Kubernetes의 kubelet이 관리하는 pod의 수에 따른 성능 영향 분석†

권민수*, 이재학*, 명노영*, 유헌창*, 길준민**††
*고려대학교 컴퓨터학과
**대구가톨릭대학교 IT공학부
e-mail: minsuft19@korea.ac.kr

Performance Analysis According to The Number of Pods Managed by Kubelet in Kubernetes

Min-su Kwon*, Jae-Hak Lee*, Rohyoung Myung*. Heonchang Yu*,
Joon-Min Gil**
*Dept. of Computer Science and Engineering, Korea University
**School of Information Technology Eng., Daegu Catholic University

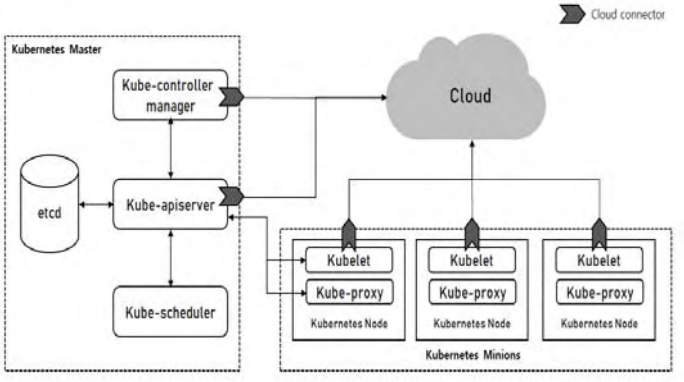
요 약

클라우드 벤더들은 많은 컨테이너를 효율적으로 배포하기 위해 컨테이너 관리 도구를 사용한다. 컨테이너 관리 도구는 Availability, Self-healing, Automated rollouts and rollback 등 여러 기능을 제공한다. 많은 관리 도구 중 Kubernetes는 가장 최소 단위로 컨테이너의 추상적인 모임 pod을 배포한다. pod에 대한 정보는 마스터에서 정의되며 슬레이브 노드에 배포된다. 슬레이브 노드에는 마스터의 명령을 받아 pod을 관리하는 노드 에이전트 kubelet이 생성된다. 하나의 노드에 할당된 자원과 상관없이 kubelet이 관리하는 pod의 개수가 많아지게 되면 작업 중인 pod이 CPU를 훔치는 오버헤드가 발생한다. 따라서 본 논문에서는 pod의 개수에 따른 CPU사용률 실험을 통해 kubelet이 효율적으로 관리할 수 있는 pod의 개수를 분석한다.

1. 서론

기존 클라우드 시장은 가상 머신과 함께 발전해 왔다. 하이퍼바이저를 이용하는 가상화 기술은 컨테이너로 구성된 환경 보다 무겁게 운영된다는 단점이 있다. 컨테이너 가상화 기술은 어플리케이션을 구동할 수 있는 환경을 가상화하는 기술로 가볍고 성능이 좋으며 이식성이 높다[1]. 이런 장점을 토대로 클라우드 시장에서 컨테이너 기술이 빠르게 가상 머신을 대체하고 있다. 때문에 컨테이너를 효율적으로 관리할 수 있는 도구는 필수적이다. 이런 흐름에 맞춰 컨테이너를 효율적으로 관리할 수 있도록 해 주는 오케이스트레이션 플랫폼 Kubernetes[2], Mesos[3], Amazon ECS[4] 등 여러 플랫폼이 개발되었다. 개발된 플랫폼들의 목적은 비슷하지만 특정 목적에 따라 사용하는 플랫폼이 바뀔 수 있다. 본 논문에서는 오케스트레이션 플랫폼 중 현 시장에서 가장 많이 사용되고 있는 Kubernetes 환경[5]에서 실험을 진행하였다.

Kubernetes에서 서비스를 배포하는 가장 작은 단위는 pod이다. Pod은 한 개 이상의 컨테이너 그룹이다. Pod에 있는 컨테이너들은 스토리지와 포트를 공유하고 함께 스케줄링이 된다[2]. Pod들은 마스터에 의해 spec이 정의되고 클러스터 안에 존재하는 노드에 배포된다. 각 노드에는 할당된 여러 pod들을 관리하는 노드 에이전트인 kubelet이 존재한다. kubelet은 노드에 배포되는 에이전트로 Kubernetes 마스터의 API 서버와 통신하며 노드가 수행해야 할 명령을 받아 실행하고 노드의 상태 등을 마스터로 전달하는 역할을 한다[2].



(그림 1) Kubernetes 구조도

Master에서 많은 pod을 하나의 노드에 할당하여 kubelet이 관리하는 pod의 개수가 너무 많아지게 되면 작업중인 pod의 CPU를 훔치는 오버헤드가 발생한다[6]. 작업중인 pod의 리소스를 가져와 사용하는 것은 서비스를

† 이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2016R1D1A3B03933370).
†† 교신저자

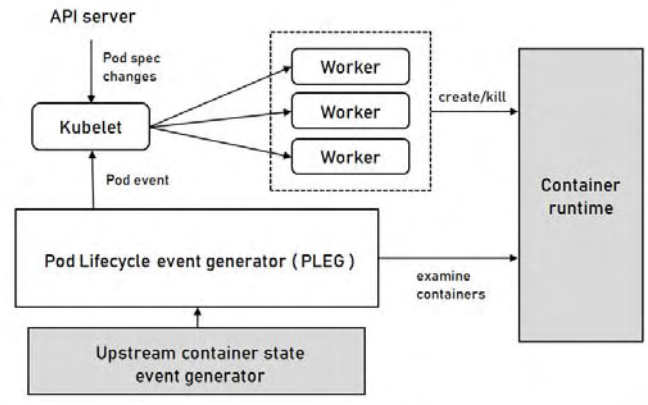
제공받는 클라이언트에게 속도 저하를 일으킬 수 있고 비용을 더 지불해야 하는 상황이 발생한다.

따라서 본 논문에서는 하나의 노드에 pod의 수를 변경하며 배포하는 여러 상황을 만들고 그것에 대한 kubelet의 CPU사용률을 실험한다. 적당한 수의 pod에서 부터 일정 수준을 초과하는 개수를 배포하며 pod의 수에 따른 kubelet 성능 영향을 분석한다.

2. 관련 연구

2.1 Kubernetes

Kubernetes는 구글이 자사 서비스를 위해 개발했던 Borg[7]에서 얻은 운영의 노하우를 넣어 만든 시스템으로 컨테이너 응용 프로그램의 배포, 확장 및 관리를 자동화하는 오픈소스이다. Kubernetes는 크게 마스터와 노드 등 두 개의 컴포넌트로 구성된다. 마스터는 전체 클러스터를 관리하는 역할을 하고 노드는 Kubernetes 위에서 동작하는 워크로드를 호스팅하는 역할을 한다. Kubernetes 마스터는 kube-apiserver를 통해 컨테이너 실행, 제거 가능을 수행하고 kubelet과 통신을 통해 pod을 관리한다. Kubernetes의 4가지 특징은 다음과 같다. 1) 서비스의 리소스 요구사항 및 기타 제약 조건에 따라 컨테이너를 자동으로 배치한다(automatic binpacking). 2) 서비스하고 있는 컨테이너가 종료되거나 사용자가 정의한 health-check가 실패하는 경우 자동 복구된다(self-healing). 3) 수동 또는 CPU 사용량에 따라 자동으로 컨테이너 확장이 가능하다(horizontal scaling). 4) 롤링 업데이트를 진행할 수 있고 잘못 배치한 상황에 대해 롤백을 즉시 사용할 수 있다(automated rollouts and rollbacks).



(그림 2) PLEG 구조

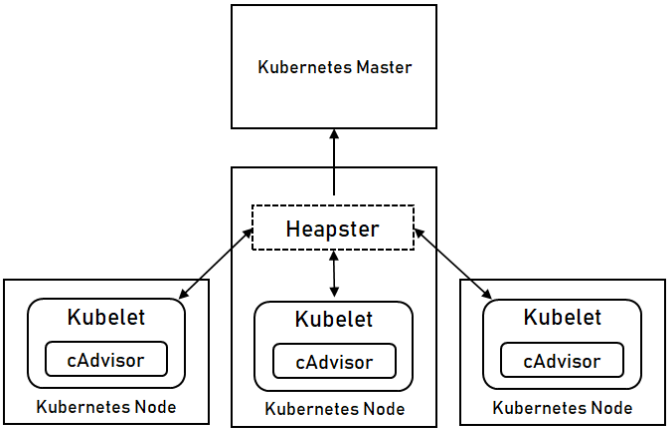
kubelet은 마스터에서 만든 spec에 맞춰 상태를 유지시키기 위해 각 pod의 상태 정보를 가지고 있어야 한다. 이런 상태 정보를 유지하기 위해 kubelet이 pod을 polling 하는 작업을 한다. 주기적으로 polling하는 작업은 pod의 수가 증가함에 따라 무시할 수 없는 오버헤드가 발생한다. 현재 버전에서는 polling 하는 작업의 오버헤드를 줄이기 위해 PLEG (Pod Lifecycle Event Generator)를 사용한다[8]. PLEG를 활용하는 것은 kubelet이 polling해서 정보를 가져오는 작업과 유사하지만 PLEG를 수행하는 싱글 쓰레드를 통해 컨테이너의 상태를 확인하여 kubelet이 polling 하는 작업보다 오버헤드를 줄이는 효과를 볼 수 있다.

3. 실험 환경

본 실험은 Kubernetes 마스터 한대와 minion에 속하는 노드 한 대에 배포하여 실험을 진행한다. 노드는 총 2대이지만 pod의 배포는 하나의 노드에만 배포하여 실험을 진행한다. 마스터의 CPU 사양은 Intel® CoreTM i5-7400 CPU @ 3.00GHz × 4, disk는 250 GB이고 노드의 CPU 사양은 Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz x 4, disk는 500GB이다.

Kubernetes의 노드에서 kubelet의 pod의 생성 개수가 default로 110개로 제한되어 있다. kubelet 설정 yaml 파일을 수정하여 pod 생성 제한을 200으로 늘리고 60개부터 20개씩 늘리며 180개 까지 pod을 생성하는 시나리오를 진행한다. 180개 pod 이하의 개수 생성은 모두 8분 이내로 생성되는 것을 확인하여 8분을 기준으로 kubelet이 사용하는 CPU 사용률을 분석한다.

성능을 분석하기 위해 Kubernetes에서 제공하는 모니터링 도구인 heapster와 dashboard를 사용한다. heapster는 kubelet 설치 바이너리 파일에 포함되어 있는 cAdvisor를 이용해 pod의 상태정보를 수집한다. cAdvisor는 컨테이너 리소스 사용률을 수집하는 오픈소스이다. 이것과 heapster를 연결시키고 노드에 마스터의 dashboard와 연결하여 몇 가지 인증 절차를 거쳐 실행하면 마스터와 함께 구성된 노드들의 상태들을 모니터링 할 수 있다.

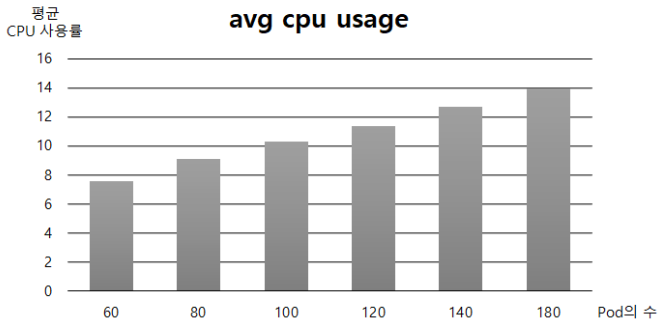


(그림 3) 모니터링 구조

4. 실험 및 분석

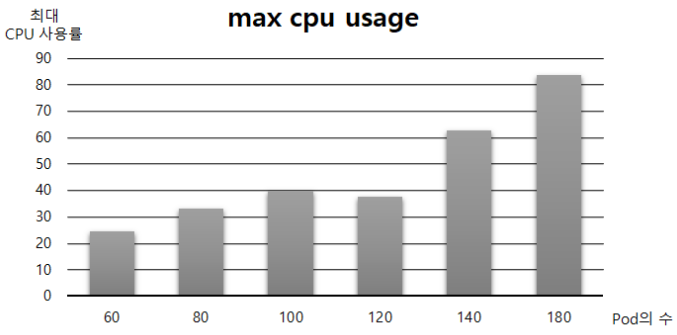
실험은 Kubernetes 마스터에서 하나의 노드에 pod을 생성하며 진행하였다. 노드에는 기본적인 kubelet, kube-proxy 등 기본적인 컴포넌트들을 제외하고 배포한 pod은 없는 상태에서 진행하였다. 처음 생성부터 pod 상태의 정보를 수집하는 과정에 대한 성능을 분석하였다. 진행하는 실험의 모든 상황들에 대한 pod 생성되는 시간은 8분 이내 인

것을 확인하였다. 따라서 실험은 8분의 시간을 기준으로 진행하였다.



(그림 4) Pod 수에 따른 평균 CPU 사용률 분석

(그림 4)에서는 pod의 개수에 따라 kubelet 서비스가 사용하는 평균 CPU 사용률을 분석하였다. x축은 kubelet이 관리하는 pod의 개수이고 y축은 관리하는 pod의 수에 대한 kubelet이 사용하는 평균 CPU 사용률이다. 단위는 모니터링 도구에서 보여주는 % 사용률이다. kubelet이 관리할 pod의 개수가 많아지면서 kubelet 서비스의 CPU 사용률이 어느 정도 일정한 비율로 증가하는 것을 볼 수 있다. 평균 사용률만 보면 180개의 pod을 사용한다고 해서 오버헤드가 발생한다는 것은 알기 어렵다. 그래서 각 pod의 개수에 따라 최대로 사용하게 되는 CPU 사용률을 분석하였다.



(그림 5) CPU 최대 사용률 분석

(그림 5)에서 x축은 pod의 개수이고 y축은 해당 노드에서 순간 kubelet의 최대 CPU 사용률이다. kubelet이 작업 중인 pod의 CPU를 뺏어오는 상황은 많은 CPU 자원이 필요해서 이다. 그래서 각 pod의 개수에 따른 kubelet 서비스의 최대 CPU 사용률을 분석하였다. pod 60개부터 120개 까지 생성할 때 kubelet의 최대 CPU 사용률은 큰 차이가 없다. 100개의 pod보다 120개의 pod 최대 CPU 사용률이 조금 줄어드는 부분은 사용률 측정 주기가 1초로 어느 수준까진 정확하지만 조금의 오차가 있을 수 있다(기본적으로 pod 개수에 따라 증가하는 것으로 예상). 하지만 140개를 생성할 때 최대 CPU 사용률이 60%이상으로 증가하였다. pod 180개를 만들 경우에는 80%이상의 최대 CPU 사용률을 보여 준다. 이런 상황이 발생하게 되면 노드에서 너무 많은 CPU 자원을 pod 관리에 사용하는 상황이 발생한다. 따라서 작업 중인 pod의 CPU 자원을 뺏어오는 상황이 발생할 수 있다.

기존에 기본 값으로 설정 되어 있는 110개보다 더 많은 pod을 설정하여 관리하더라도 평균 cpu 사용률의 증가 말고는 큰 오버헤드를 볼 수 없었다. 본 실험을 통해서는 120개까지 생성을 하더라고 kubelet이 큰 차이 없이 효율적으로 관리할 수 있다고 있다.

5. 결론

본 논문에서는 클라우드 시장에서 많이 사용되고 있는 컨테이너 관리 도구인 Kubernetes를 사용하였다. Kubernetes 클러스터에 속해 있는 하나의 노드에서 pod을 관리하는 노드 에이전트인 kubelet에 대한 성능 분석을 하였다. kubelet은 시스템 성능에 직접적인 영향을 미친다. 그래서 kubelet이 효율적으로 동작할 수 있는 설정을 해주는 것은 중요하다.

Kubernetes 노드의 kubelet이 관리하는 최대 pod의 수에 대한 설정은 기본 값으로 110으로 설정되어 있다. 기본 값 이상인 120개를 관리할 때도 큰 오버헤드가 발생하지 않았다. 환경에 따라 다를 수 있겠지만 최대 pod 기본 설정을 수정하여 하나의 노드에서 적은 오버헤드로 좀 더 많은 pod을 관리할 수 있다. 향후에는 PLEG를 통해 kubelet이 pod 관리 방법을 분석하고 그것보다 더 많은 pod을 관리할 때에도 오버헤드를 줄일 수 있는 방법을 연구할 것이다.

참 고 문 헌

[1] https://www.docker.com/
[2] https://kubernetes.io/
[3] http://mesos.apache.org/
[4] https://docs.aws.amazon.com/AmazonECS/atest/developeguide
[5] https://www.g2crowd.com/categories/container-orchestration
[6] Mastering Kubernetes: Master the Art of Container Management by Using the Power of Kubernetes, 2nd Edition
[7] A Verma, L Pedrosa, M Korupolu, D Oppenheimer, E Tune, J Wilkes Google Inc. Large-scale cluster management at google with Borg. EuroSys '15 Proceedings of the Tenth European Conference on Computer Systems Article No. 18
[8]https://github.com/fabric8io/kansible/blob/master/vendor/k8s.io/Kubernetes/docs/proposals/pod-lifecycle-event-generator.md