웹 서버 클러스터를 위한 효율적인 내용 기반의 부하 분배

(Efficient Content-based Load Distribution for Web Server Clusters)

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

요 약 클러스터 구조는 고가용도와 고성능 그리고 확장성을 요구하는 웹 서비스나 정보시스템 같은 응용 분야에서 저 비용으로 유용하게 사용 가능하다. 웹 서버 클러스터의 내용 기반 분배는 각각의 웹 서버들 사이에서 지능적으로 사용자 요구를 전달하기 위해 애플리케이션 계층에서 알려진 상세한 데이타를 이용한다. 본 논문에서는 웹 서버 클러스터 시스템을 대상으로 캐시의 적중과 각 서버의 부하 상태를 고려한 효율적인 내용 기반의 부하 분배를 수행하고 사용자의 동적인 문서 요구를 수용할 수 있도록 하는 알고리즘을 제안하였다. 특히 제안된 알고리즘은 부하 분배기로 하여금 각 서버에 있는 캐쉬의 내용을 모델링 하기 위한 시도나 웹 문서에 대한 사용자 접근 확률을 계산하기 위한 오버헤드가 없다.

키워드 : 클러스터 시스템, 웹 서버, 부하 분배

Abstract 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. Content-based load distribution for web server clusters uses the detailed data found in the application layer to intelligently route user requests among web servers. In this paper, we propose a content-based load distribution algorithm that considers cache hit and load information of the web servers under the web server clusters. In addition, we expand this algorithm in order to manage user requests for dynamic file. Specially, our algorithm does not keep track of any frequency of access information or try to model the contents of the caches of the web servers.

Key words : Cluster Systems, Web Server, Load Distribution

1. 서 론

최근 애플리케이션들은 한 대의 컴퓨터가 처리할 수 있는 능력 이상을 요구하고 있으며 이를 해결하는 방법 중 하나는 프로세서와 메모리를 비롯한 다른 구성요소들의 속도를 향상시키는 것이나 이 역시 빛의 속도에 의해 제한을 받는다. 이에 대한 실행가능하고 비용 효율적인 대안은 다수의 컴퓨터를 연결하여 서로 협동함으로써 컴퓨팅 능력을 향상시키는 것이다[1].

특히 고성능 계산 능력과 대규모 웹 서비스의 요구가 최근 증가함에 따라 고성능 서버에 대한 요구가 더욱 커지고 있다. 이러한 요구는 고성능 마이크로프로세서, 고속 네트워크가 보편화되기 이전에는 고가의 슈퍼컴퓨터로 해결하였으나 비용 효율적인 방법으로 제시된 기술이 클러스터 시스템이다[2].

클러스터 시스템은 단일 시스템으로 관리되는 독립적인 서버 그룹으로 여러 대의 컴퓨터를 서로 연결하여 마치 하나의 컴퓨터처럼 사용하는 것이다[3]. 따라서 사용자와 시스템 관리자에게 개별 컴퓨터의 그룹이 아니라 하나의 시스템처럼 보인다.

클러스터 시스템은 대량생산되는 제품을 사용하기 때문에 가격 대 성능비가 우수하며 무료 소프트웨어를 사용할 경우 기존의 동급 고성능 컴퓨터에 비해 현저하게 가격을 낮출 수 있다[4]. 특히 고속 네트워크 장비의 개발로 인해 속도 문제가 해결되었기 때문에 현실적으로 실현 가능하다는 장점을 지니고 있으며 고성능과 고가

· 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2004-003-D00282)
† 종신회원 : 명지전문대학 컴퓨터정보과 교수
abback@mail.mjc.ac.kr
†† 종신회원 : 아주대학교 정보및컴퓨터공학부 교수
sskim@ajou.ac.kr
논문접수 : 2003년 9월 26일
심사완료 : 2004년 9월 22일

용도를 요구하는 분야에서 결함 허용 컴퓨터의 대안으로 등장하고 있다[5,6]. 또한 최근에는 웹 서비스와 연계하여 많은 연구가 진행되고 있다[7,8].

지난 수년간 많은 대학과 연구기관에서 웹 서버 클러스터에 사용자의 요구를 분산시키기 위한 연구가 수행되어 왔으며 이러한 연구들은 Client-side 접근 방식, DNS 기반의 접근 방식, TCP 연결 라우팅 방식, 그리고 HTTP 리다이렉션(redirection)으로 분류될 수 있다[9].

Client-side 접근 방식은 클라이언트 측에서 다운로드되고 실행될 수 있는 자바 애플릿을 통해 수행되는 방식으로 자바 애플릿에 의해 다수의 URL로 변환될 수 있으나 보안 문제 때문에 요구 스케줄링을 수행하기에 적합하지 않다. DNS 기반의 접근 방식은 DNS 서버에 의해 동일한 이름을 여러 IP 주소로 변환하여 서비스하는 방식으로 다른 방식에 비해 비교적 간단하여 초기에 널리 사용되었으나 확장성의 문제와 서비스 노드의 결함을 반영할 수 없는 단점이 있다. TCP 연결 라우팅 방식은 가장 널리 사용되는 방식으로 단일 IP 주소로 외부에 알려지며 부하 분배기가 사용자의 요구를 받아 적당한 서비스 노드에게 전달하는 방식이다. 그러나 DNS 기반의 접근 방식과 TCP 연결 라우팅 방식은 클라이언트의 HTTP 요구가 보내지기 전에 서비스 노드를 결정하기 때문에 내용 기반의 라우팅을 할 수 없다. 내용 기반 분배는 각각의 웹 서버들 사이에서 지능적으로 사용자 요구를 전달하기 위해 애플리케이션 계층에서 얻려진 상세한 데이타를 이용하는 방식이다.

HTTP 리다이렉션에서 부하 분배기는 클라이언트가 요구한 데이타를 보내는 대신에 선택된 서버의 IP 주소를 보내고 클라이언트는 그 주소로 다시 요구를 보내는 방식으로 동작한다. 이와 같은 방식은 내용 기반의 라우팅을 위해 사용될 수는 있으나 모든 스케줄된 요구에 대하여 라운드트립(round-trip) 지연이 발생하고 서비스 노드의 IP 주소가 노출되는 단점이 있다.

Layer-4 부하 분배 알고리즘은 TCP/IP 레벨에서 동작하며 HTTP 요구가 보내지기 전에 TCP/IP 연결 설립이 이루어지기 때문에 서비스 노드에 대한 선택이 요구된 내용에 의해 결정될 수 없다. 이에 비하여 Layer-7 부하 분배 알고리즘은 서비스 노드를 결정하기 전에 HTTP 요구를 분석할 수 있기 때문에 클라이언트 요구의 내용에 대하여 상세한 정보를 고려한 분배가 가능하다. 즉 OSI Layer-7에 포함된 정보를 이용하며 클라이언트가 요구한 내용에 기반을 두어 동작하기 때문에 내용 기반의 라우팅으로 알려져 있다.

이러한 내용 기반 라우팅은 부하 분배기에 추가적인 프로세싱 오버헤드를 초래하지만 이를 해결하기 위한 연구가 수행되고 있으며 더욱이 내용기반 라우팅만이 가질 수 있는 장점으로 인해 현재 웹 서버 클러스터에 대한 연구는 내용 기반 클러스터링의 영역으로 진행 중에 있다[10].

내용 기반 요구 분배의 잠재적인 장점으로는 서비스 노드의 메인 메모리 캐쉬에서 향상된 적중률을 가질 수 있는 것과 디스크 같은 기억 장치의 확장성 그리고 특정 목적의 서비스 노드 운영 등을 들 수 있다.

이와 같은 내용 기반 요구 분배 방식의 핵심 요소는 사용자의 요구가 서비스 노드가 결정되기 전에 검사되도록 하는 것이며 이를 구현하기 위한 방법으로 TCP hand-off[11]와 TCP splicing[12] 방식이 존재하고 그 중 TCP hand-off 방식이 더 좋은 성능을 나타내는 것으로 증명되었다[13].

본 논문에서는 웹 서버 클러스터를 대상으로 효과적인 내용 기반의 부하 분배 알고리즘을 제안하고 사용자의 동적 문서 요구를 수용할 수 있도록 이를 확장하였다.

논문의 구성으로 2장에서는 관련연구를 알아보고 3장에서는 내용 기반의 부하 분배 알고리즘을 제안한다. 4장에서는 시뮬레이션을 통해 기존의 방법들과 성능을 비교하였으며 5장에서 결론을 내린다.

2. 관련연구

웹 서버 클러스터의 부하 분배 방식은 Layer-4 switching with layer-2 packet forwarding(L4/2), Layer-4 switching with layer-3 packet forwarding(L4/3), Layer-7 switching with layer-2 packet forwarding(L7/2), Layer-7 switching with layer-3 packet forwarding(L7/3)의 네 가지로 분류되기도 한다[10]. L4/2에 관한 연구로는 Bell 연구소의 ONE-IP[14], IBM의 eNetwork Dispatcher[15] 등이 있으며 L4/3에 관한 연구로는 버클리 대학의 Magicrouter[16]와 시스코 시스템의 LocalDirector[17] 등이 존재한다. 현재까지는 이와 같은 Layer-4의 부하분배기에 대한 연구가 주류를 이루었으나 점차 IBM의 Web Accelerator[18] 같은 Layer-7에 관한 연구가 수행되고 있는 실정이다[19,20].

Layer-7을 이용하는 부하 분배 방식은 Layer-4의 리다이렉션을 이용하는 방식과 달리 클라이언트에게 IP 주소의 노출 없이 보안을 유지하는 것이 가능하며 라운드트립 지연이 발생하지 않는 장점을 가지고 있다[21].

LARD(Locality-Aware Request Distribution)는 웹 서버 클러스터에서 내용 기반 라우팅에 대한 초기의 연구로 관련 분야의 연구에 많은 영향을 미치고 있다[11]. 이는 내용 기반 요구 분배의 장점 중 특히 서비스 노드

의 메인 메모리 캐쉬 적중률을 향상시키는 데에 초점을 맞추었다.

LARD에서 부하 분배기는 각 웹 문서에 대하여 서비스 노드와 일대일 대응을 유지하며 주어진 문서에 대하여 첫 번째 요구가 도달하면 가장 적은 부하가 걸려 있는 서비스 노드에 우선적으로 할당한다. 그 후 이 문서에 대한 요구가 도착되면 이전에 할당되었던 서비스 노드로 보내지나 만일 그 노드에 임계 값을 넘는 과부하가 걸려 있으면 다른 노드 중 부하가 가장 작게 걸린 노드를 선택하여 그 문서에 대한 새로운 서비스 노드로 할당한다.

즉 기존의 동적 부하 알고리즘은 사용자 요구가 들어왔을 때 무조건 가장 적은 부하를 가진 노드를 할당하는데 비해 LARD는 캐쉬 적중률 향상을 위해 지나친 과부하가 걸리지 않는 이상, 이전의 서비스 노드를 할당하는 정책을 취한다.

하버드 대학의 HACC(Harvard Array of Clustered Computers)는 LARD가 FreeBSD를 이용한 반면 Windows NT 하에서 구현되었다[22]. 부하 분배 알고리즘의 기본 개념은 LARD와 유사하며 서비스 노드의 부하를 측정하기 위해 다음의 공식을 이용한다.

$$load = weight_{CPU} \times load_{CPU} + weight_{disk} \times load_{disk}$$

이와 같은 방식으로 HACC에서는 정적 파일을 대상으로 하는 웹 서버 시스템에서는 $weight_{CPU}$를 0, $weight_{disk}$를 1로 하여 부하를 계산하고 동적 파일이 있는 경우 weight 값을 조절하여 측정할 수 있도록 하였다.

또한 CAP(Client-Aware Policy)은 서비스 노드의 부하를 고려하지 않고 단지 사용자의 요구만 가지고 부하를 분배하였다[23]. 이 방식에서 사용자의 요구는 정적 서비스, CPU bound 서비스, 디스크 bound 서비스, CPU와 디스크 bound service의 네 가지 클래스로 분류되어 서비스된다. 즉 사용자 요구가 도착하면 부하 분배기는 URL을 분석하고 각 요구가 속해있는 서비스 클래스를 고려함으로써 적당한 서버를 선택한다. 만일 노드 A와 B가 동일한 수의 사용자 요구를 받을 경우에도 A에는 CPU bound 서비스, B에는 디스크 바운드 서비스만 요구된다면 비효율적인 서비스가 이루어진다. 따라서 CAP에서는 각각의 요구 클래스를 고려한 부하 분배를 수행함으로써 성능을 향상시켰다.

분배 알고리즘으로 LARD는 정적인 파일 서비스만을 대상으로 하였으며 시스템마다 다른 복잡한 파라미터 값을 시스템 관리자가 설정해야 하는 어려움이 있다. HACC에서는 동적인 파일 서비스를 위한 시도로서 부하에 관련된 metric을 정의하여 사용하였으나 시스템 관리자가 각각의 작업부하(workload)에 대하여 weight 값을 설정해 주어야 하는 문제점이 있다. 이는 시스템 관리자로 하여금 부하 분배 알고리즘에 대한 전문적인 지식을 요하며 최적의 weight 값을 구하는 것과 관련된 연구가 추가로 수행되어야 한다. CAP은 웹 서버 노드의 부하를 측정하지 않고 사용자의 요구 내용만 고려하여 부하 분배를 수행하나 서비스되는 각각의 파일마다 특정한 자원에 미치는 영향이 정의되어야 한다.

또한 이들 모두는 이질적인 환경을 고려하지 않았으며 각 서비스 노드가 동일한 집합의 문서와 서비스를 수행하는 것을 가정하였다.

특히 최근에는 단순한 정적 파일 요구와 함께 CPU 자원을 이용하는 동적 파일에 대한 요구가 증가하는 추세이며 서비스 요구 증가에 따라 하드웨어를 추가할 경우 빠른 개발 주기로 인해 기존의 하드웨어보다 높은 사양의 노드가 추가되므로 이질적 환경에 대한 연구는 고려되어야 할 부분이라 할 수 있다.

따라서 본 논문에서는 웹 서버 클러스터 상에서 효율적인 내용 기반의 부하 분배를 수행하고 사용자의 동적인 문서 요구를 수용할 수 있도록 하며 각 노드가 동일한 파일의 집합을 갖지 않아도 되는 환경에서 응용 가능한 알고리즘을 제안하였다.

본 연구에서 수행하는 내용 기반의 라우팅은 L7/2에 속하며 이는 L7/3과 달리 부하 분배기는 사용자의 요구를 서비스 노드에 전달만 해 주고 서비스 노드가 직접 사용자에게 응답하는 과정을 거친다.

3. 부하 분배 알고리즘

본 논문에서 고려하는 웹 서버 클러스터의 구조는 사용자 요구를 분배하기 위한 부하 분배기와 요구된 문서를 서비스 하는 다수의 노드가 고속의 네트워크로 연결되어 있는 시스템으로 그림 1은 이를 보여주고 있다.

특히 내용 기반의 부하 분배는 기존의 방식과 달리 파일 A, B, C에 대한 요구가 각 노드마다 할당되어 파일 A를 요구하는 사용자는 첫 번째 노드로 전달하고 파

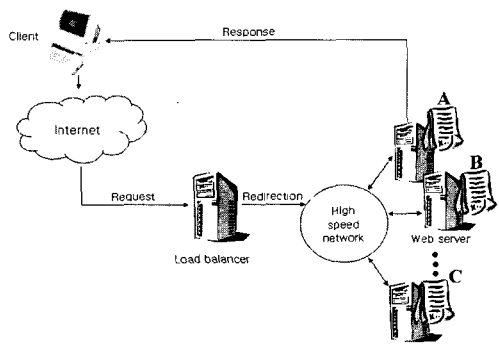

그림 1 웹 서버 클러스터 구조

일 B를 요구하는 사용자는 두 번째 노드에 전달하는 방식으로 캐쉬 적중률의 향상을 가져올 수 있다.

관련연구에서 언급한 LARD는 지나친 부하 불균형이 발생하지 않는 한 기존에 연결되었던 경험이 있는 노드를 할당함으로써 캐쉬 적중률을 높이는 방식을 이용하였으나 이는 적중률 향상을 위해 특정 노드의 부하가 심각한 상황까지 도달해야 한다. 여기서 지나친 불균형의 의미를 명확히 하기 위해 노드가 유휴(idle) 상태의 자원을 가질 것 같은 활성화된 연결의 수 TL과 사용자 요구를 서비스 하는데 잠재적인 지연의 원인이 될 것 같은 연결의 수 TH를 정의하였다.

LARD는 해당 노드가 임계 값 TH를 넘고 다른 모든 노드가 임계 값 TL 이하이면 새로운 노드를 할당한다. 즉 다른 노드들이 한가하더라도 두 개 이상의 노드가 임계 값 TL 이상이면 해당 노드가 임계 값 TH의 두 배 이상이 될 때까지 새로운 노드를 할당하지 않는 문제점이 있다. 이 조건은 간단히 그림 2의 예를 고려해 보아도 비효율적임을 알 수 있다. 즉 노드 2와 6의 부하가 임계 값 TH(65)를 넘고 다른 노드들이 TL(25) 이하인 경우에서 노드 2에 저장된 요구가 들어오면 LARD는 다른 모든 노드가 임계 값 TL 이하가 아니므로 새로운 노드를 할당받지 않는다. 이에 비하여 제안된 알고리즘 CUL(Content-based Useful Load distribution)은 동일한 경우에 대해 거의 요구가 없는 노드 1을 할당 받아 서비스 집합에 추가시키므로 부하가 분산됨을 볼 수 있다.

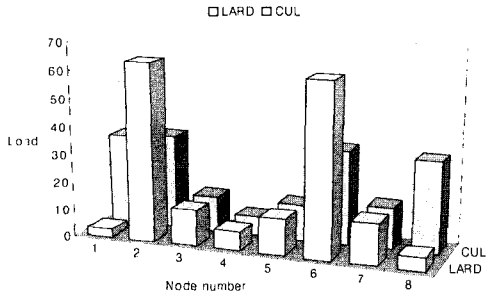

그림 2 LARD와 제안된 알고리즘의 비교 예

물론 부하 불균형이 성능의 저하를 의미하지는 않으며 불균형 정도가 심해지더라도 캐쉬 적중률을 높여 단위 시간당 사용자 요구 처리량을 높일 수 있음을 기존의 연구들이 보여주고 있다. 사용자의 HTTP 요구가 보내지기 전에 연결 설립을 하는 방식에서는 최대한 각 노드들의 부하를 동일하게 유지하기 위한 알고리즘들이 소개되었으나 내용 기반 부하 분배에서는 캐쉬 적중률을 우선으로 고려한다.

본 논문에서 제안하는 다음의 알고리즘은 이와 같은 기존 방식들의 하이브리드(hybrid)로서 부하 균등이나 캐시 적중률 향상의 어느 한 쪽에 치우치지 않고 특정 조건을 만족할 때까지는 캐쉬의 적중률을 고려하나 그 이상이 되면 부하가 적게 걸린 노드들을 서비스 집합에 참여시키는 방식을 취한다.

Load Distribution Algorithm : CUL

```
1 while (true)
2   fetch next request R
3   if (ServerSet[R.target] = NULL)
4       L, ServerSet[R.target] ← least loaded node
5   else
6       L ← least loaded node in ServerSet[R.target]
7       M ← most loaded node in ServerSet[R.target]
8       A.load ← average load of nodes in ServerSet[R.target]
9       E.load ← average load of nodes except ServerSet[R.target]
10      if (A.load > 2TH)
11          if (∀ node with load < TH || E.load < 2TH)
12              L, ServerSet[R.target] ← least loaded node
13      else if (A.load > TH)
14          if (∀ node with load < TL || E.load < TH)
15              L, ServerSet[R.target] ← least loaded node
16      else if (A.load > TL && E.load < TL)
17              L, ServerSet[R.target] ← least loaded node
18      if ( |ServerSet[R.target]| > 1)
19          if (A.load ≤ TL)
20              remove M in ServerSet[R.target]
21          else if (A.load ≤ TH && ∀ node with load > 2TH)
22              remove M in ServerSet[R.target]
23          else if (A.load ≤ 2TH && E.load > 2TH)
24              remove M in ServerSet[R.target]
25  send R to L
```

즉 사용자의 첫 번째 요구가 들어오면 이는 현재 가장 적게 부하가 걸린 노드에 할당되며 이를 서비스 노드의 집합에 추가한다. 계속해서 들어오는 요구에 대해서는 특정 조건을 만족시키지 않는 한 기존에 할당된 노드로 전달되며 만일 기존에 할당된 노드의 집합이 존재하면 집합 내에서 가장 부하가 적게 걸린 노드를 선택하여 전달한다.

여기서 특정 조건이란 해당 파일의 서비스 노드 집합의 평균 부하 A.load가 2TH, TH 그리고 TL보다 클 각각의 경우마다 다르다. 예를 들어, TH를 넘는다면 TL 이하의 노드가 존재하거나 다른 모든 노드들의 평균 부하가 TH 이하일 경우 등 알고리즘에서 제시하는 바와 같다.

알고리즘의 조건에 해당하는 경우에는 현재 남아 있는 노드들 중 제일 적은 부하의 노드를 선택하여 해당 파일의 서비스 집합에 참여하게 하고 현재의 요구를 전달한다. 이와 같이 파일의 서비스 집합을 이용하지 않으

면 극단적인 경우 하나의 인기 파일이 존재하고 대부분의 사용자 요구를 받는 상황 발생 시 심각한 성능 저하가 생긴다. 즉 하나의 노드가 할당되고 한계에 도달하면 또 다른 노드가 그 요구를 위해 할당된다. 이 역시 한계에 도달하면 또 다른 노드가 할당되는 악순환이 반복된다. 이는 다른 모든 노드들이 유휴 상태이더라도 캐쉬 적중률을 우선으로 고려하여 하나의 노드만 집중적으로 요구를 할당 받는 알고리즘의 문제를 단적으로 보여준다.

그러나 서비스 집합을 이용하게 되면 사용자 요구가 특정 파일에 집중되더라도 하나의 노드가 하나의 파일만 담당하는 알고리즘들과 달리 서비스 노드가 계속해서 바뀌는 현상이 발생하지 않고 모든 노드들이 거의 동일하게 참여하게 된다. 이는 동적으로 부하를 측정하여 부하가 가장 적은 노드에 할당하는 방식과 유사하게 동작한다.

한편 특정 파일의 인기도가 충분히 하락하면 더 이상 다수의 노드가 필요하지 않으므로 서비스 집합이 하나 이상일 경우 불필요하다면 이를 제거하는 코드가 라인 18에서 24까지 삽입되었다.

제안된 알고리즘은 부하 분배기로 하여금 서비스 노드에 있는 캐쉬의 내용을 모델링 하기 위한 시도나 웹 문서에 대한 사용자 접근 확률을 계산하기 위한 오버헤드가 없으며 특히 서비스 노드에 의해 사용되는 페이지 교체 정책과 독립적인 특성을 지닌다. 또한 모든 노드가 잠재적인 서비스 대상이 될 가능성이 있어 모든 노드의 디스크에 동일한 집합의 파일들을 저장해야 하는 시스템들과 달리 서비스 집합의 수를 제한하거나 필요시 복사하는 방식을 이용하면 디스크 용량 및 비용을 다소 절감하는 효과를 거둘 수 있다.

최근 부하분배 알고리즘과 관련한 연구의 대부분은 주로 정적 파일을 대상으로 하고 있으며 동적 파일 연구가 시도되고 있으나 추가적인 연구가 필요한 상황이다.

따라서 우리는 동적 파일에 대한 요구를 효과적으로 수용하기 위하여 CUL을 확장한 알고리즘 DCUL을 개발하였으며 이는 파일에 대한 사용자의 요구를 서버 자원에 미치는 영향에 따라 분류한다. 즉 메모리에 캐쉬가 가능한 정적 파일 요구, 웹 트랜잭션(transaction)과 같은 디스크 관련요구, CPU 관련 요구, CPU와 디스크를 모두 이용하는 요구로 분류하여 사용자 요구를 할당한다.

Load Distribution Algorithm : DCUL

```
1  while (true)
2    fetch next request R
3    if (R.class = Static)
4       execute CUL algorithm line 3 to 24
5    if (R.class = CPU)
6       L ← node that has the smallest C_count
7       L.C_count = L.C_count + 1
8    if (R.class = Disk)
9       L ← node that has the smallest D_count
10      L.D_count = L.D_count + 1
11   if (R.class = Disk & CPU)
12      for each node
13         if (C_count > D_count)
14            DC_count = C_count
15         else
16            DC_count = D_count
17      L ← node that has the smallest DC_count
18      L.C_count = L.C_count + 1
19      L.D_count = L.D_count + 1
20   send R to L
```

이는 어느 노드는 CPU 관련 요구만 처리하고 어느 노드는 디스크 관련 요구만 처리할 경우 발생하는 비효율성을 제거한다. 관련연구에서 언급된 CAP 알고리즘의 경우 이 정보만을 이용하여 부하 분배에 이용하였다.

위 알고리즘에서 사용자 요구 R은 각각의 클래스로 우선 분류된 후 정적 파일이면 메모리 캐쉬의 장점을 이용하는 CUL 알고리즘을 이용한다. CPU나 디스크 관련 요구이면 각 노드들이 가능한 균등하게 배분되도록 해당 클래스에서 가장 적은 요구 수를 가지는 노드를 선택한다.

또한 CPU와 디스크를 모두 이용하는 요구이면 각 노드에서 C_count와 D_count 중 큰 값을 선택하여 그 값이 적은 노드를 선택한다. 이와 같은 방법을 이용하는 이유를 설명하기 위해 그림 3을 보자. 디스크 요구가 가장 적은 B나 CPU 요구가 가장 적은 C보다는 노드 A에 할당되는 것이 더 빨리 서비스될 가능성이 높다.

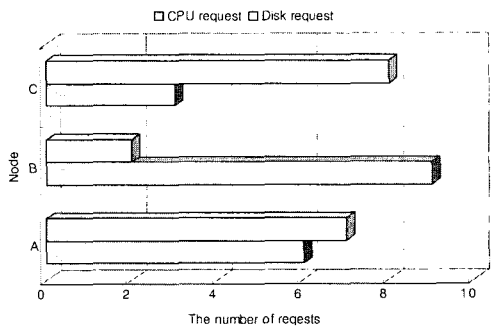

그림 3 CPU와 디스크를 모두 이용하는 요구의 할당 예

4. 성능 평가

이 장에서는 3장에서 제안된 알고리즘의 성능을 평가하기 위해 시뮬레이션을 수행하였으며 그 결과를 보여준다. 시뮬레이션 모델에서 각 노드들은 자신의 CPU,

메모리, 디스크를 가지고 있으며 실험의 간소화를 위해 캐싱은 파일단위로 수행되고 부하 분배기와 네트워크는 클러스터의 성능을 제한하지 않는 것으로 가정하였다.

시뮬레이션을 위해 사용된 파라미터 값은 표 1과 같으며 [19]와 [21]을 주로 참고하였다. 시뮬레이션 모델에서 사용자 요구의 도착시간 간격은 지수분포를 따르는 것으로 가정하였으며 부하분배기의 큐에 도달하여 서버 노드를 할당받은 후 메모리에 존재하면 바로 서비스되고 그렇지 않으면 디스크를 접근하게 된다. 각 노드에서의 메모리 교체정책은 LRU(Least Recently Used)를 가정하였으며 임계 값 TH와 TL은 각각 사용자 연결 수 65와 25를 가정하였다. 또한 웹 문서 접근 확률 분포는 Zipfian 분포를 이용하였는데 이는 일반적으로 웹 문서 접근 확률 분포에 사용되는 방식으로 주로 0에서 2의 skew 값이 사용된다[24].

표 1 시뮬레이션 파라미터

Parameter	Value
Number of nodes	1 10
Memory per node	64MB
Memory transfer rate	100MB
Disk transfer rate	10MB
Seek time	9ms
Rotational latency	8ms
File size	64KB
Connection establishment and teardown	1.5ms
Arrival rate	5000/sec
Zipfian Distribution	skew=1.5

그림 4는 각 서비스 노드에 연결된 사용자 수가 제일 적은 것을 우선적으로 할당하는 WLC(Weighted Least Connection), LARD 그리고 제안된 알고리즘 CUL의 처리량을 비교하여 보여준다. 처리량은 웹 서버 클러스터 전체 노드에서 초당 처리된 사용자 수를 이용하였으며 이는 부하 분배 알고리즘의 성능을 평가하기 위한 척도로 주로 사용된다.

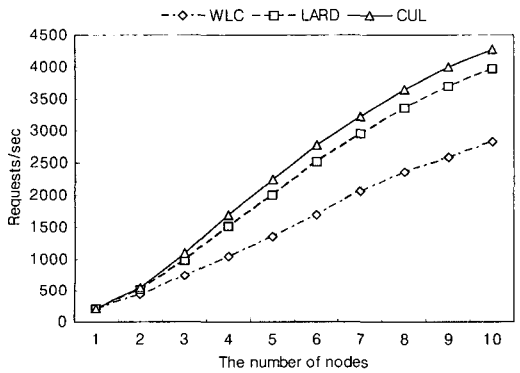

그림 4 노드 증가에 따른 처리량 비교

이 실험에서는 정적 파일만을 대상으로 하였고 그림에서 알 수 있듯이 노드 수의 증가에 따라 처리량은 증가하며 캐쉬 적중률을 고려하지 않는 WLC가 가장 낮은 성능을 나타내고 있다. CUL은 캐쉬 적중률과 부하 분산을 상황에 따라 효과적으로 고려하기 때문에 캐쉬의 적중에 주로 초점을 맞추어 심각한 부하 불균형의 가능성을 가지고 있는 LARD의 처리량보다 높은 것으로 나타난다.

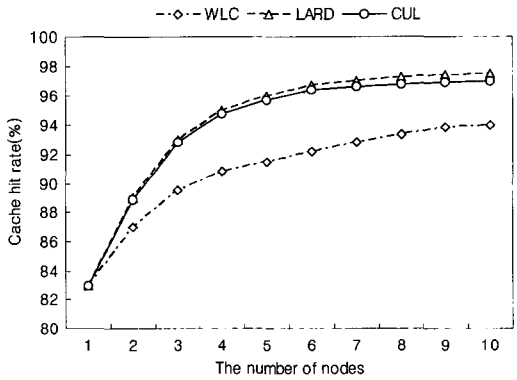

그림 5 노드 증가에 따른 캐쉬 적중률

그림 5는 각 알고리즘의 캐쉬 적중률을 보여주고 있다. 이 그림에서는 가장 우수한 처리량을 보였던 CUL의 적중률이 LARD 보다 다소 낮은 것으로 나타났는데 이는 LARD의 경우 CUL에 비해 캐쉬 적중률을 우선으로 하기 때문이다. 즉 CUL은 부하가 많이 걸려 있는 노드에서 캐쉬 적중을 시도하기보다는 현재 쉬고 있는 노드들을 이용해 부하를 분산시키므로 캐쉬 적중률 측면에서 LARD보다 떨어진다. 그러나 전체적인 측면에서 볼 때 부하 분산에 대한 효과가 캐쉬 적중을 하면서도 높은 부하로 인한 지연의 효과 보다 상대적으로 크기 때문에 우수한 처리량을 보인다.

또한 위 그림에서 알 수 있듯이 WLC는 가장 낮은 캐쉬 적중률을 보여주고 있다. 이는 WLC의 경우 캐쉬의 적중을 전혀 고려하지 않고 현재 각 노드에 걸린 부하만을 고려하기 때문이다. 이와 같은 방식을 취할 경우 각 노드에 걸린 부하가 거의 균등하게 되지만 빈번한 디스크 접근 때문에 오히려 전체적은 성능은 떨어지게 된다. 이와 같은 결과들은 캐시 적중률이나 부하 분배를 균등하게 하는 것이 곧 성능의 향상을 의미하지 않는다는 기존의 실험 결과들과 일치한다.

한편 위 실험은 정적 파일만을 대상으로 하였으나 CUL을 동적 파일 처리를 위해 확장한 DCUL의 성능을

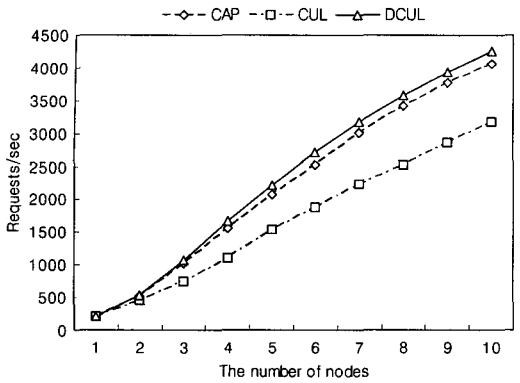

그림 6 노드 증가에 따른 처리량: 동적 파일(15%)

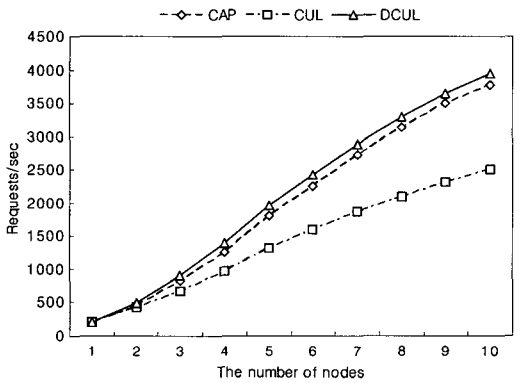

그림 7 노드 증가에 따른 처리량: 동적 파일(30%)

관찰하기 위하여 동적 파일을 위한 실험이 추가로 수행되었다. 그림 6은 사용자 요구의 15%가 동적 파일인 경우이며 그림 7은 요구의 30%가 동적 파일로 구성되어 있다. 동적 파일 중에서는 디스크 관련 요구, CPU 관련 요구 그리고 CPU & DISK 관련 요구가 각각 3분의 1씩 구성되도록 하였다.

CAP 방식의 경우 각 노드들의 부하를 고려하지 않고 사용자 정보만을 이용하여 동적 파일에 대한 요구를 처리하나 그림 6과 7에서 알 수 있듯이 동적 파일에 대한 고려를 하지 않은 CUL보다는 높은 성능을 나타내었다. 하지만 사용자 정보와 서버들의 정보를 동시에 이용한 DCUL보다는 다소 낮은 처리량을 보임을 알 수 있다. 이는 DCUL의 경우 사용자의 정적 요구에 대해서는 캐시의 적중률을 고려하고 동적 요구에 대해서는 자원의 이용을 고려하기 때문이다.

또한 동적 파일의 양을 15%로 한 그림 6과 30%로 증가시킨 그림 7을 비교하여 볼 때 DCUL과 CAP은 영향을 적게 받으나 CUL의 경우 보다 많은 성능의 저하를 보인다.

5. 결 론

클러스터 시스템은 가격 대 성능비가 우수하며 무료 운영체제를 이용할 경우 동급의 고성능 컴퓨터에 비해 상당히 가격을 낮출 수 있다. 현재 성공적으로 평가받고 있는 인터넷과 웹을 이용한 서비스가 크게 확산되고 있고 이에 대한 많은 성장 가능성이 존재하지만 제한된 대역폭과 서버 측 용량의 한계로 인한 문제들이 해결되어야 한다.

인터넷과 웹에 대한 사용자의 요구가 급격히 증가하는 추세에서 내용 기반의 부하 분배를 이용한 웹 서버 클러스터의 개발은 비용 효율적인 시스템 개발로 인해 웹 서비스를 제공하는 기업에게는 비용 절감을, 서비스를 받는 사용자에게는 응답시간의 향상을 통한 만족을 동시에 이룰 수 있을 것으로 기대된다.

이와 같은 상황에서 본 논문에서는 웹 서버 클러스터 시스템을 대상으로 캐시의 적중과 각 서버의 부하 상태를 고려한 효율적인 내용 기반의 부하 분배를 수행하고 사용자의 동적인 문서 요구를 수용할 수 있도록 하는 알고리즘을 제안하였다.

한편 Layer-4 부하 분배 방식은 Layer-7에 비하여 구현이 용이하고 비용 측면에서 유리하므로 Layer-7에 의해 완전히 대체되기보다는 애플리케이션에 따라 적용되어 사용될 것으로 예상된다.

향후 이 분야에서 클라이언트와 서버의 정보를 효과적으로 묶는 방법들이 추가적으로 연구되어야 하며 이질적인 시스템에서의 적용 및 확장성에 대한 연구가 이루어져야 한다.

참 고 문 헌

[1] R. Buyya, High Performance Cluster Computing: Architectures and Systems, Chapter 1, Prentice-Hall, 1999.

[2] J.Y. Chung and S. Kim, "Efficient Memory Page Peplacement on Web Server Clusters," Lecture Notes in Computer Science, Vol. 2331, pp. 1042-1050, Apr. 2002.

[3] G. Hunt, E. Nahum, and J. Tracey. Enabling Content-based Load Distribution for Scalable Services. Technical report, IBM T.J. Watson Research Center, May 1997.

[4] 유찬수, "리눅스 클러스터링", 정보과학회지, 제 18권, 제2호, pp. 33-39, 2000. 2.

[5] V. Cardellini, M. Colajanni and P.S. Yu, "Dynamic Load Balancing on Web-server Systems," IEEE Internet Computing, pp. 28-39, May 1999.

[6] 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.
[7] 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.
[8] 김성수, 정지영, "웹서버 클러스터를 위한 효율적인 부하분배 알고리즘", 한국정보과학회논문지(정보통신), 한국정보과학회, 제 28권, 제 4호, pp. 550-558, 2001. 12.
[9] C.-S. Yang and M.-Y. Luo, "Efficient Support for Content-based Routing in Web Server Clusters," Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Oct. 1999.
[10] T. Schroeder, S. Goddard and B. Ramamurthy, "Scalable Web Server Clustering Technologies," IEEE Network, pp. 38-45, May 2000.
[11] V.S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel and E. Nahum, "Locality-aware Request Distribution in Cluster-based Network Servers," Proceedings of the 8th ACM Conference on Architecture Support for Programming Languages, Oct. 1998.
[12] A. Cohen, S. Rangarajan and H. Slye, "On the Performance of TCP Splicing for URL-aware Redirection," Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Oct. 1999.
[13] M. Aron, et al., "Scalable Content-based Network Servers," Proceedings of the 2000 Anual USENIX Technical Conference, June 2000.
[14] O.P. Damani, et al., "ONE-IP: Techniques for Hosting a Service on a Cluster of Machines," Proceedings of the 6th International World Wide Web Conference, Apr. 1997.
[15] G.D.H. Hunt et al., "Network Dispatcher: A connection Router for Scalable Internet Services," Proceedings of the 7th International World Wide Web Conference, Apr. 1998.
[16] E. Anderson et al., "The Magicrouter: an Application of Fast Packet Interposing," The 2nd Symposium Operational System Design and Implementation, May 1996.
[17] Cisco Systems Inc. LocalDirector, http://www.cisco.com
[18] E. Levy-Abegnoli et al., "Design and Performance of a Web Server Accelelator," IEEE INFOCOM, 1999.
[19] V. Cardellini et al., "The State of the Art in Locally Distributed Web-server Systems," ACM Computing Surveys, Vol. 34, No. 2, June 2002.
[20] M. Andreolini, M. Colajanni and M. Nuccio, "Scalability of content-aware Server Switches for Cluster-based Web Information Systems," Proceedings of the 12th International World Wide Web Conference, May 2003.
[21] A. Carzaniga and A.L. Wolf, "Content-based Networking: A New Communication Infrastructure," Lecture Notes in Computer Science, Vol. 2538, pp. 59-68, 2002.
[22] X. Zhang, M. Barrientos, J.B. Chen and M. Seltzer, "HACC: An Architecture for Cluster-based Web Servers," Proceedings of the 3rd USENIX Windows NT Symposium, July 1999.
[23] E. Casalicchio and M. Colajanni, "A Client-aware Dispatching Algorithm for Web Clusters Providing Multiple Services," Proceedings of the 10th International World Wide Web Conference, May 2001.
[24] S. Venkataraman, M. Livny and J. Naughton, "Memory Management for Scalable Web Data Servers," 13th International Conference on Data Engineering, Apr. 1997.

정 지 영

1998년 아주대학교 정보 및 컴퓨터공학부 졸업(공학사). 2000년 아주대학교 정보통신전문대학원 정보통신공학과 졸업(공학석사). 2002년 아주대학교 정보통신전문대학원 정보통신공학과 박사과정 수료. 2002년~현재 명지전문대학 컴퓨터정보과 조교수. 관심분야는 클러스터 시스템, 고가용성 시스템, 이동컴퓨팅, 시스템 성능분석

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. 1996년~현재 아주대학교 정보통신전문대학원 부교수. 관심분야는 클러스터 시스템, 그리드 컴퓨팅, 유비쿼터스 컴퓨팅, 고가용성 시스템