부하분산 알고리즘을 적용하여 반응시간을 감소시키는 웹 클러스터 시스템 구축

(System Infrastructure of Efficient Web Cluster System to Decrease the Response Time using the Load Distribution Algorithm)

김 석 찬 † 이 영 ††
(Seok-chan Kim) (Young Rhee)

요 약 본 논문에서는 웹 클러스터 시스템의 효율적인 자원사용 방법에 관하여 연구하고자 한다. 시스템 자원, 즉 메모리 사용을 균형적으로 하는 웹 클러스터 시스템을 구현하기 위한 부하 분산 알고리즘을 제안하여 검토하고자 한다. 다양한 클러스터링 모델에서 반응시간을 성능 측정의 지수로 제시하였다. 또한 웹 클러스터 시스템의 동시사용자수를 기반으로 반응시간은 사용자 수를 증가시키면서 실험을 실시하였다. 제안된 알고리즘에 대한 모의실험 결과는 기존의 상용화된 알고리즘보다 좋은 결과를 보여주고 있다. 제안된 부하분산 알고리즘은 동시 사용자 수가 많은 시스템의 경우, 더 좋은 결과를 보여주고 있다. 제안된 알고리즘은 이기종으로 구성된 웹 서버 시스템에 유용할 것으로 사료된다.

키워드 : 웹 클러스터 시스템, 동시사용자, 효율성, 반응시간

Abstract In this paper, we consider the methodology of efficient resource usage, specially web clustering system. We develope an algorithm that distributes the load on the web cluster system to use the system resources, such as system memory equally. The response time is chosen as a performance measure on the various clustering models. And based on the concurrent user to the web cluster system, the response time is also examined as the number of users increases. Simulation experience with this algorithm shows that the response time seems to have a good results compare to those with the other algorithm. And, also the effectiveness of clustered system becomes better as long as the number of concurrent user increases. The usage of developed algorithm is more useful when the system consists of many different sub-systems, a heterogeneous clustering system.

Key words : Web cluster system, Concurrent user, Effectiveness, Response time

1. 서 론

컴퓨터의 처리 능력을 향상시키기 위한 연구는 다양한 방법으로 이루어져왔고 지금도 진행되고 있다. 초기의 연구는 주로 컴퓨터를 구성하는 컴포넌트의 성능을 향상시켜 전체 시스템의 성능을 향상시키는 하드웨어적 관점에서 이루어졌다. 그러나 다른 한편으로는 기존의 시스템을 재구성하여 성능을 향상시키는 소프트웨어적인 관점의 연구 분야로 확대되어, 그 중의 하나가 클러스터 시스템에 대한 연구라고 할 수 있다. Pfister[1]에 의하면 클러스터 시스템은 "전체 컴퓨터들이 상호 연결된 컴퓨터 집합으로 구성되며 이렇게 통합된 자원들이 하나의 단일 컴퓨터처럼 사용되는 병렬 혹은 분산 시스템의 한 종류다." 라고 정의하고 있다. 클러스터 시스템의 기본적인 특성은 고가용성 추구에 있지만, 응용분야에 따라 고속 계산 능력을 가진 HPC(High Performance Cluster) 시스템과, 웹 클러스터 시스템으로 나눌 수 있다. 전자의 경우, 고성능의 계산 능력에 초점이 맞추어져 있어 주로 과학계산용으로 활용되고 있다. HPC 시스템들은 고도의 계산능력이 요구되는 문제를 여러 개의 작은 문제로 분할하여 시스템의 다수의 하부 시스템(이하 노드로 표기)에 분배한다. 각 노드는 할당받은 문제를 계산하고 각 노드의 결과를 통합하여 전체 문제를 해결하는 방식을 사용한다. 후자의 경우, 웹 혹

† 비 회 원 : 계명대학교 산업공학과
kalsman@kmu.ac.kr
†† 정 회 원 : 계명대학교 산업시스템공학과 교수
yrhee@kmu.ac.kr
논문접수 : 2004년 2월 4일
심사완료 : 2004년 8월 12일

은 데이타베이스 시스템 등의 응용 프로그램에 대한 부하를 여러 대의 시스템으로 분산시킴으로서 시스템에 미치는 부하를 감소시키는 방식이다. 지금까지 응용분야에서 사용되고 있는 클러스터 시스템들은 운영체계에 의존적이라 할 수 있고 대표적인 예로 NT 계열의 클러스터 시스템과 UNIX 계열의 시스템으로 나눌 수 있다. NT 계열의 경우, Cornhusker, WLBS와 같은 시스템을 이용한 웹 클러스터 시스템이 대표적이다. 이들 시스템은 서비스 측면에서의 고 가용성에 초점이 맞춰진 시스템이라고 할 수 있다. UNIX 계열의 경우, 고성능의 계산에 초점을 맞춘 Beowulf 시스템과 웹, 기타 인터넷 서비스의 부하 분산에 초점을 맞춘 LVS(Linux Virtual Server) 클러스터 시스템이 있다.

인터넷 사용자는 매년 폭발적으로 증가하고 있다. 특히 웹서비스의 경우, 정보 전달이라는 1차 적 기능을 넘어서 웹을 통한 경제활동 즉, 전자상거래의 발전으로 그 사용은 더욱 활성화되고 있다. 특히 사용자의 웹 서비스 요청이 폭주하는 사이트의 경우, 사용자의 만족도를 충족하는 QoS와 가용성을 보장하기 위한 대안으로는 하드웨어적인 확장, 즉 단일 서버의 업그레이드 또는 추가하는 것과 소프트웨어적인 확장, 즉 소프트웨어적인 클러스터링을 통한 확장을 고려할 수 있다. 하드웨어적인 확장의 경우, QoS와 가용성은 충족할 수 있으나 투자비용이 크다라는 점을 들 수 있고, 이외에도 확장의 한계 및 기술적인 어려움도 존재한다. 한편 소프트웨어적인 확장에서는 시스템의 가용성과 QoS를 보장할 수 있다는 점 외에 비교적 적은 비용으로 확장이 가능하다는 장점이 있다. 또한 확장 면에서도 하드웨어적인 확장보다는 상대적으로 우수하기 때문에 클러스터링 기술에 대한 연구가 진행되고 있다. 특히 인터넷 서비스 분야에서 웹 클러스터 기술이 도입되어 폭주하는 사용자의 요청에 대한 QoS를 보장해주려는 연구가 목격되고 있다[2,3,4]. 그러나 기존 웹 클러스터 시스템의 사용으로 사용자의 요청에 대하여 어느 정도 QoS를 보장할 수 있게 되었지만, 시스템 자원의 균등한 사용은 고려되지 않았다. 따라서 시스템 자원의 균등한 사용이 추가적으로 고려되었을 때 웹 클러스터 시스템의 QoS 변화를 분석하는 것은 의미가 있다. 여기서의 시스템 자원이란 가용한 메모리의 양과 같이 시스템의 성능에 영향을 미치는 요소를 칭한다. 기존 웹 클러스터 시스템의 경우 실제 서비스를 제공하는 서버의 현재 접속 수에 근거하여 접속을 분배하는 방식으로 부하를 분산시키고 있어 클러스터 시스템을 구성하는 서버들의 시스템자원의 사용 정도는 무시되고 있다는 것이다. 이러한 상황은 동일기종의 서버로 구성된 클러스터 시스템조차도 시스템 자원이 균등하게 사용될지 의문이다. 웹 클러스터 시스템을 구성하는 서버 중 시스템의 자원을 가장 많이 소모하는 서버에게도 같은 양의 부하가 계속 부과된다면 전체 웹 클러스터 시스템의 응답시간을 저하시킬 수 있는 요인이 될 수 있다. 그러므로 각 서버의 자원 사용량과 접속 수를 동시에 고려한 부하분산 정책을 사용할 경우의 응답시간과 접속 수만을 고려한 부하분산 정책에서의 응답시간을 상호 비교함은 의미가 크다고 할 수 있다.

본 연구는 클러스터 시스템의 자원을 균등하게 사용하도록 하는 알고리즘을 바탕으로 하여 응답시간을 향상시킬 수 있는 방법에 관하여 연구를 수행하고자 한다. 본 연구는 2절에서 선행 연구 사례에 관하여 서술하고, 3절에서 부하분산 알고리즘 및 실험환경에 대해 서술한다. 그리고 4절에서는 실험방법 및 실험 결과와 그 분석을 서술하고 마지막으로 5절에서 본 연구의 결론과 향후 연구방향에 대해 서술한다.

2. 선행연구

2.1 부하 분산 방법의 접근

웹 서버상의 부하를 분산시키는 방식에 관해서는 다양한 연구결과가 도출되었고, 특히 하드웨어적인 접근방식과 소프트웨어적인 접근방법으로 분류할 수 있으며, 운영체계 기반의 부하분산에 관한 연구도 목격되고 있다.

2.1.1 하드웨어적 부하분산 방식

하드웨어적인 부하분산 방식의 경우, 분산 스위치를 이용하는 방법이 대표적이다. 분산 스위치는 웹서버들의 선두에 위치하여 웹서버로 들어오는 요청을 분산테이블에 따라 각 웹서버에 할당하는 방식이다[3]. 이 방식은 설치는 간단하나 분산스위치 자체의 가격이 상당히 고가라는 단점이 있다. 일반적으로 웹 클러스터 시스템의 구현에서 하드웨어적인 방법을 주저하는 경향이 있으며 이는 비용 관점에서 크게 작용하는 것으로 판단된다.

2.1.2 소프트웨어적 부하분산 방식

소프트웨어적인 부하분산 접근방법 중에서 가장 먼저 사용된 방법으로 라운드로빈DNS(Round-Robin DNS) 방식을 들 수 있다. 이 방식은 DNS 서버에서 하나의 도메인에 대해 여러 개의 IP 주소를 지정하는 방식이다. 이때 각 IP 주소의 웹서버는 동일한 콘텐츠를 보유하고 있으며, DNS 서버는 요청을 각 IP 주소로 분배하는 방식이다. 그러나 라운드-로빈DNS 방식은 요청을 분산시키는 비율을 조절할 수 없다는 단점이 있다. 특히 웹서비스의 경우, 클라이언트 브라우저의 캐쉬로 인해 한 IP로 계속 접속하게되어 서비스가 지속될 경우 부하가 한 곳으로 집중되어 적절한 부하분산을 할 수 없다는 점과 CGI와 같은 공용의 프로그램을 서비스하는 경우, TTL(Time To Live) 초과 시 다른 서버로 연결되어 일관된 작업을 방해할 수 있다는 단점이 지적되고 있다

[1, 2]. DNS의 IP 기반의 부하분산과 다르게 IP 멀티캐스팅을 이용한 부하분산 방식은 특성상 브로드캐스팅보다 대역폭을 절약할 수 있는 장점이 있다. 이 방식은 서비스를 제공하는 IP 주소를 멀티캐스팅 주소로 지정하고 실제 서비스를 수행하는 각각의 웹서버들은 특정 URL 클래스에서 오는 요청만을 처리한다. 그러나 서버와 클라이언트를 연결하는 중간 장비인 라우터가 멀티캐스팅을 지원하지 못할 수도 있다는 점에서 부하분산을 위한 일반적인 응용에 한계가 있음으로 지적되고 있다[6].

2.1.3 운영체계 기반의 부하분산

소프트웨어 기반에서의 부하분산 연구의 결과는 대부분 운영체계와 독립적인 경우가 많다. 그러나 부하분산 알고리즘을 이용하여 특정 웹서버로 부하를 분산시키려는 연구에서는 운영체계와 종속된 연구가 대부분이다. 이 같은 이유는 부하분산 알고리즘을 구현하기 위하여 특정 운영체계에 국한하여 구현하는 것이 상대적으로 쉽기 때문이다. 현재까지의 연구 결과를 보면 크게 NT 혹은 UNIX 계열의 운영체계를 기반으로 하는 방식이 구현되어있다. 전자의 경우 WLBS(Windows NT Load Balancing Service)를 대표적인 예로 들 수 있다. NT 기반의 구현방식은 웹, 프록시 혹은 FTP 같은 TCP/IP 기반 서비스에 대한 부하를 분산시키고, 가용성을 보장한다는 점에서 여타의 웹 클러스터 시스템과 유사하다. 그러나 별도의 부하분산서버(Load Balance Server)를 두지 않고, 웹 클러스터 시스템을 구성하는 각각의 서버가 시스템 내의 서버의 가동여부를 체크하여 서버의 가용성, 생존성을 높여주는 방식으로 정확한 알고리즘은 공개되어 있지 않으나 기존의 특정 부하분산 알고리즘을 채택하여 사용하고 있는 것으로 알려져 있다[4,7].

UNIX 계열에서 대표적인 웹 클러스터 시스템으로는 LVS(Linux Virtual Server)를 예로 들 수 있다. LVS는 WLBS와는 달리, 부하분산서버를 채택한 점이 특징이라 할 수 있다. 부하분산서버는 웹 클러스터 시스템의 서비스 주소를 대표하여 요청을 받아들이고 부하분산 알고리즘을 바탕으로 각 요청을 웹 클러스터 시스템 내의 각각의 서버에게 분산시키는 방식을 적용하고 있다[2]. 그러나 위에서 서술한 모든 방식들은 대부분이 순차적으로 부하를 분산시키는 방식 혹은 접속 수에 한정한 경우가 대부분이다. 즉, 웹 클러스터를 구성하는 각각의 웹서버의 성능은 무시한 정책이 대부분이고 할 수 있다. LVS의 경우, 서버의 성능을 고려하기는 하지만 서버 성능을 서비스 개시 시점에서 경험적으로 판단하여 고정시킨다. 그러므로 서버에 할당되는 요청에 의해 시시각각 시스템 자원의 상태가 변화하는 웹 클러스터 시스템에 단순하고 획일적인 부하분산방식을 적용시킬 경우 더 나은 QoS와 합리적인 부하 및 요청분산이 이루어질지는 의문이다.

2.2 기존 부하분산 알고리즘

기존의 접속 수에 근거한 부하분산 알고리즘을 간략히 소개하면, 다음과 같다[2].

라운드-로빈 스케줄링(Round-Robin Scheduling) 알고리즘은 각 서버별로 부하를 동일하게 분배하는 방식으로 라운드-로빈DNS와 유사하다. 그러나 이 알고리즘은 네트워크의 연결 수에 기반을 두고 있어서 라운드-로빈DNS 보다 효과적이라 할 수 있다. 가중치 기반 라운드-로빈 스케줄링(weighted Round-Robin Scheduling) 알고리즘은 서버간의 서로 다른 처리능력 고려하여 서버마다 부하의 양을 다르게 분산시킬 수 있다. 먼저 각 서버들에 성능에 따른 가중치를 부여하고, 가중치가 높은 서버에 그만큼 더 부하를 할당하는 방식이다. 하지만 부하가 폭주하는 경우, 어느 한 서버에 집중될 가능성이 있다. 이는 부하의 크기를 고려하지 않고 각 요청을 동일하게 간주하므로 단시간에 폭주하는 시스템의 경우 부적합하다고 할 수 있다. 최소 접속 수 스케줄링(Least-Connection Scheduling) 알고리즘은 새로운 서비스 요청이 발생할 경우, 웹 클러스터를 구성하는 각각의 서버 중 현재 가장 적은 연결 수를 가진 서버에 할당하는 방식이다. 가중치 기반 최소 접속 수 스케줄링(Weighted Least-Connection Scheduling - WLC) 알고리즘은 웹 클러스터를 구성하는 서버 각각의 성능에 따라 가중치를 부여한 후, 가중치가 높은 서버에게 현재 연결 중 많은 비율의 연결을 할당하는 방식이다. 즉 n 개의 서버에 대한 각 가중치와 이때 각 서버별 접속 수를 각각 나누어 이 값 중 최소 값을 가지는 쪽에 접속을 할당하는 방식이다. 식 (1)에서 W_i는 n 개의 서버 각각에 대한 가중치를 의미하며, C_i $(i=1,2,..n)$는 서버 i의 접속 수를 의미한다. 마지막으로 S는 웹 클러스터 시스템의 전체 접속 수를 의미한다고 할 때 새로운 서비스 요청이 들어올 경우 n 개의 서버 중 아래의 식을 만족하는 임의의 서버 j 에게 접속을 할당하게된다.

$$\frac{C_j}{W_j} = \min\left\{\frac{\left(\frac{C_i}{S}\right)}{W_i}\right\} = \min\left\{\frac{C_i}{W_i}\right\} \quad (1)$$

WLC 알고리즘이 지금까지 설명한 알고리즘 중 가장 신뢰할 수 있는 알고리즘이라고 할 수 있으나 이 또한 웹 클러스터 시스템 내의 각 서버의 상태정보, 즉 부하가 고려되지 않은 단순하고 획일적인 부하분산 방식에서 벗어나지 못하고 있다. 왜냐하면 한번 부여된 가중치는 각 서버의 현재 부하정도에는 상관없이 고정되기 때문이다.

3. 제안된 부하분산 알고리즘

2. 2절에서 WLC 알고리즘도 부하 분산시 각 서버의 실제 부하 량을 동적으로 참고하지는 않는다고 했다. 이런 단점을 극복할 수 있는 방법으로는 웹 클러스터 시스템을 구성하는 각 서버의 상태정보를 동적으로 체크하여 부하의 크기에 따라 서버의 가중치를 변경함으로서 보완할 수 있다.

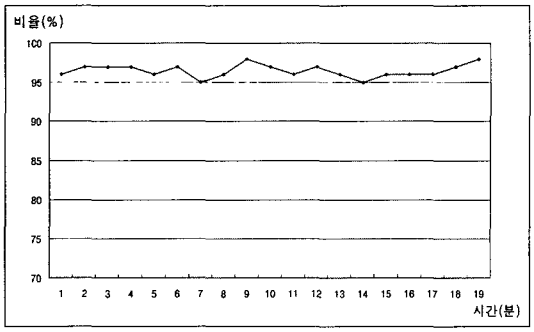

그림 1 CPU 유휴 시간 비율

3.1 서버의 성능 지표

서버의 하드웨어적 성능은 연산 성능의 척도인 CPU와 버퍼링 기능의 메모리, I/O 전달을 담당하는 시스템 버스(BUS), 하드디스크의 데이타 탐색속도 등으로 평가한다. 그러나 웹서비스의 성능의 척도는 CPU의 속도보다는 데이타의 버퍼링 기능을 담당하는 메모리에 큰 영향을 받는다고 한다[5]. 그림 1은 웹 서버에 일정 부하를 주었을 때 CPU의 유휴시간 비율을 나타낸 것이다. 그림 1을 보면 CPU의 유휴시간 비율이 평균 96~97%로 높게 나타남을 볼 수 있어 웹서버에 있어서 CPU가 중요 성능 지표가 될 수 없다고 할 수 있다. 물론 CGI와 같이 CPU 의존도가 높은 웹 응용 프로그램에서 부하가 문제인 경우에는 논외의 대상이 된다. 본 연구에서는 동일한 성능을 가지는 서버로 구성된 웹 클러스터 시스템의 각각의 서버 성능 지표로서, 각 서버가 현재 가지고 있는 가용한 메모리의 양을 기준으로 판단하고자 한다.

3.2 실험환경의 구성

본 연구를 위해 구성한 웹 클러스터 시스템의 실험환경은 동일한 사양의 다수의 서버와 부하분산서버(Load balancer) 1대, 그리고 동시사용자 관점의 부하를 생성하기 위한 웹 부하생성 서버로 구성하였다. 실제 웹 서비스를 하는 서버의 하드웨어 사양은 동일하게 256 MByte 메인 메모리와 Pentium3 800MHz CPU로 구성하였다. 웹 부하 생성 서버는 항상 동일한 수준의 동시사용자를 발생시켜 부하분산서버로 접속시키고, 부하분산서버는 각 요청을 다수의 웹 서버에 할당한다. 웹 클러스터 시스템 구성에 있어서는 각 서버로 할당되는 요청은 웹 클러스터 시스템의 내부 네트워크를 통해 전달된다. 실제 서버에서 요청에 대해 서비스한 결과는 각 서버의 별도 인터넷 연결회선을 통해 클라이언트에게 전달된다. 부하의 생성을 위하여 실제 사용자가 클러스터 시스템에 접속하여 일정시간 동안 주어진 요청을 수행하고 이를 아젠다(agenda) 파일에 기록한다. 여기에는 개별 요청 및 요청과 요청 사이의 공백 시간까지 기록된다. 기록된 아젠다 파일을 토대로 단일 혹은 다수의 부하생성 서버가 부하를 생성한다. 부하의 크기는 한번의 실험 기간 동안 점진적으로 동시사용자 수를 증가시켜 일정시점에서는 동일한 수준의 동시사용자 수준으로 실험을 수행하였다.

3.3 동적 부하분산 알고리즘

본 연구에서 제시하는 동적 부하분산 알고리즘은 주기적으로 각 서버별 부하를 측정한 다음 이를 근거로 새로운 가중치를 계산한다. 그리고 주기적으로 얻어진 새로운 가중치를 가중치 기반 최소 접속 수 스케줄링 알고리즘에 적용하여 각 서버의 부하를 고려하도록 했다.

서버의 부하 정도는 각 서버의 상태정보, 즉 서버가 보유하고 있는 가용한 메모리의 양을 기초로 그 정도를 판단한다. 서버에 접속이 할당되고 서비스를 위해 콘텐츠 및 서버 프로세스를 메모리에 로딩 할 경우, 서버에 설치된 물리적인 메모리를 사용하게 되어 서버의 가용한 메모리는 서비스가 지속되는 동안 계속 소모하게 되고, 서버가 가용한 메모리의 대부분을 소모한 상태에서 추가 접속이 할당될 경우 서버는 메모리 스와핑을 한다. 그 결과 추가적인 하드디스크 접속이 발생하여 시스템의 반응시간은 느려지고 전체 웹 클러스터 시스템의 응답시간을 저하시키게 된다. 그림 2는 각 서버의 가용한 메모리의 양을 바탕으로 부하를 측정하여 다음 주기동안 사용하게 될 가중치를 계산하는 과정이다.

본 알고리즘에서 중요한 결정 변수로 *a, alpha* 그리고 K가 있고 이를 기반으로 가중치 W를 산정 한다. 여기에서 W, 가중치는 웹 클러스터 시스템 내의 각 서버의 부하 정도를 나타내며 상대적으로 부하가 낮을수록 높은 값을 가지게 된다. K는 웹 클러스터 시스템 내에 있는 특정 서버로 1분 동안 접속하는 평균 접속 수를 의미한다. a는 하나의 접속 당 평균적으로 소모되는 메모리의 양을 의미하며, *alpha*는 최대 추가 접속 수를 의미한다. *alpha*는 각 서버의 상태정보 중에서 최대 가용메모리와 최소 가용메모리의 차이를 평균 접속 당 할당 메모리 a로 나눈 값으로, 특정서버에 할당될 추가 접속 수는 그 서버의 가용메모리와 최소 가용메모리의 차이, 그리고 최대 가용메모리와 최소 가용메모리의 차이를

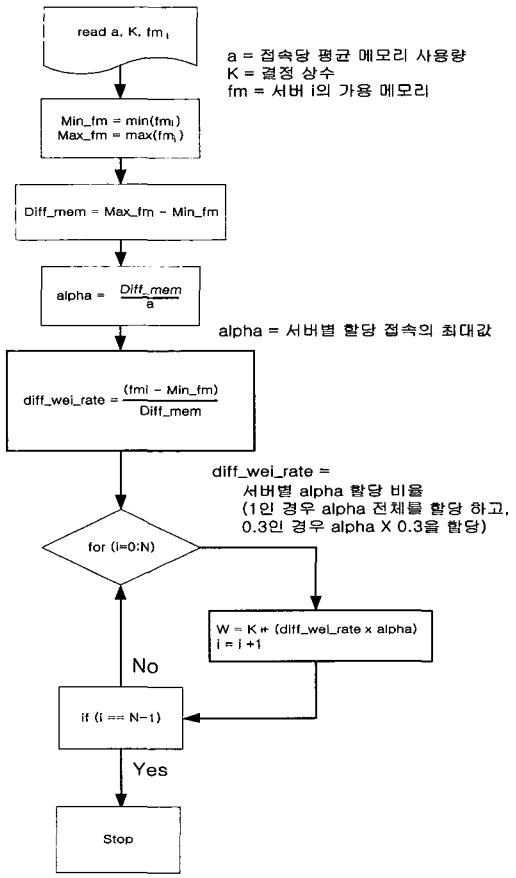

그림 2 동적인 부하분산 알고리즘

선형비율로 산출한 값이다. 따라서 추가 접속 수는 0과 *alpha* 사이의 값이라고 할 수 있다. 가중치 선정과정에서 K는 평균 접속 수로서 웹 부하 생성 서버에서 일정한 동시사용자를 유지하기 위하여 접속을 계속 생성하는데 각 서버로 기본적으로 접속된 분당 평균 접속 수를 의미한다. K의 산정은 부하분산의 주요 요소로서 K 값이 작은 경우는 부하분산이 *alpha*값에 의존적이고, K 값이 큰 경우는 상대적으로 *alpha*값에 덜 의존적이다. 본 연구에서의 부하분산 알고리즘의 구현은 1분 간격으로 서버의 상태정보를 검토하여 최대 *alpha* 개의 접속을 더 할당하고자하는 것이다. K와 *alpha*에 의한 간단한 모의실험을 하였는데, 예를 들어 *alpha*가 50이고 K가 67인 경우는 1분 동안 가용메모리가 많은 서버에 약 50회 정도 더 접속을 부과하고자 하는 의도라고 할 수 있다. 표 1과 표 2는 K와 *alpha*에 의한 접속 과정의 일부를 나타낸 것이다. 접속을 서버 별로 할당하는 기준은 가중치 기반 최소 접속 수 스케줄링(2.2절 참조)에 의거하며, 1분간의 서비스 후 웹 클러스터 시스템 내의 각 서버 별로 부하를 측정하여 새로운 가중치를 각 서버에게 부여하게 되며 그 과정은 그림 2에서 이미 전술되었다. 표 1은 단순히 *alpha* 만을 고려할 경우로써, 분당 300회 접속을 4대의 서버에 할당한 결과를 보여주고 있다. 표 1에서 최종 300번째 접속 후의 결과는 서버 1에 4회, 서버 4에는 164회 접속이 할당되어 초기의 50개 정도의 접속을 서버 4에 더 할당하려는 의도와는 달리 최대 40~50배의 접속이 더 할당되는 결과를 낳게되어 본 모의실험의 의도에 부합되지 않는다. 반면에 표 2는 K와 *alpha*를 동시에 고려한 경우로써, 분당 약 50개에 근접한 값의 접속이 그에 상응한 가중치를 가지는 서버에 차등적으로 할당되고 있음을 보여준다. 이는 본 실험의 목적과 부합되는 좋은 결과라 할 수 있다. 이 결과 K값의 선택이 부하분산의 중요 요소로 부각되고 있어, 이러한 근거를 바탕으로 분 당 접속 회 수(turn around/min)와 동시사용자 값을 이용하여 산정하였다.

표 1 *alpha*만을 가중치로 한 경우

총 접속수	서버별 접속 (예: 현재접속수 / 가중치)				할당 서버
	1	2	3	4	
1	0/1	0/10	0/30	1/50	4
2	0/1	0/10	1/30	1/50	3
3	0/1	1/10	1/30	1/50	2
4	1/1	1/10	1/30	1/50	1
5	1/1	1/10	1/30	2/50	4
. .			. .		. .
296	4/1	33/10	97/30	162/50	4
297	4/1	33/10	98/30	162/50	3
298	4/1	33/10	98/30	163/50	4
299	4/1	33/10	98/30	164/50	4
300	4/1	33/10	99/30	164/50	3

표 2 K+ *alpha*를 가중치로 한 경우

총 접속수	서버별 접속 (예: 현재접속수 / 가중치)				할당 서버
	1	2	3	4	
1	0/67	0/77	0/97	1/117	4
2	0/67	0/77	1/97	1/117	3
3	0/67	1/77	1/97	1/117	2
4	1/67	1/77	1/97	1/117	1
5	1/67	1/77	1/97	2/117	4
. .			. .		. .
296	55/67	64/77	80/97	97/117	4
297	56/67	64/77	80/97	97/117	1
298	56/67	64/77	81/97	97/117	3
299	56/67	64/77	81/97	98/117	4
300	56/67	65/77	81/97	98/117	2

4. 모의 실험 결과

4.1 실험 방법

실험은 웹서비스를 요청하도록 표본으로 프로그램을 작성하였고, 프로그램의 실행은 부하생성서버에서 동시 사용자수를 30, 50, 70, 90, 110, 130, 150으로 증가하면서 웹 클러스터 시스템에 부과실험을 5분간 지속한다. 실험에 사용된 결정상수 K값은 실험에 바탕하여 볼 때 평균적으로 부하 크기의 3 배수에 해당하는 값에 대하여 서버수로 나눈 값을 취하였다. 왜냐하면 사용자 하나에 대하여 여러 개의 프로세스가 발생되고 이들 각 프로세스를 접속수로 하여 계산하였기 때문이다. a의 값 또한 실험에 바탕하여 볼 때 평균적으로 167KByte로 나타났다. 이외의 값 최대 가용메모리 및 최소 가용메모리, *alpha*값은 그림 1에서 전술한 바와 같이 계산되어진다. 서버의 부하를 고려하지 않은 웹 클러스터 시스템(가중치가 모두 동일하게 고정된 웹 클러스터 시스템)과 서버의 부하를 고려하여 가중치가 변화하는 웹 클러스터 시스템에 대하여 반응시간을 평가하였다. 데이타는 1분 주기로 서버의 가용한 메모리의 양과 가중치를 수집하였고 마지막 5분 시점에서 평균 반응시간을 측정하였다. 그리고 실험은 웹 클러스터 시스템의 실제 서버대수와 부하크기에 따라 5회 실험 반복 후 시스템을 재 부팅하고 이를 각 8회 씩 반복하여 수행하였다. 재 부팅의 이유는 매 실험의 이전 부하에 의한 영향을 제거하기 위함이었다. 그리고 웹 클러스터 시스템의 실제 서버의 수는 2대와 4대, 그리고 8대로 구성된 시스템에 대하여 각각 실험을 수행하였다.

4.2 실험 결과 및 분석

기존 알고리즘(가중치기반 최소 접속 수 스케줄링 알고리즘에 서버의 부하를 고려하지 않는 경우)과 제안된 알고리즘(가중치기반 최소 접속 수 스케줄링 알고리즘에 동적인 가중치를 부여하여 부하를 고려한 경우)에 대하여 실험을 수행하여 그림 3과 4는 실제 서버의 수가 4대 일 때 두 실험 군에 대한 메모리 변화를 비교 한 결과의 한 예를 보여준 것이다. 그림 3의 기존 알고리즘은 가중치가 고정되어 있으므로 서버 2와 나머지 3대의 서버들의 시스템 자원, 즉 가용메모리의 차이가 시간이 지남에 따라 극복되지 못하고, 한 서버에 편향되고 있음을 보여주고 있다. 이는 이후의 반응시간 측정에도 상당한 영향을 주리라고 판단된다. 그러나 그림 4의 제안된 알고리즘을 적용한 경우에는 시간이 지남에 따라 전체 서버의 메모리를 균등하게 사용하여 1000 KByte에 접근하는 것을 볼 수 있다. 이러한 경향은 2대 및 8대에 대한 실험에서도 동일하게 목격되었다.

또한 평균 반응시간을 2대, 4대, 8대의 서버로 구성된 웹 클러스터 시스템으로 구분하여 표 3, 표 4와 표 5의 결과를 얻었다.

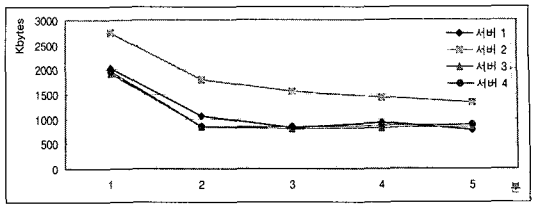

그림 3 기존 알고리즘에 의한 가용 메모리의 변화

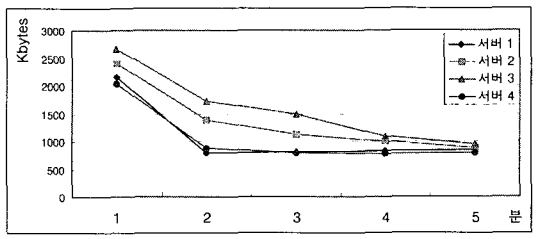

그림 4 제안된 알고리즘의 가용 메모리의 변화

표 3 서버 수가 2대인 경우의 반응시간의 비교

동시사용자 / 실험군	30	50	70	90	110	130	150
기존 알고리즘	0.017	0.040	0.104	0.167	0.273	0.402	0.597
제안된 알고리즘	0.017	0.030	0.091	0.143	0.224	0.343	0.492

표 4 서버의 수가 4대인 경우의 반응시간의 비교

동시사용자 / 실험군	30	50	70	90	110	130	150
기존 알고리즘	0.014	0.016	0.018	0.034	0.047	0.093	0.109
제안된 알고리즘	0.013	0.014	0.017	0.028	0.036	0.075	0.094

표 5 서버의 수가 8대인 경우의 반응시간의 비교

동시사용자 / 실험군	30	50	70	90	110	130	150
기존 알고리즘	0.008	0.012	0.013	0.014	0.015	0.017	0.021
제안된 알고리즘	0.008	0.011	0.012	0.012	0.014	0.015	0.018

실험결과에서 보듯이 제안된 알고리즘이 기존 알고리즘 보다 대체적으로 반응시간이 더 단축되었음을 알 수 있다. 물론 표 3, 표 4와 표 5에서 제시된 값은 각 페이지 안에 포함된 컨텐츠에 관한 순수한 서버의 반응시간에 관하여 조사한 것이므로, 실제 서비스되는 클러스터 시스템을 구현함에 있어서 웹 페이지와, 페이지에 포함되는 컨텐츠의 수가 많아질 경우, 즉 대용량의 컨텐츠를 보유한 클러스터 시스템의 경우에는 기존 알고리즘과 제안된 알고리즘을 이용한 반응시간의 차이는 무시할

수 없는 수치라고 할 수 있다. 또 부하의 크기가 작은 경우 보다 큰 경우에 보다 나은 결과를 보인다는 것도 알 수 있었다. 즉 클러스터링 방법론에 의한 효과는 부하가 큰 웹 클러스터 시스템에 효과적이라는 것이다. 실험을 반복 수행한 결과, 비교 대상 알고리즘들 모두 5회 반복 실험 중 최초 실험에서 측정된 반응시간이 이후의 실험, 즉 2~5회 실험에서 측정된 반응시간보다 부하의 크기에 따라 최소 두 배에서 10배 이상 더 크게 나타남을 볼 수 있었고, 또한 기존 알고리즘과 제안된 알고리즘의 최초 실험들에 대한 반응시간의 비교 결과도 제안된 알고리즘이 더 나은 결과를 보였다. 이런 경향은 각 실험 군에 대한 실험 수행 시 시스템을 재부팅한 결과 첫 반복시 각 서버별로 메모리에 버퍼링된 데이타가 전혀 없었기 때문에 서버의 하드디스크로부터 데이타를 로드하는 시간이 많이 소모되어 시스템의 반응시간이 크게 나타난 것이다. 이 결과로 서버의 가용메모리가 많을 수록 버퍼 공간이 커지기 때문에 많은 데이타를 한꺼번에 저장할 수 있고, 따라서 스와핑 발생 빈도도 그에 비례하여 작아진다고 할 수 있다. 달리 말해 가용메모리가 많은 쪽으로 요청을 더 할당할 수록 실제 서버의 메모리 부족 상황이 발생할 확률은 그만큼 줄어들 수 있다는 의미이다.

5. 결 론

컴퓨터 시스템을 확장하는 대안에는 하드웨어적인 방법론과 소프트웨어적인 방법론이 있다. 그 중에서 소프트웨어적 접근방법으로 클러스터링 방법이 제시되고 있다. 클러스터링 시스템 중에서 적용분야가 웹인 클러스터 시스템은 대부분이 기존에 제공되는 알고리즘에 근거한 부하분산을 하고 있다. 그러나 이러한 방법은 웹 클러스터 시스템을 구성하는 서버 중 시스템의 자원을 가장 많이 소모하는 서버에게도 같은 양의 부하가 계속 부과될 수 있어 전체 웹 클러스터 시스템의 응답시간을 저하시킬 수 있는 요인이 될 수 있다는 관점에서 본 연구를 접근했다. 본 연구에서는 클러스터 시스템의 자원을 균등하게 사용하도록 하는 알고리즘을 바탕으로 하여 응답시간만을 향상시킬 수 있는 방법에 관하여 연구를 수행하고자 하였다. 그러한 많은 원인 중의 몇몇 요소를 선택하여, 이를 시스템 자원으로 산정 하였고 정기적으로 모니터링 하였다. 또한 그 결과를 실시간으로 알고리즘에 반영하여 시스템 즉 서버를 균등하게 이용하고자 하였다. 실험은 인위적으로 부하를 생성하여 모의실험을 통하여 기존의 알고리즘과 비교하였다. 그 결과는 제안된 알고리즘의 적용이 더 나은 결과를 얻을 수 있음을 비교 분석하였다. 본 연구는 동일 사양으로 구성된 웹 클러스터 시스템을 한정하여 실험하였는데, 시스템을 구성하는 요소장비가 동일기종이기 때문이다. 그러나 만약 이기종 사양으로 구성된 웹 클러스터 시스템을 고려한다면 기존의 알고리즘은 적절한 해결 방안을 제시해 주지 못할 것으로 사료되어 제안된 알고리즘의 적용은 더욱 효과적이라 할 수 있다.

참 고 문 헌

[1] Gregory F. Pfister, *In Serach of Clusters Second Edition*, Prentice-Hall PTR, New Jersey, 1998.

[2] Wensong zhang, "The paper Linux Virtual Server for Scalable Network Services," Ottawa Linux Symposium 2000.

[3] Cisco Systems, "Cisco LocalDirector 400 Series Content Switches Relevant Technologies," http://www.cisco.com/en/US/products/hw/contnetw/ps1894, 2002.

[4] Microsoft corporation, "Server Cluster Architecture", http://www.microsoft.com/windows.netserver/techinfo/overview/servercluster.mspx, 2002.

[5] 권원상, 역. 웹 성능 최적화. Patrick Killelea. 한빛미디어, 2000. 8.

[6] Danieł A, *Menace, virgilio* A.F. Almeda, *Capacity* planning for WEB PERFORMANCE Metrics, Models, & Methods, Pretice Hall PTR, New Jersey, 1998.

[7] Microsoft corporation, "FAQ for the WLBS", http://www.microsoft.com/ntserver/productinfo/faqs/faq.asp#1, 2002.

김 석 찬

1999년 계명대학교 산업공학과 졸업(공학사). 2001년 계명대학교 대학원 산업공학과 졸업(공학석사). 2001년~현재 계명대학교 대학원 박사과정재학중. 관심분야는 네트워크 성능분석, 네트워크 시뮬레이션, 네트워크 모델링, 웹 클러스터링

이 영

1983년 고려대학교 산업공학과(공학사) 1885년 고려대학교 산업공학과(공학석사). 1990년 University of Oklahoma 산업공학과(MS). 1994년 North Carolina State University OR/CS(Ph.D). 1995년 삼성 SDS 정보기술연구소 수석연구원. 1998년~현재 계명대학교 산업시스템공학과 조교수. 관심분야는 Network 모델링과 Performance 분석, 웹 서버 capacity planning