LVS 클러스터 시스템의 요구 패턴에 따른 부하 분산 알고리즘 분석

이선홍*, 김성기, 나용희, 민병준
인천대학교 컴퓨터공학과
e-mail:{lishanhong, proteras, ramen, bjmin}@incheon.ac.kr

Analysis of the Load Balancing Algorithms according to the Request Patterns on the LVS Cluster Systems

Shan Hong Li*, Sung Ki Kim, Yong Hee Na, Byoung Joon Min
Dept. of Computer Science, Incheon University

요약

잘수록 증가하는 인터넷 사용자의 서비스 요구량에 대처하기 위해, 부하 분산 기능을 갖춘 클러스터 시스템의 이용이 늘어가고 있다. 본 연구에서는 클라이언트에게 보다 향상된 응답 성능을 제공하기 위해 사용하는 RR(Round Robin), WRR(Weighted Round Robin), LC(Least Connection), WLC(Weighted Least Connection) 방법의 알로리즘의 구현 및 결과를 논한다. 이를 위해, 실제 클러스터의 측정 결과를 토대로 단위 시간 당 응답되는 클라이언트의 요구량 변화를 7 가지 패턴으로 분류하였고, 리눅스 가상 서버(LVS: Linux Virtual Server) 클러스터 시스템의 대상으로 7 가지 요구 패턴에 따른 부하 분산 특성을 알아보았다. 본 연구를 통해서 클라이언트 요구량 변화 패턴에 따른 최적의 부하 분산 알고리즘을 제시할 수 있었다. 본 연구 결과는 향후 효율적인 동적 부하 분산 연구에 좋은 참고가 될 것이다.

1. 서론

인터넷이 대중화되면서 사용자의 인터넷 사용시간과 서비스 요구량이 증가함에 따라 서비스를 제공하는 서버의 부하 문제를 해결하기로 노력이 이루어지고 있다. 이러한 노력의 일환으로 동일한 컴퓨터를 많이 사용하는 서버를 하나의 클러스터로 묶어, 부하를 분산시키는 서비스는 클라스터 시스템의 사용이 널리 보급되고 있다. 이러한 클러스터 시스템의 일반적인 구조는 클라이언트로부터 부하를 수신하는 부하 분산(load balancing) 노드와 실제 서비스를 수행하는 real 서버로 구성된다. 모든 노드는 클라이언트에게 대표하는 가장 IP 주소를 공유하고 각자 실제 IP 주소를 가지고 서로를 인식한다. 이때 클라이언트로부터 인급되는 요구를 부하 분산 노드에서 작업 서버(real server)에게 분산하는 대표적인 방법으로는 RR(Round Robin), WRR(Weighted Round Robin), LC(Least Connection), WLC(Weighted Least Connection) 알고리즘이 있다.[1,5]

이러한 부하 분산 알고리즘의 목적은 클라이언트 요구량 작업 서버에게 고르게 분산함으로써 클라이언트에 대한 서버의 응답 성능을 높이고 QoS를 증가하는 것이다. 그러나 이들 4 가지 부하 분산 알고리즘은 단순한 작업 서버의 평균 응답 시간과 연결 수에 기초하여 클라이언트 요구량을 분산하고 있다.[2] 이를 보완하여, 작업 서버의 프로세스, 메모리 등의 컴퓨팅 자원에 부과된 부하를 반영함으로써, 동적으로 부하 분산하는 방법도 있지만, 모두 시스템에 인급되는 클라이언트 요구량 변화에 따른 부하 분산 특성을 간략화하고 있다. 실제 웹 서버의 경우, 수신하는 요구량(request/sec)이 많이날수록 웹 서버의 처리용(connection/sec)이 저하되며, 많은 시간에 요구량이 폭주하는(bursty condition) 경우에는 그러한 성능 저하가 크게 나타난다[4]. 따라서, 특정 요구 패턴에서는 현재의 서버가 받는 부하나, 연결 수 등을 부하 분산에 반영하는 자체가 오шиб되므로 주의할 필요가 있다.

본 연구에서는 일정 시간 간격을 요하는 서비스 수행 코드를 작업 서버에 준비했고, 단위 시간당 인급되는 클라이언트 요구량 변화에 따라 4 가지 알고리즘에 대한 응답 특성을 조사하여 분석한다. 이를 통해, 실험 결과에 따른 클라이언트 요구량 변화에 어떤 부하 분산 알고리즘을 사용하는 것이 안전한지, 그리고 어떤 서비스의 응답 시간을 최적화할 수 있는지, 또한, 클라이언트의 요구를 적절히 분산할 수 있는 알고리즘을 최적화할 수 있는지, 한편으로는 링크의 네트워크 사용률을 최적화할 수 있는지 등의 문제를 해결하고자 한다.
2. 관련 연구
2.1 리눅스 LVS 클러스터 시스템

(그림 1) 리눅스 LVS 클러스터 시스템 환경

본 연구에서는 (그림 1)과 같이 리눅스 기반의 LVS 클러스터 시스템 환경에서 일정 시간을 요하는 웨 어플리케이션 클러스터 시스템에서 확장 및 실태 분석을 개발하고 통계적 성능을 고려하여 분석하였다. 

본 연구에서는 (그림 1)과 같이 리눅스 기반의 LVS 클러스터 시스템 환경에서 일정 시간을 요하는 웨 어플리케이션 클러스터 시스템에서 확장 및 실태 분석을 개발하고 통계적 성능을 고려하여 분석하였다. 

본 연구에서는 (그림 1)과 같이 리눅스 기반의 LVS 클러스터 시스템 환경에서 일정 시간을 요하는 웨 어플리케이션 클러스터 시스템에서 확장 및 실태 분석을 개발하고 통계적 성능을 고려하여 분석하였다. 

3. 요구량 변화 모델 및 실험 방법
3.1 요구량 변화 모델
시스템에 시각적 데이터를 들어오는 클라이언트 요구량의 변화에 따라 4 가지 부하 분산 알고리즘의 유효성을 알아 보기 위해서는 먼저 실제 시스템을 대상으로 시간에 따라 변화 있는 클라이언트 요구 패턴이 들어오는지에 대한 조사가 필요하다. 본 연구에서는 네트워크에서 수집된 요구 패턴 분석에 대한 [2,3]의 결과를 토대로 다음과 (그림 2)와 같은 4 가지 요구 패턴을 모형화 하였다. 각 요구 패턴은 입력의 시간 동안 시스템이 수신한 단위 시간당 요구량(N(t))을 의미한다.

![요구 패턴 1](image1)

\[ N(t) = a(t) \quad (0 < t < 30) \]

![요구 패턴 2](image2)

\[ N(t) = a(t) \quad (a > 0) \]

![요구 패턴 3](image3)

\[ N(t) = at^2 \quad (a > 0) \]

![요구 패턴 4](image4)

\[ N(t) = \log_e(t) \]

![요구 패턴 5](image5)

\[ N(t) = a / t \quad (a > 0) \]

![요구 패턴 6](image6)

\[ N(t) = \sqrt{a^2 - t^2} \quad (a > 0) \]

2.2 부하 분산 알고리즘
가. RR(Round Robin) 알고리즘
부하 분산 방법이 단순히 들어온 요구수를 순시대로 변할 때 기존의 작업 시비에 대해 분산하는 방법이다. 작업 사이의 동일한 성능을 고려하기 위해선 불가능하다.
나. WRR(Weighted Round Robin) 알고리즘
물리적 성능이 우수한 작업 시비가 좀 더 많은 요구를 처리하도록 작업 시비마다 가중치를 두고 분산하는 방법이다.
다. LC(Least Connection) 알고리즘
- 시비의 현재 연결 수를 기준으로 연결이 더 적은 시비에 예약을 하겠다.
4. 분석 결과

앞장에서 소개한 요구 생성 톱을 이용하여 LVS 클러스터 시스템의 4 가지 부하 분산 알고리즘의 응답 특성을 얻었다. 다음과 (그림 3)은 7 가지 각 패턴별로 4 가지 알고리즘의 응답 특성 결과이다.

3.2 요구 생성 도구

본 연구에서는 30초 동안 매 초마다 (그림 2)에 표현된 각 패턴의 요구생성을 발생시키는 요구 생성 톱을 구현하였다. Java 언어를 이용하여 구현한 요구 생성 톱은 클러스터 시스템으로 요구 메시지를 전송하는 가상 클라이언트로 사용하였다.

3.3 리눅스 LVS 클러스터 시스템 환경

다음은 실험에 사용된 컴퓨터를 차원이다.

<table>
<thead>
<tr>
<th>구분</th>
<th>Processor</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>작업 서버 1</td>
<td>펜티엄 2 400 MHz</td>
<td>192 MB</td>
<td>100 Mbps</td>
</tr>
<tr>
<td>작업 서버 2</td>
<td>펜티엄 3 600 MHz</td>
<td>192 MB</td>
<td>100 Mbps</td>
</tr>
<tr>
<td>부하 분산 노드</td>
<td>펜티엄 4 1.7 GHz</td>
<td>256 MB</td>
<td>100 Mbps</td>
</tr>
</tbody>
</table>

3.4 응답 특성 측정 방법

알개에서 소개한 요구 생성 톱을 이용하여 요구 메시지를 전송하기 전에, 작업 서버에 일정한 서비스 수행시간이 소요되는 코드를 동작 시켜 두고, 실제 서버의 작업 처리시간을 측정 요구 메시와 데이터베이스에 저장하게 하였다. 데이터베이스는 각 작업 서버에 위치하며 데이터베이스 안에는 log라는 table이 존재한다. log table은 각 패턴별로 수집된 모든 요구에 대한 처리 시간을 저장하며, SQL query를 이용하여 평균 응답 시간을 구할 수 있다. 다음 (표 1)은 log table의 구조이다.

<table>
<thead>
<tr>
<th>log table</th>
<th>id</th>
<th>curr_time</th>
<th>exec_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>id : 30초 동안에 접속된 request의 순서</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>curr_time: 접속한 시간</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exec_time: 한 request에 대한 처리 시간</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

본 연구에서는 클라이언트 요구에 대한 응답시간을 서비스 수행 코드를 실행한 시간과 결과를 전송한 시간으로 측정하여, 클라이언트로 결과가 전송되는 시간을 동일한 실험 조건이므로 무시하였다. 따라서, 클라이언트에 대한 응답 시간을 서비스의 서비스 수행 최대시간으로 계산하였다.

(그림 3)의 각 요구 패턴 결과는 LVS 클러스터 시스템에서 30초 동안 4가지 부하 분산 알고리즘의 응답 시간을 측정한 후, 각 측정값에 대해 30초 동안의 평균 을 구한 결과이다. 각 패턴별로 응답 특성을 살펴보면, (a)의 패턴과 같은 요구량 변화에서는 서버의 물리적 사양과 현재의 연결 수를 근거로 가중치를 적용한 WRR와 WLC 알고리즘의 응답 성능이 두 작업 서버간의 차이
가 없이 좋은 응답 특성을 보였다. 이것은 가중치에 의해 두 서버의 물리적 성능과 연결 수용 과르게 분산 시켰기 때문인 것으로 판단된다. 또한, 동일한 조건에서 서버의 물리적 장애가 일어나는 작업 서버 1 과 물리적 장애가 상대적으로 좋은 작업 서버 2의 응답 성능의 차이를 비교할 때, RR 및 LC의 WRR 및 WLC의 짝을 통해 이들 두 서버의 응답 성능 차이가 큰 것은, 단위 시간당 요구량의 증가가 서버의 프로세서 성능에 비례하며, 이는 동시에 연결 수에도 영향을 주는 것으로 판단 할 수 있다.
[4]에서는 웹 서버가 일정한 요구 수신률(request/sec)에 도달하면 요구 메시지 처리를 보다 프로토콜 처리에 사용하는 프로세서 이용율이 높아지는 특성 때문에, 웹 서버의 성능 한계점(saturation point)에 도달하고, 그 결과 웹 서버의 요구 메시지 처리 능력은 서버 성능에 의존하게 된다. 따라서, 프로세서 물리적 성능을 고려하지 않고 부하를 분산한 RR 및 LC는 두 작업 서버간의 응답 성능에 차이가 상대적으로 큰 것으로 나타났다. 이러한 양상을 계획하는 그림(b), (d), (f), (g)에서도 비슷하게 나타나며, 단위 시간당 입양되는 요구량의 변화에 따라 그 기울기가 급속히 가변가된 응답 성능의 차이가 더 두드러짐을 알 수 있다.
그림(c)와 같은 패턴의 폭주 조건(bursty condition)에서는 앞에서 논의한 분산의 다르게, WRR의 다수 RR 알고리즘의 최적의 부하 분산 알고리즘을 보여준다. 이것은 폭주 조건에서 현재의 연결 수와 연결 수에 가중치를 재배치하여 부하 분산에 반응하는 그 자체가 오이렇게 도로 작용하는 것으로 판단할 수 있다. 오히려 서버의 물리적 성능에 가중치를 두고 부하 분산시킨 WRR의 가장 응답 시간의 변화가 적으면서 폭주 응답 특성을 보이고 있다. 특히 그림(c)의 실제에 있어서 일정수의 수집 요구량(request/sec)에 도달하면 대 이상의 연결이 허용되지 않았음을 확인 할 수 있었다. 이는 LVS 클러스터 시스템의 부하 분산 기능의 일부이기보다는 웹 서버의 특성인 것으로 본다. 실제로 본 연구에서, LVS 클러스터 시스템의 각 패턴에 대한 부하 분산 알고리즘을 수행하는 결과는 분산 응답률을 축소한 결과, 추가적인 병합된 부하가 발생하지 않았다.
그림(e)의 요구 패턴과 같이 초기 요구량이 많으면서 편리하게 그 요구량이 감소하는 경우에도, 단순히 들어오는 순서대로 요구 메시지를 분산시키는 RR을 제외하고는 나머지 부하 분산 알고리즘의 응답 특성이 두 작업 서버간의 응답 성능의 차이가 크게 고르게 나타났다.
그림(h)는 두 작업 서버간의 평균 응답 시간의 차이를 각 알고리즘별로 도시하고 있다. 그림(h)은 그림(c)과 같은 폭주 조건을 제외하고는 전 알고리즘에서 WLC 알고리즘 두 작업 서버간의 응답 시간 차이가 가장 적은 부하 분산 알고리즘을 보여주고 있다.
이상의 분석 결과는 효율적인 동적 부하 분산 알고리즘
설계에 있어서 2가지 요구사항을 제시하는데, 첫째는 기존의 부하분산 방법에 추가적으로 실시간으로 변화하는 수신 요구량의 변화에 대해서도 탐색적으로 적용하는 부하 분산 방법 연구가 필요하다는 것이다. 또 한편은 클라우드 네트워크에 관한 QoS 제공 측면에서, 서버의 부하를 고르게 분산한다는 것이 클라우드 네트워크 특성의 전적으로 개선하는 것이라고 볼 수 없다는 것이다. 이는 부하 분산 방법도 중요하지만 클라우드 성과에 응답하는 응용의 특성에 따라 분산 방안에 반영되어야함을 의미한다.

5. 결론
본 논문에서는 LVS 클러스터 시스템 환경에서 RR, WRR, LC, WLC 알고리즘의 응용 특성을 7가지 요구 패턴에 따라 분석해 보았다. 이를 위해, 실제 시스템에서 메트릭스를 통해 들어오는 요구량 변환을 제어 모범으로 하였고, 이 모범을 반영할 수 있는 가장 클라우드 컨텐츠로서 요구 생성 도구를 java 언어로 구현하였다. 구현된 요구 생성 도구를 이용해, 단위 시간당 입양되는 요구량 변환에 따라, 4가지 부하 분산 알고리즘의 부하 분산 특성을 적었다. 실험을 통해 얻은 결과를 분석하고, 어떤 요구량 변환 패턴에서 어느 알고리즘이 최적인지를 제시하였다.
본 연구 결과는 효율적인 동적 부하 분산 알고리즘 설계에 좋은 참고가 될 것이다.

참고문헌