웹 클러스터 시스템에서 개선된 WLC 스케줄링 알고리즘

최동준°, 정광식, 손진곤
한국방송통신대학교 평생대학원 정보과학과
hotfeel7@daum.net°, {kchung0825, jgshon}@knou.ac.kr

An Improvement on the Weighted Least-Connection Scheduling Algorithm in Web Cluster Systems

Dong Jun Choi°, Kwang Sik Chung, Jin Gon Shon
Dept. of Computer Science, Graduate School, Korea National Open University

요 약

웹 클러스터 시스템은 여러 대의 서버로 부하를 분산시키는 시스템이다. 부하 분산 알고리즘으로 가중치를 부여하고 연결 수가 가장 적은 리얼 서버를 선출하는 방식인 가중치 기반 최소-연결(WLC: Weighted Least-Connection) 스케줄링 알고리즘을 가장 많이 사용하고 있다. 이 알고리즘은 동시 사용자 수가 많은 웹 클러스터 시스템에 새로운 리얼 서버가 추가될 경우 새로운 서버에게 웹 요청을 집중적으로 할당하여 해당 서버에 과부하를 발생시키는 문제점이 있다. 본 논문에서는 새로운 리얼 서버에게 발생하는 과부하를 줄이기 위해, 웹 요청이 새로운 리얼 서버에게 연속적으로 할당되면 해당 서버를 스케줄링에서 잠시 제외되도록 하는 개선된 WLC 스케줄링 알고리즘을 제안한다.

1. 서론

컴퓨터의 처리 능력을 향상시키기 위한 방법으로는 단일 서버의 업그레이드를 통한 하드웨어적인 확장과 클러스터링 기술을 이용한 소프트웨어적인 확장이 있다. 소프트웨어적인 확장은 시스템의 가용성을 보장할 수 있고 하드웨어적인 확장보다는 비교적 적은 비용으로 확장이 가능하기 때문에 이에 대한 연구가 활발히 진행되고 있다[1,4,5].

클러스터 시스템이란 여러 대의 서버를 고속 네트워크로 연결하여 단일 서버처럼 동작하도록 하는 시스템이다. 클러스터 시스템은 그룹에 서버들을 계속 추가할 수 있기 때문에 슈퍼컴퓨터나 병렬컴퓨터보다 확장성 측면에서 우수하다. 또한 고속 컴퓨팅 및 멀티미디어 데이터를 효과적으로 처리할 수 있다. 클러스터 시스템은 사용 목적에 따라 과학 계산용과 부하 분산용으로 구분할 수 있다. 웹 클러스터 시스템은 부하 분산용 클러스터 시스템의 한 종류로써 단일 서버를 사용할 때 발생할 수 있는 부하를 여러 서버로 분산시키는 시스템이다[2,4].

웹 클러스터 시스템은 부하를 분산시키는 로드 밸런서(load balancer)와 실제로 웹 요청을 처리하는 리얼 서버(real server)로 구성되고, 부하 분산(load balancing) 스케줄링 알고리즘으로 WLC 스케줄링을 가장 많이 사용하고 있다[3]. WLC 스케줄링은 동시 사용자 수가 많은 웹 클러스터 시스템에 새로운 리얼 서버가 추가될 경우, 웹 요청이 새로운 서버에게 집중되어 리얼 서버들 사이에 부하 불균형이 발생한다. 본 논문은 새로운 리얼 서버에게 집중되는 부하를 줄이기 위해, 개선된 WLC 스케줄링 알고리즘을 제안한다.

2. 관련 연구

2.1 부하 분산 스케줄링 알고리즘

웹 클러스터 시스템에서 사용하는 부하 분산 스케줄링 알고리즘은 다음과 같다.

2.1.1 라운드-로빈 스케줄링

라운드-로빈(round-robin) 스케줄링은 라운드-로빈 방식을 이용하여 웹 요청을 다른 리얼 서버에게 보낸다. 이 스케줄링은 리얼 서버들에 대한 연결 수나 응답 시간을 고려하지 않고 모두 동일한 것으로 취급한다[6].

2.1.2 가중치 기반 라운드-로빈 스케줄링

가중치 기반 라운드-로빈(weighted round-robin) 스케줄링은 다른 처리 용량의 리얼 서버들을 취급할 수 있다. 각 리얼 서버는 처리 용량을 나타내는 정수로 된 가중치가 할당될 수 있다. 이 스케줄링은 스케줄 요청에 효과적이지만, 웹 요청들의 부하가 매우 다양하면 리얼 서버들 사이에 부하 불균형을 초래한다[6,7].

2.1.3 최소-연결 스케줄링

최소-연결(least-connection) 스케줄링은 웹 요청을 연결 수가 가장 적은 리얼 서버에게 보낸다. 최소-연결 스케줄링은 비슷한 성능의 리얼 서버들로 이루어진 웹 클러스터 시스템에서는 부하가 많은 웹 요청들이 한 리얼 서버로 보내지는 일이 없기 때문에 웹 요청들의 부하가 많을 때에도 부하를 잘 분산시킨다. 그러나 다양한 처리 용량의 리얼 서버들 사이에서는 여전히 부하 불균형이 발생한다[6].

2.1.4 가중치 기반 최소-연결 스케줄링

가중치 기반 최소-연결 스케줄링은 각 리얼 서버마다 성능 가중치가 할당될 수 있다. 보다 높은 가중치의 리얼 서버는 언제나 더 많은 비율의 연결을 받는다. WLC 스케줄링은 n개의 리얼 서버가 있고, 각 리얼 서버 i는 가중치 W_i (i=1,⋯,n)와 연결 수 C_i (i=1,⋯,n)를 가진다고 가정하면, 웹 요청은 다음 식을 만족하는 리얼 서버 j에게 보내지게 된다[6].

$$C_j/W_j = \min\{C_i/W_i\}\ (i=1,\cdots,n)$$

WLC 스케줄링은 동시 사용자 수가 많은 웹 클러스터 시스템에 새로운 리얼 서버가 추가될 경우 웹 요청을 새로운 서버에게 집중적으로 할당함으로써 해당 서버에 과부하를 발생시키는 문제점이 있다.

3. 개선된 WLC 스케줄링 알고리즘

본 논문에서 제안된 기법은 웹 요청을 새로 추가된 리얼 서버에게 연속적으로 할당함으로써 발생하는 과부하를 줄이기 위해, 웹 요청이 새로운 리얼 서버에게 연속적으로 할당되면 해당 서버를 스케줄링에서 잠시 제외시키는 방식이다. 제안된 기법은 리얼 서버 선출 알고리즘과 과부하 감소 알고리즘으로 구성된다.

3.1 리얼 서버 선출 알고리즘

그림 1은 웹 요청이 할당될 리얼 서버를 선출하는 알고리즘이다. 이 알고리즘은 기존의 WLC 스케줄링 알고리즘과 동일한 방식으로 리얼 서버를 선출한다. 그리고 선출된 리얼 서버와 활성 상태인 리얼 서버의 수를 매개변수로 하여 Reduce_Overhead 함수(과부하 감소 알고리즘)를 호출한다.

```
WLCI_Schedule(A[], n)
  입력 : A[0:n-1] : 리얼 서버의 배열,
         n : A배열 원소의 수.
  출력 : least : 선출된 리얼 서버.
{
  var least :  선출된 리얼 서버.
  var active: 활성 상태인 리얼 서버의 수.

  for (i = 0; i < n; i++)
  {
    if (A[i].flag == DISABLE)
      continue;

    if ( least.conns / least.weight >
       A[i].conns / A[i].weight )
    {
      least = A[i];
    }

    active++;
  }

  Reduce_Overhead(least, active);
  return least;
}
```

그림 1. 리얼 서버 선출 알고리즘

3.2 과부하 감소 알고리즘

그림 2는 리얼 서버의 과부하를 감소시키는 알고리즘이다. 이 알고리즘은 동일한 리얼 서버가 연속적으로 할당되면 해당 서버를 비활성화 하여 스케줄링에서 잠시 제외시키는 방식이다.

Reduce_Overhead 함수는 선출된 리얼 서버와 활성 상태인 리얼 서버의 수를 인자로 받는다. 이때 만약 활성 상태인 리얼 서버의 수가 2대 미만이면 함수를 종료한다. 선출된 리얼 서버가 이전에 할당된 리얼 서버와 동일하면 카운트를 1 증가시키고, 그렇지 않으면 선출된 리얼 서버를 기억하고 카운트를 1로 초기화 한다. 그리고 만약 카운트 값이 최대 연속 할당 수(L)보다 크거나 같으면 선출된 리얼 서버를 스케줄링에서 제외하고, 카운트 값을 L-1로 설정한다. 이후 요청부터는 카운트를 1씩 감소시키고, 만약 이 값이 1보다 작을 경우 해당 리얼 서버를 다시 스케줄링에 포함시킨다.

```
Reduce_Overhead(s, n)
  입력 : s : 선출된 리얼 서버.
         n : 활성 상태인 리얼 서버의 수.
{
  var old_s : 이전에 할당된 리얼 서버.
  var count : 리얼 서버가 연속 할당된 수.
  var flag :  과부하 모드 플래그.
  var MAX_CONT_ALLOC : 최대 연속 할당 수.

  if (flag == 1) {
    if ( --count < 1) {
      old_s.flag = ENABLE;
      flag = 0;
    }
    return;
  }

  if (n < 2) return;

  if old_s != s) {
    old_s = s;
    count = 1;
    return;
  }

  if (++count >= MAX_CONT_ALLOC) {
    count = MAX_CONT_ALLOC - 1;
    old_s.flag = DISABLE;
    flag = 1;
  }
}
```

그림 2. 과부하 감소 알고리즘

4. 성능평가

제안된 기법을 기존의 WLC 스케줄링 기법과 비교 평가를 하기 위해 C 언어를 이용하여 시뮬레이션을 하였다.

시뮬레이션은 다음 조건으로 가정한다. 웹 요청 형태는 초당 평균 250번의 일양분포로 한다. 또한 연결된 상태에서 재요청하는 형태는 평균 3번, 재요청 시까지의 대기 형태는 평균 10초의 포아송분포로 한다. 그리고 리

얼 서버가 작업을 처리하는 형태는 평균 2초, 표준 편차가 1초인 정규분포로 한다. 20초 동안 재요청이 없으면 연결을 끊고, 최대 연속 할당 수는 10으로 한다.

이와 같은 조건으로 리얼 서버가 9대인 가상 웹 클러스터 시스템에 대해 t=51(sec)일 때까지 시뮬레이션을 수행한다. 각 리얼 서버 i는 가중치 W_i (i=1,…,n), 작업 수 J_i (i=1,…,n), 연결 수 C_i (i=1,…,n)를 가진다고 가정한다. 표 1은 t=51일 때의 가상 웹 클러스터 시스템의 상태이다. 이 상태에서 가중치가 1인 R10 서버를 추가로 투입하여 기존 기법과 제안된 기법의 성능을 비교하였다.

표 1. R10 서버를 투입하기 전 가상 웹 클러스터 시스템의 상태

리얼 서버(i)	가중치(W_i)	작업 수(J_i)	연결 수(C_i)
R1	1	126	761
R2	2	227	1522
R3	1	120	760
R4	3	329	2282
R5	1	120	761
R6	2	233	1521
R7	1	115	761
R8	1	120	761
R9	3	324	2282

그림 3은 기존 기법과 제안된 기법에서 R10 서버의 연결 수를 비교한 것이다. 기존 기법에서는 R10 서버의 연결 수가 다른 리얼 서버들의 연결 수와 균등해질 때까지 웹 요청을 R10 서버로 집중적으로 할당하기 때문에 연결 수가 급격하게 증가한다. 그러나 제안된 기법에서는 웹 요청이 R10 서버로 연속적으로 10번 할당되면 9번은 다른 리얼 서버에게 할당되기 때문에 연결 수가 기존 기법보다 완만하게 증가한다.

그림 4는 기존 기법과 제안된 기법에서 R10 서버의 작업 수를 비교한 것이다. 기존 기법에서는 R10 서버에 초당 약 250개의 웹 요청이 할당되기 때문에 작업 수가 순간적으로 증가한다. 그러나 제안된 기법에서는 R10 서버에 초당 약 140개의 웹 요청이 할당되기 때문에 작업 수가 순간적으로 증가하지 않는다. 따라서 제안된 기법이 기존 기법보다 새로 추가된 리얼 서버에게 부하를 적게 준다는 것을 확인할 수 있다.

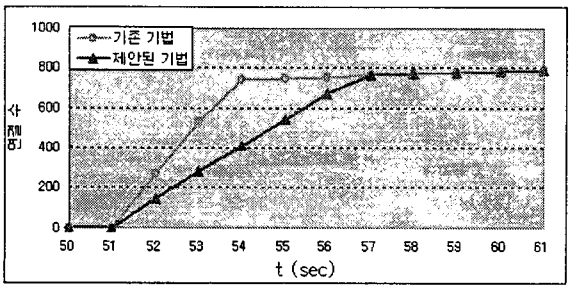

그림 3. R10 서버의 연결 수 비교

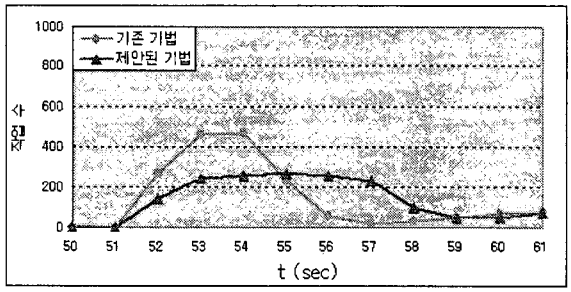

그림 4. R10 서버의 작업 수 비교

5. 결론 및 향후 연구과제

본 논문에서는 새로 추가된 리얼 서버에게 부하를 적게 주는 개선된 WLC 스케줄링 알고리즘을 제안하였다. 제안된 기법은 웹 요청이 새로운 리얼 서버에게 최대 연속 할당 수(L) 이상으로 연속 할당되면, 해당 서버를 스케줄링에서 L-1회 동안 제외시키는 방식이다. 시뮬레이션을 수행한 결과 기존 기법에서는 새로 추가된 R10 서버의 작업 수가 최대 508개인 반면, 제안된 기법에서는 새로 추가된 R10 서버의 작업 수가 대략 260개를 유지하였다. 결과적으로 제안된 기법은 기존 기법보다 새로 추가된 리얼 서버에게 부하를 적게 줌으로써 부하 분산 효율을 향상시킨다.

향후에는 리얼 서버가 동시에 2대 이상 추가될 경우에도 과부하를 줄일 수 있는 기법과, 웹 클러스터 시스템의 상태에 따라 최대 연속 할당 수를 자동으로 조절하는 기법에 대해 연구할 계획이다.

참고문헌

[1] 김석찬, 이영, "부하분산 알고리즘을 적용하여 반응시간을 감소시키는 웹 클러스터 시스템 구축", 한국정보과학회 논문지C, 제10권, 제6호, pp.507-513, 2004.

[2] 권오영, "클러스터 시스템 개요", 기술·정책 동향, 한국과학기술정보연구원 소식지, 2000.

[3] 윤천균, "웹 클러스터 시스템의 실시간 서버 상태를 기반으로 한 부하분산 방안", 한국정보처리학회 논문지A, 제12권, 제5호, pp.427-432, 2005.

[4] Nortel Networks, Alteon Web Switch, http://www.nortel.com/, (2006.4.7 방문).

[5] W. Zhang and et al, Linux Virtual Server Project, http://www.linuxvirtualserver.org/, (2006.4.7 방문).

[6] W. Zhang, "Linux Virtual Server for Scalable Network Services", Ottawa Linux Symposium, 2000. http://www.linuxvirtualserver.org/ols/lvs.pdf.

[7] G. Hunt and G. Goldszmidt, "Network Dispatcher: a connection router for scalable Internet services", Proc. 7th Int'l World Wide Web Conf., 1998. http://www.unizh.ch/home/mazzo/reports/www7conf/fullpapers/1899/com1899.htm