ← 1 to M {
   do Reweight_Ri ← Reweight_Ri / Total
}
Temp_Ri ← Reweight_Ri for each document
While (if any document left) {
   do select document i with maximum Reweight_Ri
      While (if Reweight_Ri is not 0) {
         do select server j with maximum capacity Cj
            if (Cj >= Reweight_Ri) {
               Cj ← Cj - Reweight_Ri
               Redirection_table[i][j] ← Reweight_Ri
               Reweight_Ri ← 0
            }
            else {
               Reweight_Ri ← Reweight_Ri - Cj
               Redirection_table[i][j] ← Cj
               Cj = 0
            }
      }
}
For i ← 1 to M {
   For j ← 1 to N {
      do Redirection_table[i][j]← Redirection_table[i][j] /
         Temp_Ri
   }
}
```

즉 D_1에 대한 사용자의 요구는 0.14의 확률로 S_2, 0.86의 확률로 S_3에 할당되며 D_2에 대한 사용자의 요구는 0.89의 확률로 S_1, 0.11의 확률로 S_2에 할당된다. 또한 D_3, D_4, D_5는 모두 S_2에 할당되며 그림 3은 이러한 부하 분배 과정을 보여주고 있다. 여기서 부하 분배기로부터 나가는 선 위의 숫자는 각 문서의 접근 확률을 나타낸다.

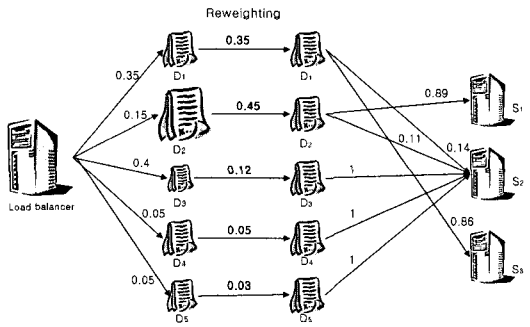

그림 3 부하 분배 알고리즘 수행과정

그림 3과 같은 확률로 서비스를 수행하는 도중에 문서 서버가 다운되거나 처리 용량이 바뀔 때 또는 문서의 접근 확률이 변할 경우 이에 따라 새로운 문서 분배 정책이 필요하다. 예를 들어 서비스 도중 S_1이 다운되었다고 가정하면 Load Scheduler 모듈은 S_1로부터 heartbeat 메시지를 받지 못하게 된다. 이 때 S_1을 서버 리스트에서 제외하고 S_2와 S_3은 동일한 처리 용량을 가지고 있으므로 C값을 각각 0.5로 바꾼 후 다시 부하 분배 알고리즘을 수행하게 된다. 그 결과 분배 테이블은 표 3과 같은 값을 가지게 되며 S_1은 복구될 때까지 서비스 요구를 받지 않게 된다. 그림 4는 결함 발생 후 각 문서의 서비스를 보여주고 있다. 이 경우 D_2와 D_4는 S_2에서 전용으로 서비스하며 D_1, D_3, D_5는 S_3에서 서비스하게 되므로 캐쉬의 적중률을 높일 수 있다. 즉 라운드 로빈이나 최소 연결 스케줄링과 같은 방법에서 각각의 서버는 모든 문서를 대상으로 서비스가 이루어지나 제안된 방법에서는 전체 문서 중 일부만 서비스가 요구되므로 요구된 문서가 캐쉬 내에 존재할 확률이 높아지게 된다.

초기에 클러스터 시스템의 성능을 제한하는 지배적인 요소는 프로세서의 속도였으나 하드웨어가 향상됨에 따라 프로세서가 시스템 전체 성능에 미치는 영향이 감소되었다. 현재는 메모리 대역폭이 성능의 병목으로서 프로세서의 역할을 대신하고 있다[4]. 네트워크 역시 항상 클러스터 시스템의 성능을 제한하는 요소였으나 100Mbps 이더넷과 함께 영향이 감소하는 추세이다. 이러한 견지에서 볼 때 캐쉬의 적중률 향상은 클러스터 시스템 성능에 많은 영향을 준다고 할 수 있다.

현재까지 제안된 부하 분배 알고리즘들은 각 서버에서 각 문서들을 서비스할 확률이 대체로 동일하다. 따라서 n개의 문서가 존재하고 캐쉬가 하나의 문서만 포함할 수 있다고 가정할 때 캐쉬의 적중률은 1/n이 된다. 이에 비해 제안된 알고리즘을 이용할 경우 가장 이상적인 경우에, 즉 n개의 문서가 n개의 서버에 일대 일로 할당되면 100%의 적중률을 가지게 되며 최악의 경우라 할지라도 1/n의 적중률을 가지게 된다.

표 3 S_1에서 결함발생 후 분배 테이블 값

	D_1	D_2	D_3	D_4	D_5
S_1	0	0	0	0	0
S_2	0	1	0	1	0
S_3	1	0	1	0	1

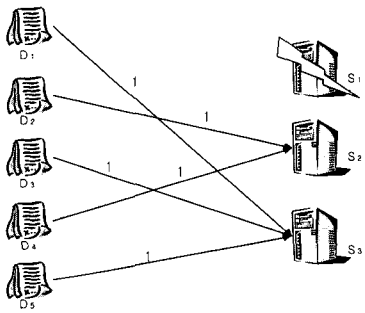

그림 4 결함발생 후 서비스 과정

4.2 성능 평가

이 절에서는 앞에서 제안한 분배 알고리즘이 제대로 부하 분배를 이루는지 알아보기 위해 시뮬레이션 실험을 수행하였으며 그 결과를 보여준다. 시뮬레이션에 사용된 파라미터 값으로 각 문서의 크기와 문서 접근 확률, 서버의 수와 처리 용량 등은 4.1절의 예와 동일하다.

그림 5는 사용자의 문서 요구 수가 증가함에 따라 각 서버 노드에 부과되는 양을 보여주고 있다. 서버 1은 서버 2와 서버 3에 비하여 약 4/3 배의 요구를 받고 있음을 볼 수 있는데 이는 서버 1의 처리 용량이 그 만큼

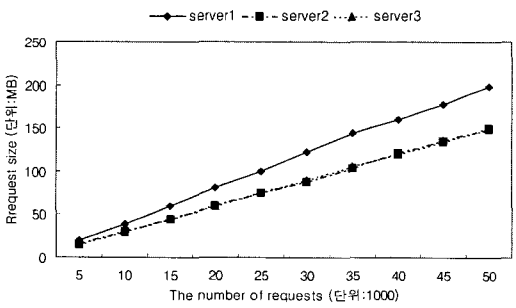

그림 5 사용자 요구에 따른 서버 부하

높기 때문이다. 따라서 처리 용량을 고려하면 부하 분배가 이루어짐을 알 수 있다.

그림 6은 시간의 증가에 따른 서버 부하 비율을 보여준다. 서버의 부하는 특정 시점에서 해당 서버가 처리해야할 데이타의 양을 기준으로 측정하였으며 서버 부하 비율은 각 서버의 (부하/처리용량)를 전체 시스템에 대해 나눈 값으로 서버의 수를 세 개로 하여 실험하였기 때문에 각각 약 0.33 정도에서 수렴하고 있음을 볼 수 있다. 그림에서 알 수 있듯이 일정 시간이 지나면 점차 안정화 상태(steady-state)로 들어가게 되어 각 서버의 부하 비율이 동일해 진다.

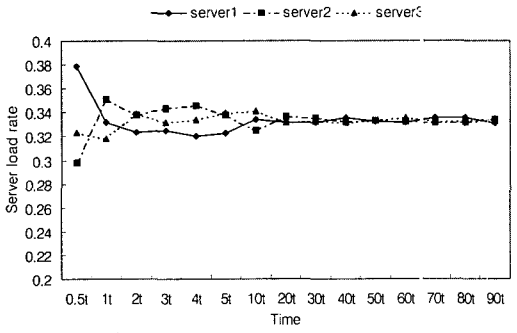

그림 6 시간에 따른 서버 부하 비율

2장에서 언급했던 RobustWeb과 같이 서비스를 대상으로 하고 있는 문서들의 크기가 유사하다고 가정하면 문서 접근 확률만 가지고도 어느 정도 부하 분산을 이룰 수 있다. 이러한 경우 라운드 로빈 방식도 잘 동작하지만 문서 크기가 차이가 많을 경우에는 이를 고려해야 한다. 그림 7과 그림 8은 이를 보여주고 있는데 실제로 다섯개 문서 각각의 크기를 10KB, 30KB, 3KB, 10KB, 6KB로 하여 시뮬레이션 하였다.

그림 7은 문서 크기 정보를 고려하지 않고 문서 접근 확률만 가지고 실험한 결과로서 서버 1이 서버2와 서버3에 비해 다소 부하가 적게 걸리는 것을 볼 수 있다. 따라서 RobustWeb과 같이 문서 접근 확률만 고려해서는 제대로 부하 분산을 이룰 수 없음을 알 수 있다. 이에 비해 그림 8은 문서 크기를 고려한 제안된 알고리즘을 이용한 것으로 부하 분배가 잘 이루어짐을 볼 수 있다.

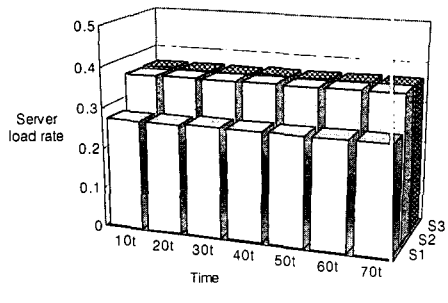

그림 7 문서 접근 확률만 고려한 부하 분배

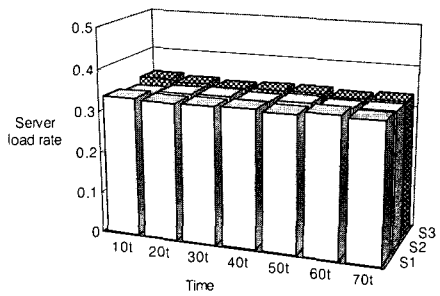

그림 8 문서 크기를 고려한 부하 분배

그림 9는 세 개의 서버로 서비스를 하는 도중에 서버 1에서 결함이 발생한 후의 서버 부하 비율을 보여주고 있다. 이 경우 서버 1에는 더 이상의 부하가 걸리지 않으며 서버 2와 서버 3이 각각 0.5 정도의 부하 비율로 서비스하게 됨을 알 수 있다.

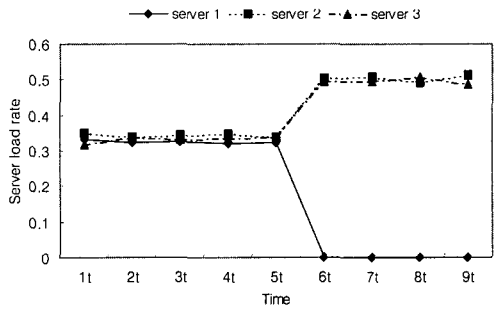

그림 9 서버 1에서 결함발생 후 서버 부하 비율의 변화

실제로 가중 라운드 로빈과 제안된 방식의 부하 분배를 비교한 결과 정확도 측면에서는 거의 유사하였다. 그러나 앞에서 설명한 바와 같이 기존의 부하 분배 알고리즘들은 서버 노드에서 각각의 문서들을 서비스할 확률이 대체로 동일하나 제안된 방식은 특정 노드에서 서비스되는 문서의 수를 제한함으로써 디스크 접근을 줄이고 캐쉬 적중률을 향상시키는 장점이 있다.

그림 10은 이러한 알고리즘과 비교해 우리가 제안한 알고리즘의 캐쉬 적중률이 우수함을 보여주고 있다. 그

래프에서 경우 1은 클러스터 시스템 내의 서버 1에서 결함이 발생하기 전의 캐쉬 적중률이며 경우 2는 결함이 발생한 후의 캐쉬 적중률을 나타낸다. 여기서 캐쉬 적중률은 각 서버에서 서비스하는 문서들 크기의 합과 문서 접근 확률에 따라 결정된다. 시뮬레이션에서 사용된 캐쉬 정책은 LRU(Least Recently Used) 방식이며 교체되는 블록의 크기는 1KB 단위로 하여 실험하였다.

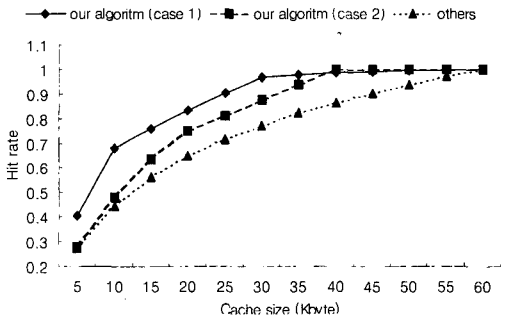

그림 10 캐쉬 적중률 비교

5. 결 론

클러스터 시스템은 고가용도와 고성능을 요구하는 웹 서비스와 같은 응용 분야에서 유용하게 사용될 수 있을 뿐 아니라 확장성이 요구되는 분야에서 저비용으로 사용 가능하다. 웹 서버 클러스터에서 부하 분배 알고리즘은 시스템 전체의 성능을 좌우하기 때문에 신중히 고려되어야 하며 부하 분배로 인한 오버헤드가 최소화 되도록 설계하여야 하나, 아직 주로 사용되는 표준은 존재하지 않는다. 따라서 이 분야에 대한 더 많은 연구가 필요하다.

본 연구에서는 클러스터링 기법을 이용한 고가용도 웹 서버 시스템을 대상으로 부하 분배기의 구조를 제안하고 평균 문서 접근 확률과 문서 크기 정보를 동시에 고려하여 부하 분산을 수행하는 효율적인 알고리즘을 개발하였다. 이와 같이 서비스될 문서마다 확률과 함께 서버의 위치가 결정되는 부하 분산 기법을 이용할 경우, 서버 노드 모니터링에 대한 오버헤드가 최소화 될 수 있으며 문서 크기 정보를 고려하기 때문에 텍스트 문서와 멀티미디어 문서가 공존한다 할지라도 각 노드들 간에 부하 균형을 이룰 수 있게 된다. 또한 각 서버 노드에서 서비스할 문서의 수를 제한하여 캐쉬 적중률을 극대화 할 수 있도록 하였으며 다양한 처리 용량을 지닌 서버 노드에 대해서도 동작 가능하게 하였다.

웹의 경우 동일한 클라이언트와 서버 사이에서 세션을 일정하게 유지해야할 필요가 있다. 이와 같은 경우를 해결하기 위한 방법 중 한가지는 비록 해당 노드에 추가적인 부하가 걸릴지라도 클라이언트와 특정 노드 사이에 세션을 유지하도록 하는 것이며 이에 대한 추가적인 연구가 필요하다.

참 고 문 헌

[1] R. Friedman and D. Mosse, "Load Balancing Schemes for High-Throughput Distributed Fault-Tolerant Servers," Journal of Parallel and Distributed Computing, pp. 475-488, Dec. 1999.
[2] V. Cardellini, M. Colajanni and P.S. Yu, "Dynamic Load Balancing on Web-server Systems," IEEE Internet Computing, pp. 28-39, May 1999.
[3] H. Zhu, T. Yang, Q. Zheng, D. Watson, O.H. Ibarra and T. Smith, "Adaptive Load Sharing for Clustered Digital Library Servers," Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, pp. 28-31, July 1998.
[4] R. Buyya, "High Performance Cluster Computing: Architectures and Systems," Prentice-Hall, Chapter 14, p. 849, 1999.
[5] V. Carellini, M. Colajanni and P. Yu, "Redirection Algorithms for Load Sharing in Distributed Web-server Systems," Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, pp. 528-535, May 1999.
[6] 유찬수, "리눅스 클러스터링", 정보과학회지, 제18권 제2호, pp. 33-39, 2000. 2.
[7] Beowulf Project, http://www.beowulf.org
[8] A. Wong and T. Dillon, "Load Balancing to Improve Dependability and Performance for Program Objects in Distributed Real-time Co-operation over the Internet," The 3rd IEEE International Symposium on Object-Oriented Real-time Distributed Computing, Mar. 2000.
[9] 김동호, 박권, 김명호, "리눅스 상에서 고가용성 웹 서버 클러스터의 설계 및 구현", 2000 한국정보과학회 춘계학술발표논문지, Vol. 27, No. 1, 2000. 4.
[10] E. Khalil and T. Carl, "On Metrics for the Dynamic Load Balancing of Optimistic Simulations," Proceedings of the 32nd Hawaii International Conference on System Sciences, Jan. 1999.
[11] Linux Virtual Server Project, http://www.linuxvirtualserver.org
[12] D. Andresen, T. Yang, V. Holmedahl and O. Ibarra, "Sweb: Towards a Scalable WWW Server on Multicomputers," Proceedings of International Symposium on Parallel Processing, IEEE, pp. 850-856, Apr. 1996.
[13] B. S. Siegel and P. Steenkiste, "Automatic Generation of Parallel Programs with Dynamic Load Balancing," Proceedings 3rd International Symposium on High-Performance Distributed

Computing, pp. 166-175, 1994.
[14] H. Maeng, H. Lee, T. Han, S. Yang and S. Kim, "Dynamic Load Balancing of Iterative Data Parallel Problems on a Workstation Clustering," Proceedings of the High-Performance Computing on the Information Superhighway, HPC-Asia, Apr. 1997.
[15] M. Colajanni, P. Yu and D. Dias, "Scheduling Algorithms for Distributed Web Servers," Proceedings of International Conference on Distributed Computer Systems, IEEE, pp. 169-176, May 1997.
[16] B. Narendran, S. Rangarajan and S. Yajnik, "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access," Proceedings of the 16th Symposium on Reliable Distributed Systems (SRDS), Oct. 1997.
[17] 권세오, 김상식, 김동승, "리눅스 클러스터형 웹 서버 설계", 정보과학회지, 제 18권 제3호, pp. 48-55, 2000. 3.
[18] 최병갑, 이천희, "분산 시스템의 결합시 지분배 알고리즘의 선정기준을 위한 특성 분석", 한국 시뮬레이션 학회 논문지 제3권, 제 1호, pp. 89-97, 1994. 7.
[19] Linux-HA Project, http://www.linux-ha.org
[20] J. Griffioen and R. Appleton, "The Design, Implementation, and Evaluation of a Predictive Caching File System," Technical Report CS-264-96, University of Kentucky, June 1996.

김 성 수

1982년 서강대학교 전자공학과(공학사). 1984년 서강대학교 전자공학과(공학석사). 1995년 Taxas A&M University, 전산학과(공학박사). 1983년 ~ 1986년 삼성전자(주) 종합연구소 컴퓨터연구실 (주임연구원). 1986년 ~ 1996년 삼성종합기술원 수석연구원. 1991년 ~ 1992년 Taxas Transportation Institute 연구원. 1993년 ~ 1995년 Taxas A&M University, 전산학과, T.A. 1997년 ~ 1998년 한국정보과학회, 한국정보처리학회 논문지 편집위원. 1995년 ~ 현재 아주대학교 정보통신전문대학원 부교수. 관심분야는 클러스터시스템, 그리드컴퓨팅, 고가용성 시스템, 시스템 성능분석

정 지 영

1998년 아주대학교 정보 및 컴퓨터공학부(공학사). 2000년 아주대학교 정보통신전문대학원 정보통신공학과(공학석사). 현재 아주대학교 정보통신전문대학원 정보통신공학과 박사과정. 관심분야는 클러스터 시스템, 고가용성 시스템, 시스템 성능분석, 이동컴퓨팅