붙임1

연구논문/작품 제안서

2020년도 제 1학기

논문/작품	○논문(　　) ○작품(　✔　) ※ 해당란에 체크
제목	Kubernetes Auto-scaling을 통한 동적 자원 할당
GitHub URL	https://github.com/berta2708/graduation-thesis
팀원명단	서범수 (인) (학번: 2015312159) 박은찬 (인) (학번: 2017312838)

2020년 03월 25일

지도교수 : 김 영 훈 서명

Abstract

쿠버네티스는 컴퓨팅 자원을 관리하며 자원의 활용을 실시간으로 관찰하고 관찰 결과를 바탕으로 컨테이너를 효율적으로 배치하여 클라우드 서비스가 제대로 제공하는 것을 목표로 한다. 하지만 쿠버네티스의 자원할당은 초기에 어플리케이션의 요구에 의해 설정된다. 이로 인해 컴퓨터 자원들이 종종 낭비된다. 우리는 이 점을 인식하고, 실시간 트래픽 양과 workload를 고려한 자원 할당을 ingress와 autoscaler를 통해 실현하여 위 문제점을 개선하고자 한다. 또한, 제한된 자원 속에서 여러 개의 autoscaler 가 자원 요청을 할시 자원 재분배를 통해 서비스간 효과적인 협업이 가능하도록 코드를 수정하고자 한다. 그 후 수정한 코드가 정상적으로 작동하는지 확인할 환경을 임의로 조성한 후 scaling 실행 여부와 그로 인한 서비스 처리 속도 등의 효율성을 검토한다.

서론

얼마전까지는 다수의 어플리케이션을 운영하기 위해서 Hypervisor를 통해 다수의 가상 머신을 만드는 접근이 대다수였다면, 최근에는 가볍고 확장성이 뛰어난 컨테이너 기반의 환경을 구축하는 곳이 많이 생겨나고 있다. 컨테이너는 가상머신과 달리 운영체제 수준에서 가상화를 실시하여 컨테이너를 OS 커널에서 직접 구동하게 된다. 운영체제 커널을 공유하기 때문에 훨씬 가볍고 빠르게 배포할 수 있고, 서버를 코드로 구성하고 관리할 수 있다는 장점도 있다. 컨테이너 가상화의 대표적인 도구는 오픈소스로 배포되고 있는 도커인데, 도커가 보편적으로 사용되고 관리해야 할 컨테이너가 많아지면서 컨테이너 오케스트레이터인 쿠버네티스도 같이 주목을 받고 사용되고 있다.

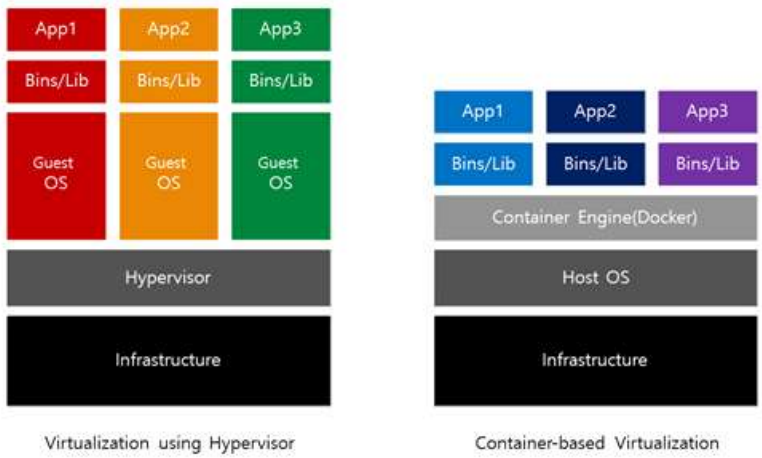

그림 1 Hypervisor 와 Container 기반 Virtualization 의 차이점

Hypervisor와 마찬가지로 컨테이너 환경에서도 역시 자원 관리는 매우 중요한 부분을 차지한다. 쿠버네티스는 어플리케이션의 요구와 실제 자원의 여유분에 따라 각 서비스에게 자원을 할당하게 되는데, 각 서비스에게 정적으로 자원을 할당할 경우 자원이 부족하거나 낭비될 수 있다. 따라서 자원을 모니터링 하며 각 서비스의 필요에 맞게 분배해주는 기능이 필요하고, 이에 쿠버네티스는 CPU 사용량을 실시간으로 모니터링 하며 스케일링을 해주는 Horizontal Pod Autoscaler 와 해당 서비스에 들어오는 트래픽을 생성된 Pod간 분산해주는 ingress load balancer 를 지원한다. 여기서 Pod 란 쿠버네티스가 어플리케이션을 배포할 때 사용하는 가장 기본적인 단위로서, 한 개 이상의 컨테이너를 포함하고 있다.

하지만 쿠버네티스에서 기본적으로 제공하는 Autoscaling과 load balancing은 제한된 자원을 가지고 여러 서비스가 동시에 자원을 요청할 시에 문제가 발생할 수 있다. Autoscaler는 각 서비스마다 존재하기 때문에 각 서비스의 필요에 따라 각 autoscaler 가 master node 에게 자원을 요청하게 된다. 이때 여러개의 서비스에서 동시간대에 자원이 필요하여 자원 요청을 하는 경우, 특정 서비스가 제한된 자원을 독차지 하는 문제가 발생할 수 있다. 따라서 지속적으로 작업을 하는 서비스 (long-lived service) 와 주기적으로 작업을 하는 서비스 (short-lived service) 가 공존하는 경우, 지속적으로 작업을 하는 서비스가 자원을 독식하여 주기적으로 작업을 하는 서비스는 무기한 pending 상태에 빠질 수 있는 것이다.

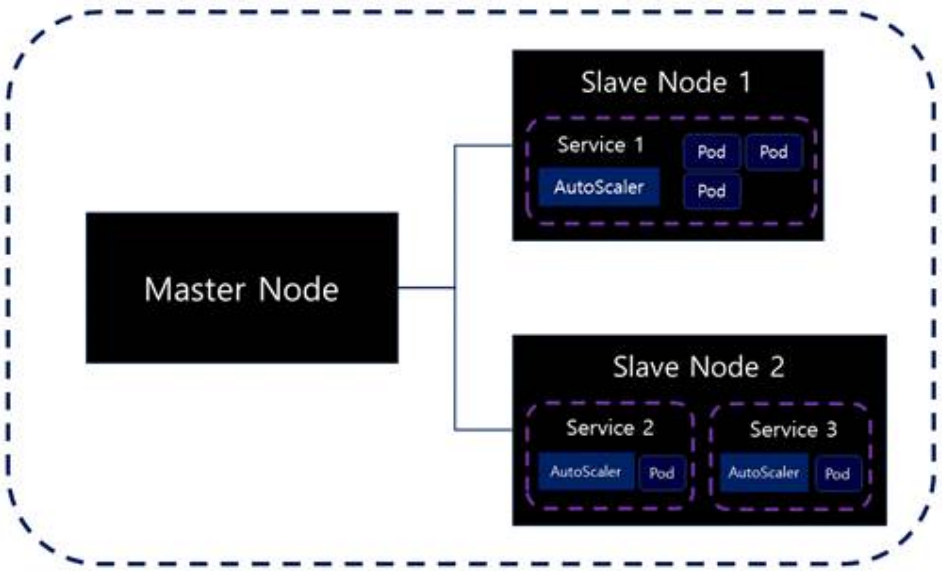

그림 2 쿠버네티스 AutoScaler 구조

우리는 따라서 제한된 자원 속에서 다수의 autoscaler가 동작할 때에 자원 할당을 조율해보려고 한다. 특히, 지속적인 작업을 하는 서비스와 주기적으로 작업을 하는 서비스가 공존하는 시나리오를 가정하고 제한된 자원을 조율하여 관리할 수 있는 시스템을 구축하는 것을 목표로 한다. 이를 통해 제한된 자원을 더 효율적이게 사용하며 쿠버네티스의 autoscaling 기능을 더 많은 환경속에서 활용할 수 있을 것이며, 컨테이너 기반 환경 오케스트레이션을 더 발전시킬 수 있을 것이다.

선행연구 및 기술현황

어플리케이션을 도커를 이용해 컨테이너화 하는 것이 보편적이게 되면서, 컨테이너 관리를 위한 쿠버네티스 역시 많은 연구가 되고 있다. 구글, 마이크로소프트, 아마존, IBM 등 대부분의 클라우드 서비스 제공자들이 컨테이너 서비스에 쿠버네티스를 제공하며, 도커 자체에서도 쿠버네티스 지원을 하는 등 많은 서비스들이 VM 중심 환경에서 컨테이너 기반 환경으로 변화하고 있다. 현재 쿠버네티스 깃헙 레포지토리는 약 64300개의 스타, 25000명의 컨트리뷰터, 그리고 9만개 가까이 되는 커밋을 가지고 있다.

우리가 관심있게 본 쿠버네티스 상에서의 리소스 로드 밸런싱과 스케일링에 관련된 다양한 연구가 있는데, 그 중 2017년도에 한 중국 연구진이 GLOBECOME 2017에 쿠버네티스 상에서 돌아가는 서비스들을 모니터링하며 다이나믹하게 리소스를 할당하는 플랫폼을 소개하였다 [1]. 이 연구는 기존에 CPU 사용률만 이용하여 리소스 관리를 하던 쿠버네티스의 기능을 확장시켜 어플리케이션의 response time 등 QoS를 활용하여 더 효율적인 resource provisioning 알고리즘을 만들고자 하였다. 확장된 알고리즘을 적용시켜 서비스들의 리소스 사용률을 모니터링한 후 Pod Scaler을 이용하여 해당 서비스에 필요한 팟의 개수를 늘리며 로드 밸런싱을 하게된다. 또한, 필요에 따라 유저가 해당 알고리즘을 수정하거나 교체할 수 있도록 설계하여 플랫폼의 확장성이 뛰어난 장점이 있다.

2019년에는 다른 연구진이 쿠버네티스 클러스터상에서 낭비되고 있는 자원을 더 효율적이게 활용하는 방법을 제시하였다 [2]. 저자들에 따르면 쿠버네티스 클러스터는 특정 시간대에 작업량이 몰리는 것을 대비하여 설계되어 있는데, 때문에 평상시의 CPU Utilization 이 40% 이상 넘어가는 일이 드물다. 이러한 낭비를 막기 위해 작업량이 많지 않을 때에 클러스터의 노드 개수를 조정하는 Cluster Scaling Algorithm을 제안하여 CPU utilization을 기존에 대비해 약 29% 높이는 것을 성공하였다. 이 알고리즘을 통해 위해 지속적으로 서비스들의 CPU

사용량을 모니터링 하고 모니터링한 데이터를 기반으로 클러스터에 할당된 노드 개수를 조정하게 된다.

선행 연구들은 등록되어있는 모든 서비스의 자원을 모니터링하며 다이나믹하게 할당하는 것을 연구한 반면에, 우리는 지속적으로 작업을 하는 서비스 (long-lived service) 와 주기적으로 작업을 하는 서비스 (short-lived service)가 공존할 때에 cpu 할당량을 조절하고자 한다. 쿠버네티스에서 현재 지원하는 Horizontal Pod Autoscaler는 CPU 사용량을 기반으로 한 서비스에 할당된 팟의 개수를 조절하게 되는데, 두개 이상의 서비스가 동시에 팟을 요청할 경우 두개의 서비스가 자원을 경쟁하게 된다. 이때, 지속적으로 작업하는 서비스가 CPU를 독점하게 되면 주기적으로 작동하는 서비스의 작업이 계속 pending 상태에 머물게 되고, 이는 결과적으로 효과적이지 못한 리소스 할당이 된다. 따라서 지속적으로 작업을 하는 long-lived service 가 CPU를 대부분을 차지하고 있다가, 주기적인 작업을 하는 short-lived service가 CPU 자원 요청을 하는 경우에 CPU를 효과적으로 재할당 하는 것이 우리의 목표이다. 즉, 이미 할당된 CPU를 필요에 따라 재할당해주고 사용을 끝냈으면 다시 돌려주는 구조를 구현하고 테스트해보는 것이다.

진행 계획 및 구성

쿠버네티스 코드 분석

활용하기로 한 ingress와 autoscaler는 이미 쿠버네티스에 구현되어 있는 상태이므로 우리는 이 코드를 분석해서 수정해야한다.

autoscaler는 각 슬레이브 노드에서 존재하는데 이들간의 소통은 되지 않으며, 마스터노드에서 정보를 취합해 자원을 배정해준다. 그러므로 우리는 마스터 노드와 autoscaler간의 커뮤니케이션과 cpu할당의 메커니즘을 담당하는 코드를 찾아낸 후, 우리가 원하는 방식의 메커니즘으로 수정해야한다.

그 다음은 ingress의 코드를 분석해서 트래픽을 분산하는 방법을 알아내고, 여기서 트래픽 양을 계산 혹은 추적할 방법과 그것을 마스터 노드에게 전달할 방법을 찾아낸다.

최종적으로 마스터 노드와 autoscaler 코드를 workload에 따라서 실시간으로 cpu할당량을 조절할 수 있도록 수정한다. 이 때 배정된 cpu가 감소할 때 진행중이던 thread가 정상적으

로 마저 수행이 가능한지 여부를 파악해야한다. 만약 제대로 실행되지 않고 사라져버린다면 우리는 이 thread를 다른 cpu로 migration을 하는 코드를 추가해야한다.

알고리즘 수행 환경 구성

우리는 위에서 수정한 코드가 정상적으로 작동하는지 확인할 필요가 있다. 그래서 이를 실험할 환경을 구성할 것이다. 실험을 위해서는 클러스터를 먼저 만들어야하는데, 쿠버네티스의 클러스터는 한 개의 마스터 노드와 여러개의 슬레이브 노드들로 이루어진다. 우선 cpu가 8개인 기기에서 실험한다고 가정한다면, 구성될 환경은 다음과 같다.

1. 가상머신을 이용해 리눅스 환경의 한 개의 master 노드와 최소 한 개 이상의 slave 노드를 만든다.
2. cpu를 지속적으로 사용하는 서비스 1을 실행하고 여기에 8개의 cpu를 모두 할당한다.
3. 주기적으로 cpu를 사용하는 서비스 2를 실행시킨다.
4. 마스터 노드의 scaling에 의해 cpu 할당량이 서비스 2의 주기에 따라서 계속 변화한다.

우리는 이 환경에서 수정한 코드에 의해 마스터 노드가 scaling을 정상적으로 수행하는지 여부를 파악한다. 또한 서비스 1이 수행중이던 작업중 cpu 할당량 감소로 인해 작업이 취소된 thread가 발생하는지를 검사한다. 마스터노드의 scaling 수행이 누락되는 thread 없이 원활하게 이루어지는 것을 확인하면, 서비스 1의 처리 속도의 변화 등 우리가 만든 시스템의 효율성을 검증한다.

기대효과

우리는 이 작품을 통해서 일차적으로 Kubernetes의 성능 평가를 기대할 수 있다. 자원이 한정되어있는 상황에서 쿠버네티스의 Autoscaling 협업 방식과 효율성을 평가할 수 있고, 그것을 long-lived service와 short-lived service로 구분되어 있는 시나리오 상에서 더 효율적이게 사용할 수 있게 된다. 뿐만 아니라 Autoscaler를 활용하여 long-lived service 와 short-lived service 가 동시에 효과적으로 동작할 수 있는 환경을 구축하게 되면, 제한된 자원으로 두 종류의 서비스가 공존하는 것이 가능하게 되어 새로운 쿠버네티스 활용법을 제안할 수 있게 된다.

또한, 쿠버네티스 성능 향상을 위한 로드 분산에 관련된 연구를 추가적으로 수행할 수 있을 것이라고 생각한다. 현재 쿠버네티스의 Autoscaler는 서비스의 CPU와 메모리 점유율을 모니터링하여 서비스의 자원을 스케일링을 하게 되는데, Autoscaler를 이용하기 위해선 각 서비스의 threshold 값을 사용자가 일일이 조정해주어야 한다. 때문에 현재는 서비스가 완전히 자동으로 스케일링 된다고 볼 수 없다. 하지만 다른 서비스의 Autoscaler 간 협업 방식을 조정할 수 있게 된다면, 조금 더 자동에 가까운 스케일링 이 가능할 것이다.

마지막으로 컨테이너 환경에서 이미 할당된 자원을 필요에 따라 재할당 하는 것의 가치와 효율성에 대한 판단을 할 수 있을 것이다. 앞서 서술한 대로 작업이 취소되는 thread를 다른 CPU로 migrate 하는 것이 가능한지, 또 가능하다면 overhead가 너무 크진 않은지 등에 대한 판단을 할 수 있게 되어 쿠버네티스 Autoscaling 연구에 기여하는 것을 기대한다.

기타

역할분담

서범수	쿠버네티스 코드 분석, 자원 재할당 시 thread 재할당 여부 파악, 스케일링 알고리즘 구현
박은찬	쿠버네티스 코드 분석, 테스트 환경 구성, 스케일링 알고리즘 구현

비용분석

소프트웨어 기반 작품이고 학교에서 서버가 제공 되는 것을 가정하면 추가적인 비용은 없을 것으로 생각된다.

참고문헌

[1] C. Chang, S. Yang, E. Yeh, P. Lin and J. Jeng, "A Kubernetes-Based Monitoring Platform for Dynamic Cloud Resource Provisioning," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Singapore, 2017, pp. 1-6.

[2] Q. Wu, J. Yu, L. Lu, S. Qian and G. Xue, "Dynamically Adjusting Scale of a Kubernetes Cluster under QoS Guarantee," *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, Tianjin, China, 2019, pp. 193-200.

[3] 김경일, 이규진, 김태현, 최제민, 하수정, 정양재, 진성근, “클라우드 서비스를 위한 쿠버네티스 구조” 한국통신학회지(정보와통신), 35(11), 11-19, 2018