캐쉬 적중률 향상을 위한 웹 서버 클러스터의 부하 분배

김희규, 정지영, 장인재, 김성수
아주대학교 정보통신전문대학원
e-mail: {ikki, abback, jangij, sskim}@madang.ajou.ac.kr

Load Balancing of Web Server Clusters for Improving Cache Hit Rate

Hee-Gyu Kim, Ji Yung Chung, In Jae Jang, Sungsoo Kim
Graduate School of Information and Communication, Ajou University

요약

클러스터링을 이용한 웹 서버 구축 시 각 노드에 적절한 부하 분산이 이루어지도록 하는 것은 매우 중요하다. 부하 분배 알고리즘은 서버의 성능에 많은 영향을 미치며 부하 분배에 이용되는 기준은 간단하고 빨리 계산될 수 있어야 한다. 본 논문에서는 고가용도 및 확장성을 제공하는 클러스터링 웹 서비스를 대상으로 문서 접근 확률과 문서 크기 정보를 이용하는 동시에 캐쉬 적중률을 향상시키는 알고리즘을 제안한다.

1. 서론

급속한 인터넷의 성장과 함께 삶에서 인터넷이 차지하는 비중이 증가함에 따라 인터넷의 과부하가 발생하고 있다. 서버의 처리량이 빠르게 증가함에도 불구하고 유명한 웹서버의 경우 자주 과부하가 걸리기도 한다. 이러한 과부하로 인한 웹 서비스의 중단은 서비스 제공자와 소비자에게 큰 손실을 가져올 수 있기 때문에 웹 서버는 항상 서비스가 가능하도록 설계되어야 한다.

결함 허용 컴퓨터들은 하드웨어를 중복시키거나 자기 검사를 수행함으로써 결함을 방지하기 때문에 시스템 구축 시 많은 비용을 필요로 한다. 이에 비해 클러스터링 기법은 비용 측면에서 매우 유리하며 특히 웹 서비스나 정보 시스템과 같이 고성능과 고가용도를 요구하는 응용 분야에서 결함 허용 컴퓨터의 대안으로 등장하고 있다[1].

This work is supported in part by the Ministry of Information & Communication of Korea.("Support Project of University Foundation Research<2001>" supervised by ITA)
This work is supported in part by the Ministry of Education and Human Resources Development of Korea.(Brain Korea 21 Project supervised by Korea Research Foundation)

이러한 클러스터링을 이용한 웹 서버 구축 시 중요하게 고려해야할 사항은 각 노드에 적절한 부하 분산이 이루어지도록 하는 것이며 노드에 결함이 발생할 경우를 대비하여 적절한 서비스 시스템 전이가 이루어지도록 해야 한다.

본 논문에서는 고가용도 및 확장성을 제공하는 클러스터링 웹 서비스를 대상으로 문서 접근 확률과 문서 크기 정보를 이용하는 동시에 캐쉬 적중률을 향상시키는 알고리즘을 제안한다.

논문의 구성으로 2장에서는 관련연구를 알아보고 3장에서는 문서 접근 확률과 문서 크기 정보를 고려한 부하 분배 알고리즘을 알아보고 이를 개선한 알고리즘을 소개한다. 4장에서는 시뮬레이션 결과를 설명하고 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

2.1 기존 부하 분배 연구

클러스터링 웹 서버의 부하 분배기는 적절한 스케줄링 알고리즘을 통해 부하를 분산시키는 역할을 하며 서버 클러스터가 하나의 가상서버로 보이게 하는

역할을 한다. 부하 분배알고리즘은 서버의 성능에 많은 영향을 미치며 부하 분배에 이용되는 기준은 간단하고 빨리 계산될 수 있어야 한다[2].

부하 분배 스케줄링에 이용되는 방법은 라운드 로빈 스케줄링 방식이나 최소 연결 방식 등이 있다.

SWEB[3]은 워크스테이션들의 네트워크와 분산 메모리를 사용하는 웹 서버로서 모든 서버가 전체문서를 포함하지 않으며 LAN을 통해 다른 서버 노드의 문서를 가져온다. 또한 각 서버 노드의 평균 휴지 시간(idle time)을 이용하여 부하를 분배시키는 방식이 존재하며 이 외에도 CPU 부하나 큐 길이 등을 이용한 부하 분산 기법이 연구되었다.

RobustWeb[4]은 미리 정의된 문서 접근 확률에 따라 부하 분배기가 사용자의 요구를 분배하나 문서들의 크기가 대체로 일정하지 않을 경우 부하 불균형이 발생한다.

2.2 웹 서버 클러스터 구조

본 연구에서 고려하는 웹 서버 구조는 요구 분배를 위한 부하 분배기와 요구된 문서를 서비스하는 다수의 서버 노드로 구성되어 있다. 사용자의 요구는 부하 분배기에 전달되며 부하 분배기는 제안되는 분배 알고리즘에 의해 주어진 확률에 따라 요구된 패킷을 문서 서버에 전송하는 역할을 한다.

이 구조에서 각 문서 서버는 주기적으로 자신의 존재를 부하 분배기에 알리며 문서 서버가 다운되면 부하 분배기는 이를 고려하여 알고리즘을 재 수행함으로써 나머지 노드들에 부하를 분산시킨다.

특히, 부하 분배기는 사용자 요구를 처리하는 부분, 주어진 정보를 이용하여 각 문서에 따라 요구될 서버를 확률과 함께 결정하는 부분, 그리고 결정된 계획에 따라 실제 분배하는 부분 등으로 구성된다[5]. Request Manager는 문서에 대한 사용자 요구수를 관리하고 Redirection Table을 참조하여 문서 서버를 결정한다. Load Scheduler는 주기적으로 알고리즘을 수행함으로써 분배 확률을 갱신하며 각 문서 서버의 heartbeat 메시지를 받는다.

3. 부하 분배

3.1 문서 접근 확률과 문서 크기의 이용

[5]에서 우리는 웹 서버 클러스터 상에서 문서 접근 확률과 문서 크기를 이용한 부하 분배 알고리즘을 제안하였다.

이 알고리즘은 웹 서버에 존재하는 M개의 문서 각각이 사용자로부터 요구될 확률 $R_1, R_2, \cdots, R_M$ ($R_1 + R_2 + \cdots + R_M = 1$)과 함께 주어지고 각각의 문서 크기는 $D_1, D_2, \ldots, D_M$ 과 같이 주어진다고 가정한다. 이 때 사용자로부터 문서가 요청되면 부하 분배기는 주어진 확률에 따라 N개의 문서 서버 $S_1, S_2, \cdots, S_N$ 중에서 서비스 받을 서버를 결정하고 패킷을 전달하게 된다. 또한 $C_1, C_2, \ldots, C_N$ 은 시스템 전체에서 각 문서 노드가 처리할 수 있는 용량의 비율을 의미하며 $C_1 + C_2 + \cdots + C_N = 1$ 이 된다.

알고리즘이 동작하는 과정을 예를 들어 설명하면 다음과 같다. 6개의 문서 각각이 0.13, 0.2, 0.05, 0.33, 0.1, 0.19의 접근 확률을 가지고 있으며 각각의 문서 크기가 20KB, 30KB, 25KB, 4KB, 10KB, 6KB라고 하자. 또한 각 문서 노드는 순서대로 0.4, 0.3, 0.3의 처리 용량인 C 값을 갖는다고 가정하자. 이 때 문서 크기 정보를 고려한 Reweight_R_i는 0.2, 0.45, 0.1, 0.1, 0.07, 0.08로 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. C값이 같을 때는 임의의 서버가 선택된다고 가정한다.

문서 D_2의 Reweight_R_2가 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.

D_2에 대한 사용자의 요구는 0.89의 확률로 S_1, 0.11의 확률로 S_2에 할당되며 D_1과 D_6에 대한 사용자의 요구는 1의 확률로 S_3에 할당된다. 또한 D_3, D_4는 모두 S_2에 할당되며 D_5에 대한 사용자의 요구는 S_2에 0.71, S_3에 0.29의 확률로 할당된다. 그림 1은 이러한 부하 분배 과정을 보여주고 있다. 여기서 부하 분배기로부터 나가는 선 위의 숫자는 각 문서의 접근 확률을 나타낸다.

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.

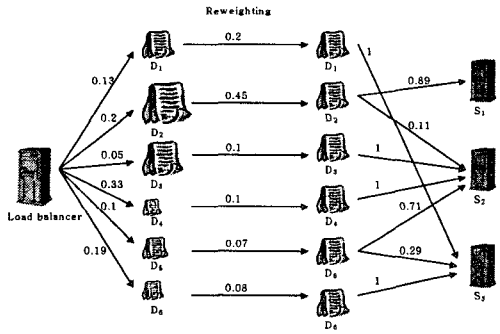

그림 1 부하분배 알고리즘 수행과정

각 서버가 서비스하는 문서의 크기를 V_1, V_2, V_3이라 하면, 위의 알고리즘은 표 1과 같이 S_1은 30KB, S_2는 64KB, S_3은 36KB의 문서를 할당받게 된다. 만약 캐쉬의 크기가 50KB라고 가정하면 S_1과 S_3은 모두 캐쉬에서 서비스를 할 수 있으나 S_2의 경우에는 디스크에서 읽어오게 된다. 위의 알고리즘은 접근 확률과 문서 크기를 고려하였지만 각 서버가 서비스하게 되는 문서의 크기를 고려하지 않았다. 따라서 어떤 서버가 다른 서버에 비해 서비스하는 문서의 크기의 합이 크게 차이나는 경우 그 서버의 캐쉬 적중률이 떨어지는 단점이 있다. 3.2절에서는 이것을 보완한 새로운 알고리즘을 소개하고자 한다.

표 1 각 서버가 서비스하는 문서의 크기

	D_1	D_2	D_3	D_4	D_5	D_6	합계
V_1	0	30	0	0	0	0	30
V_2	0	30	25	4	10	0	69
V_3	20	0	0	0	10	6	36

3.2 개선된 부하분배 알고리즘

기존의 알고리즘은 Reweight_R_i가 가장 큰 문서부터 선택해서 처리 용량이 가장 큰 서버에게 할당하였다. 반면에 개선된 알고리즘에서는 각 서버가 서비스하는 문서의 크기 V를 고려하였다. 제일 크기가 큰 순서로 문서를 선택해서 서버 중 가장 작은 크기의 문서를 서비스하는 서버에 할당하게 된다. 단, Reweight_R_i 값이 처리 용량 C 값 보다 작은 서버가 있으면 그 서버 중 가장 작은 크기의 문서를 서비스하는 서버를 할당하게 된다. 만약 서버에서 서비스하는 문서의 크기가 같다면 C 값이 큰 서버를 선택한다고 가정한다.

```
New Redirection Algorithm
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 Di
      While (if Reweight_Ri is not 0)
         do
            if(there exist servers with
               Reweight_Ri <= C)
               select server j with the
               minimum Vj of them
               Cj ← Cj - Reweight_Ri
               Redirection_table[i][j] ← Reweight_Ri
               Reweight_Ri ← 0
               Vj ← Vj + Di
            else
               select server j with the
               minimum Vj of all servers
               Reweight_Ri ← Reweight_Ri - Cj
               Redirection_table[i][j] ← Cj
               Cj = 0
               Vj ← Vj + Di
For i ← 1 to M
   For j ← 1 to N
      do Redirection_table[i][j]
         ←Redirection_table[i][j]/Temp_Ri
```

알고리즘이 동작하는 과정은 다음과 같다. 앞의 예와 같이 접근 확률과 문서 크기, 서버의 처리 용량이 같다고 가정할 때 문서 크기 정보를 고려한 Reweight_R_i는 0.2, 0.45, 0.1, 0.1, 0.07, 0.08로 결정되고 문서 크기가 가장 큰 D_2의 Reweight_R_2가 선택된다. 따라서 D_2는 이때 0.45보다 큰 C 값을 가진 서버가 없으므로 서비스하는 문서 크기가 가장 작은 서버를 선택해야 한다. 그러나 현재 모두 서비스하는 문서 크기 V 값이 0KB이므로 이때는 가장 큰 C 값을 가진 S_1을 선택한다. 0.45중 0.4만 S_1에 할당되고 나머지 0.05는 S_2에 할당된다.

문서 D_2의 Reweight_R_2가 모두 할당되면 그 다음으로 높은 크기를 갖는 D_3이 선택된다. S_2와 S_3 둘 다 C 값이 D_3의 Reweight_R_3 값보다 크다. 이때 V_2가 30KB, V_3이 0KB이므로 서비스하는 문서의 크기가 작은 S_3에 할당된다.

문서 D_3의 Reweight_R_3이 모두 할당되면 다음으로 높은 크기를 갖는 D_1이 선택된다. S_2와 S_3 둘 다 C 값이 D_1의 Reweight_R_1 값보다 크거나 같다. 이때 서비스하는 문서의 크기는 V_2가 30KB, V_3이 25KB이므로 서비스하는 문서의 크기가 작은 S_3에 할당된다.

이와 같이 모든 문서와 서버에 대해 수행되어 알고리즘이 끝나면 분배 테이블은 표 2와 같은 결과를 갖게 된다.

표 2 알고리즘 수행결과

	D_1	D_2	D_3	D_4	D_5	D_6
S_1	0	0.89	0	0	0	0
S_2	0	0.11	0	1	1	1
S_3	1	0	1	0	0	0

각 서버에서 서비스하는 문서의 크기는 표 3과 같은 결과를 갖게 되며 V_1의 값이 30KB, V_2가 50KB, V_3이 45KB 값을 가지게 된다. 이를 표 1과 비교해 보면 각 서버에 고르게 분포된 것을 알 수 있다.

표 3 각 서버가 서비스하는 문서의 크기

	D_1	D_2	D_3	D_4	D_5	D_6	합계
V_1	0	30	0	0	0	0	30
V_2	0	30	0	4	10	6	50
V_3	20	0	25	0	0	0	45

각 서버에서 서비스하는 문서의 크기가 고르게 분배되면 각 서버의 캐쉬 적중률이 비슷하게 되어 서버들이 고르게 성능을 낼 수 있다. 개선된 알고리즘은 기존 알고리즘의 장점을 살리면서 문서의 크기가 고르게 분배되지 않는 단점을 보완할 수 있다.

4. 성능 평가

이 절에서는 제안된 분배 알고리즘의 효율성을 알아보기 위해 시뮬레이션 실험을 수행하였으며 그 결과를 보여준다.

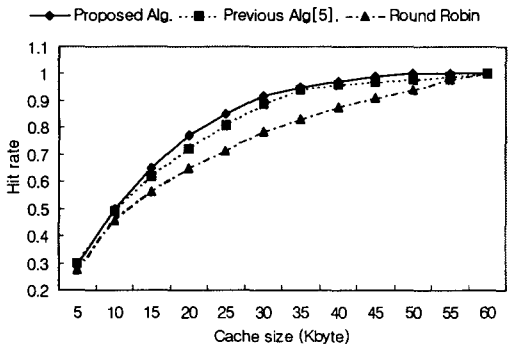

그림 2 캐쉬 적중률 비교

시뮬레이션에 사용된 파라미터 값으로 각 문서의 크기와 문서 접근 확률, 서버의 수와 처리 용량 등은 3장의 예와 동일하다. 그림 2는 제안된 알고리즘의 캐쉬 적중률이 라운드 로빈 방식이나 [5]에서 제안된 알고리즘보다 우수함을 보여주고 있다. 시뮬레이션에서 사용된 캐쉬 정책은 LRU(Least Recently Used) 방식이며 교체되는 블록의 크기는 1KB 단위로 하여 실험하였다. 또한 각 서버의 캐쉬 크기는 서로 같다고 가정하였다.

5. 결론

클러스터링을 이용한 웹 서버 구축 시 중요하게 고려해야할 사항은 각 노드에 적절한 부하 분산이 이루어지도록 하는 것이다.

본 논문은 평균 문서 접근 확률과 문서 크기 정보를 동시에 고려하여 부하 분산을 수행하는 알고리즘의 캐쉬 적중률을 높이기 위해 각 서버마다 비슷한 크기의 문서를 서비스하도록 알고리즘을 개선하였다. 이와 같이 각 서버의 캐쉬 적중률을 유사하게 하면 보다 향상된 부하 분배를 이룰 수 있다.

참고문헌

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

[2] R. Buyya, "High Performance Cluster Computing: Architectures and Systems," Prentice-Hall, p. 849, 1999.

[3] D. Andresen, et. al., "SWEB: Towards a Scalable WWW Server on Multicomputers," Proc. of International Symposium on Parallel Processing, IEEE, pp. 850-856, Apr. 1996.

[4] B. Narendran, S. Rangarajan and S. Yajnik, "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access," Proc. of the 16th Symposium on Reliable Distributed Systems(SRDS), Oct. 1997.

[5] 김성수, 정지영, "웹 서버 클러스터를 위한 효율적인 부하분배 알고리즘," 한국정보과학회논문지(정보통신), 한국정보과학회, 제28권, 제4호, pp. 550-558, 2001. 12.