Docker를 이용한 에너지 효율이 높은 로드 밸런서 설계

채민수*, 타농싹*, 양광*, 이화민**
*순천향대학교 컴퓨터학과
**순천향대학교 컴퓨터소프트웨어공학과
e-mail:{cms, leehm}@sch.ac.kr

Energy efficient web load balancer using Docker[1)]

Minsu Chae*, Xayasouk THANGONGSAK*, Yang GUANG*, HwaMin Lee**
*Dept. of Computer Science, Soonchunhayng University
**Dept. of Computer Software Engineering, Soonchunhyang University

요 약

전 세계적으로 클라우드 서버의 가상화 기술이 중요해졌다. VM에서 사용되는 데이터 특징을 기준으로 에너지를 효율적으로 사용하기 위한 연구가 지속적으로 연구되고 있다. 그러나 접속자가 적거나, 특정 시간에만 접속이 되는 경우에도 웹 서버를 작동하기 위하여 VM를 실행되고 있다. 그에 따라 VM 자원 낭비가 되고 있다. 로드 밸런서를 통해 접속 요청이 없는 웹 서버는 동작시키지 않음으로써 자원 효율을 높이고자 한다. 그에 따라 본 논문에서는 에너지 효율이 높은 로드 밸런서를 설계하였다.

1. 서론

전 세계적으로 스마트 기기의 보급과 빅데이터의 활용, 사물인터넷의 확산 등으로 인하여 클라우드 서버의 가상화 기술이 중요해졌다. 클라우드 컴퓨팅을 이용하여 IT 자원을 할당 및 배치하여 시스템을 사용할 수 있게 해준다. VM(Virtual Machine)의 CPU 사용률과 메모리 사용량을 고려하여 자동으로 VM을 추가 및 삭제하는 연구[1-3]이 진행되고 있다. 또한 VM에서 사용되는 데이터의 특징을 기준으로 에너지 효율적으로 관리하는 연구[4]가 진행되었다. 그러나 접속자가 적은 경우나, 특정 시간에만 접속이 되는 경우에는 실제 접속이 없더라도 VM이 실행되므로써 자원이 낭비되고 있다. 본 논문에서는 웹 서버가 작동이 필요로 할 경우에만 작동시켜 에너지 효율이 높은 로드밸런서 설계하고자 한다.

2. 관련연구

2.1 로드 밸런서

로드 밸런서는 웹 서버의 IP 주소를 사용하며 웹 서버의 모든 통신을 로드 밸런서를 통해 연결이 된다. 로드 밸런서는 하나 이상의 웹 서버에 연결이 되어 있으며 사용자 세션과 각 서버의 로드에 따라 다른 웹 서버에 전달하여 처리한다. 처리 속도를 비교하면 소프트웨어 로드밸런서에 비해 하드웨어 로드밸런서가 더 빠르다. 하드웨어 로드밸런서의 경우 비용 문제로 인 소프트웨어 기반 로드밸런스를 사용하기도 한다. 또한 하드웨어 로드밸런스는 제한적인 기능만 사용가능한 단점이 존재한다. 로드 밸런서는 서버 부하가 많아지거나 장애 대응을 할 수 있도록 지원한다. 일반적으로 많이 사용하는 소프트웨어 로드밸런서는 nginx와 haproxy가 있다. nginx는 다양한 기능을 제공하는 웹 서버이다. 본래의 목적은 웹 서버이나 리버스 프록시, 로드 밸런스, 메일 프록시 등 다양한 기능을 제공함으로써 리버스 프록시 서버나 로드 밸런서로도 많이 사용하고 있다. haproxy는 L4, L7 과 같은 하드웨어 로드밸런서를 대체하기 위한 오픈 소스 솔루션이다. 그러나 nginx와 haproxy는 Docker의 API를 사용할 수 없는 단점으로 인해 미리 컨테이너를 생성하여 사용한다. 그렇기 때문에 접속자가 없더라도 컨테이너가 실행 중이여야 하는 문제점이 있다.

3. 로드밸런서 설계

3.1 고려 사항

에너지 효율을 높이기 위하여 접속 요청이 없는 웹서버가 수 초 이내에 동작을 할 수 있어야 한다. VM을 이용할 경우에는 VM을 생성하는 데에 수분이 걸리며, 중지된 VM을 실행하는데 있어서도 수십 초가 걸린다. 그러나 Docker의 경우 컨테이너를 이미지를 통해 수 초 내에 실행이 가능하다[5]. 그에 따라 본 논문은 Docker를 이용하여 수초 내에 웹 서버가 동작함을 보장하고자 한다. 또한 Docker를 API를 이용할 경우 소켓 통신이 일어나며 그에 따라 성능 지연이 발생한다. 그에 따라 부하분산을 위해 지속적으로 API를 요청하는 것이 아닌 Docker의 호스트

1) 본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업의 연구결과로 수행되었음(IITP-2018-2014-1-00720 & IITP - 2017-2015-0-00403).

의 성능을 예측하여 부하분산이 필요로 한다.

3.2 알고리즘

<표1> 서버 부하 분산 알고리즘

Designed load balancing algorithm

```
1.  function loadBalance(hosts, urlHost, user)
2.  {
3.    if(user['session'][urlHost]['container'] != null)
4.    {
5.       resource = getRequestResource(user[session][urlHost]['image'])
6.       hostName = getHostName(user['session'][urlHost]['container'])
7.      container = user['session'][urlHost]['container']
8.    }
9.    else
10.   {
11.      image = getImageFromURL(urlHost)
12.      container = getUselessContainer(image)
13.      resource = getRequestResource(image)
14.      if(container == null)
15.      {
16.         select = getUselessHost(hosts)
17.         select['resource'] += resource['resource']
18.         container = createContainer(select, image)
19.         user['session'][urlHost]['image']=image
20.         user['session'][urlHost]['container']=container
21.      }
22.      else if(container['resource']+resource['request']
    > resource['maxResource'])
22.      {
23.          select = getUselessHost(hosts)
24.          for(host : hosts)
25.          {
26.           if(host['resource']<resource['maxResource'])
27.           {
28.            if((select['resource']+resource['resource'])
    /                 select['maxResource']              >
    (host['resource']+resource['resource'])              /
    host['maxResource'])
29.                 select = host
30.           }
31.          }
32.          select['resource']+=resource['resource']
33.          container = createContainer(select, image)
34.          user['session'][urlHost]['image']=image
35.          user['session'][urlHost]['container']=container
36.      }
37.   }
38.   container['resource']+=resource['request']
39.   return container
40. }
```

<표1>은 에너지 효율이 좋도록 설계한 부하 분산 알고리즘이다. 이미 특정 웹 서버로 설정된 경우 해당 컨테이너에서 처리할 수 있도록 세션 기능을 이용하여 웹 서버에서 세션 사용에 문제없도록 지원한다. 또한 웹 서버 컨테이너가 작동하고 있지 않는 경우 Docker 호스트를 고르도록 한다. 이를 구현하기 위해 사용률만을 고려하여 Docker 호스트 목록에서 사용률이 가장 낮은 Docker 호스트를 찾는다. 그 이후 Docker 호스트 목록 중 할당이 가능한 것을 찾는다. 할당이 가능한 서버 중 생성되었다고 가정하고 다시 사용률을 계산하여 사용률이 낮은 Docker 호스트에서 컨테이너를 생성한다. 만약에 또한 미리 생성된 컨테이너가 있을 경우 사용률이 낮은 컨테이너에서 처리하고, 컨테이너에서 부하량이 지속적으로 높은 경우 새로운 컨테이너를 생성하여 로드 밸런스한다.

<표2> 동작하지 않는 컨테이너 삭제 알고리즘

Designed delete container algorithm

```
1.  function deleteContainer(containers)
2.  {
3.    minResource=int.MAXNUMBER
4.    for(container : containers)
5.    {
6.      if(container['resource']==0                  &&
    container['checktime']==null)
7.         container['checktime'] = now
8.      else       if(container['resource']==0       &&
    container['checktime']!=null)
9.      {
10.        time = container['checktime']-now
11.        if(time >= 600)
12.        {
13.          host = getHost(container)
14.          image = getImageFromContainer(container)
15.          resource = getRequestResource(image)
16.          host['resource'] -= resource['resource']
17.          deleteContainer(container)
18.        }
19.     }
20.     else
21.        minResource = min(minResource,container['resource'])
22.   }
23.   for(container : containers)
24.   {
```

```
25.        if(container['resource']>=minResource)
26.        {
27.           container['resource'] -= minResource
28.           container['checktime'] = null
29.        }
30.     }
31.  }
```

<표2>은 동작하지 않는 컨테이너를 삭제하는 알고리즘이다. 접속 요청 횟수를 파악하여 사용하지 않는 컨테이너를 찾는다. 정확한 사용량을 파악하기 위해서는 Docker API를 이용을 해야 하나 Docker API는 HTTP 프로토콜로 구현되어있다. 지속적인 Docker API 요청으로 인해 성능 저하가 발생된다. 그에 따라 본 논문에서는 수식으로 실제 접속량을 파악하여 삭제한다. 본 논문에서는 지속적인 호출로 파악할 수 있도록 한다. 단, 정확한 성능을 파악하는 것이 아니기 때문에 실행 주기는 짧을 경우 컨테이너 부하 예측이 실패할 수 있다. 웹 서버의 세션은 기본값으로 10분을 사용하기 때문에 10분 내에 웹 서버 접속이 없는 경우 종료하여 자원 낭비를 막는다.

4. 결 론

본 논문에서는 Docker를 이용한 에너지 효율이 높은 로드 밸런서 설계를 하였다. 설계한 로드 밸런서는 다음의 효과를 갖는다. 첫째, 성능 예측을 통한 로드 밸런서를 제공한다. 이를 통해 Docker API로 생기는 성능 저하문제를 없앤다. 둘째, 요청이 없는 웹 서버를 종료함으로써 에너지 효율을 높인다. 이를 통해, 웹 서버가 종료가 되더라도 수 초 내에 생성하여 로드 밸런스를 제공하며, 사용하지 않는 웹 서버를 종료시킴으로써 에너지 효율을 높인다. 향후 설계한 로드 밸런서를 구현할 계획이다.

참고문헌

[1] Krioukov, Andrew, et al. "Napsac: Design and implementation of a power-proportional web cluster." Proceedings of the first ACM SIGCOMM workshop on Green networking. ACM, 2010.

[2] 함치환 외 5명, "서버 클러스터환경에서 에너지 절약을 위한 학습기반의 서버 전원 모드 제어." 2011년 정보처리학회 춘계학술대회 175-178.

[3] Kaushik, Rini T., et al. "Lightning: self-adaptive, energy-conserving, multi-zoned, commodity green cloud storage system." Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. ACM, 2010.

[4] Dangi, Sanket, et al. "Self tuning energy-aware ensemble model for server clusters." Annual international conference on green information technology (GREENIT 2010), Singapore. 2010.

[5] Asa Shiho 저, 신은화 역, 2016, 완벽한 IT 인프라 구축을 위한 Docker, 정보문화사