Kubeflow 환경에서 CPU 집약적인 작업을 위한 컨테이너 수에 따른 연산 시간 비교 및 분석

정현승, 강태신, 유헌창, 강지훈
고려대학교 대학원 컴퓨터학과
{okhs0712, teri98, yuhc, k2j23h}@korea.ac.kr

Comparative Analysis of Computation Times Based on the Number of Containers for CPU-Intensive Tasks in the Kubeflow Environment

HyunSeung Jung, Taeshin Kang, Heonchang Yu, Jihun Kang
Dept. of Computer Science and Engineering, Korea University

요 약

머신 러닝의 수요가 증가함에 따라, 머신 러닝 워크플로우의 배포 수요도 증가했다. Kubeflow를 통해 머신 러닝 배포를 편리하게 할 수 있으며, Kubeflow Pipelines에서는 하나의 작업을 여러 컨테이너로 분산시켜서 연산하는 것이 가능하다. 하지만 컨테이너 수를 많이 늘릴수록 반드시 성능이 향상되는 것은 아니다. 따라서, 본 연구에서는 성능 향상의 한계를 제공하는 원인을 분석하기 위해서, Kubeflow에서 CPU 집약적인 작업을 여러 컨테이너로 분산시켜서 연산을 수행하였다. 컨테이너 수에 따른 연산 완료 시간을 비교 및 분석한 결과, 컨테이너 수가 증가할수록 연산 속도 향상이 빨라지나, 어느 시점을 지나면 속도가 다시 완만하게 줄어드는 현상을 확인하였다. 이는 리소스 제한으로 인해 모든 컨테이너가 동시에 스케줄링 되지 못한 것이 가장 큰 원인으로 분석하였다.

1. 서론

최근, 머신 러닝이 학술 연구 영역에서 응용 분야로 발전함에 따라, 정부나 기업에서 많이 사용하게 되면서[1] 머신 러닝을 배포하는 일 또한 늘어났다. 즉, 머신 러닝 모델이 애플리케이션에 통합되는 일이 늘어났지만, 머신 러닝 알고리즘 분야에 능숙한 데이터 과학자일지라도 상용 수준의 애플리케이션 개발에는 능숙하지 않을 수 있다[2]. 이러한 문제는 MLOPS(Machine Learning Operations)라는 개념을 도입하여 해결할 수 있다. MLOPS란, 머신 러닝 제품의 엔드 투 엔드(end-to-end) 개념화, 구현, 모니터링, 배포 및 확장성에 관한 개발 문화뿐만 아니라 모범 사례, 개념 집합과 같은 측면을 포함하는 패러다임이다. MLOps는 개발과 운영 간의 격차를 해소하여 머신 러닝 시스템을 프로덕션화하는 것이 목표다[3]. Kubeflow는 이러한 MLOPS 플랫폼 중 하나다. Kubeflow는 Kubernetes에서 머신 러닝 워크플로우 배포의 편리성을 제공한다. Kubeflow에서 머신 러닝의 실행은 Kubeflow Pipelines(KFP)를 통해 워크플로우를 정의함으로써 이루어진다. KFP는 머신 러닝 워크플로우를 그래프 형태로 실행할 수 있도록 지원한다. 그래프의 각 노드는 워크플로우의 각 작업인 데이터 전처리, 모델 학습이나 평가 등에 해당하며, 각 노드는 하나의 컨테이너로 실행된다. 이때, 모델 학습 등의 연산 작업을 분산 수행할 경우, 분산된 각 연산 작업은 개별적인 컨테이너에서 동시에 수행된다.

일반적으로 하나의 작업에 대해 분산 처리를 수행하면 작업 처리 속도는 빨라진다. 하지만, 컨테이너가 늘어날수록 모든 컨테이너의 실행 및 종료까지 걸리는 시간은 더 늘어나게 된다[4]. 이러한 문제로 인해, 컨테이너 수를 늘려서 분산 정도를 높인다고 해서 연산 완료 시간이 반드시 줄어드는 것은 아니다. 따라서, 본 논문에서는 Kubeflow에서 여러 컨테이너를 통해 분산 연산 수행 시, 성능 향상의 한계를 제공하는 원인을 분석하고자 한다.

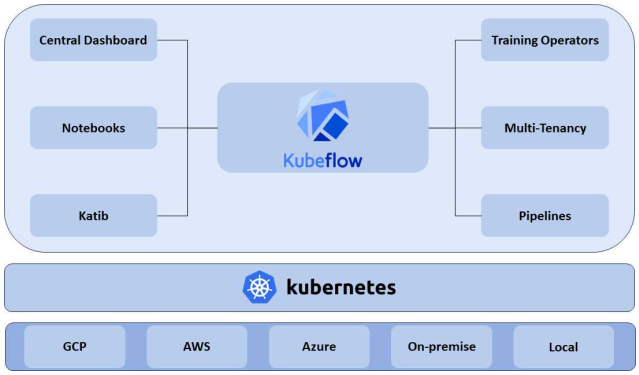
본 논문은 다음과 같은 순서로 구성된다. 2장에서는 Kubeflow 및 KFP의 개념을 소개하며, 3장에

서는 컨테이너 수에 따른 연산 완료 시간을 비교하고 결과를 분석한다. 그리고 4장에서 결론과 추후 진행할 연구에 대해서 다룬다.

2. 관련 연구

2.1 Kubeflow

Kubeflow는 Kubernetes 위에서 머신 러닝 워크플로우에 대해 간편성, 이식성, 확장성을 가지고 배포할 수 있도록 하는 툴킷이다. Kubeflow의 구조는 다음과 같다.



(그림 1) Kubeflow의 구조

그림 1의 가장 하단의 각 요소는 플랫폼 및 클라우드를 나타내며, Kubernetes는 그 위에 설치되고, Kubernetes 위에 Kubeflow가 설치된다. Kubeflow 양쪽의 요소는 Kubeflow에서 제공하는 6가지의 논리적인 구성 요소이며, 각 구성 요소가 제공하는 기능은 다음과 같다. (1) Central Dashboard : Kubeflow의 각 구성 요소에 접근할 수 있는 UI를 제공한다. (2) Kubeflow Notebooks : Pod 내부에서 실행할 수 있는 웹 기반의 개발 환경을 제공한다. (3) Katib : AutoML을 위한 Kubernetes native project이며, 하이퍼파라미터 튜닝, 조기 중지 및 신경 아키텍처 검색 등의 기능을 제공한다. (4) Training Operators : Kubernetes에서 TensorFlow, PyTorch, Apache MXNet, XGBoost, MPI 등의 작업을 분산 혹은 비분산으로 작업을 쉽게 실행할 수 있는 커스텀 리소스를 제공한다. (5) Multi-Tenancy : Kubeflow 클러스터에서 사용자를 그룹으로 격리하는 기능 제공한다. (6) Kubeflow Pipelines(KFP) : 컨테이너를 사용하여 이식 가능하고 확장 가능한 머신 러닝 워크플로우를 구축하고 배포하기 위한 플랫폼이다[5].

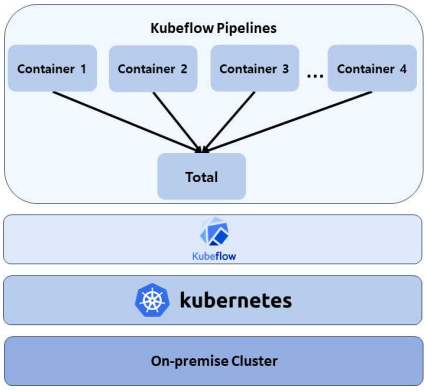
2.2 Kubeflow Pipelines(KFP)

KFP에서는 각 함수를 컴포넌트라는 형태로 작성할 수 있게 해준다. 이때 컴포넌트란, KFP의 빌딩 블록으로, 데이터 전처리, 데이터 변환, 모델 훈련 등과 같은 ML 워크플로우(파이프라인)의 한 단계를 수행하는 독립적인 코드 집합이다. 또한, 컴포넌트는 함수와 유사한 형태이며, 각 컴포넌트에 대해 정의한 내용은 컨테이너 내에서 실행된다.

KFP에서는 워크플로우의 컴포넌트들이 그래프 형태로 결합되어 나타난다. 파이프라인은 머신 러닝 워크플로우에 대한 설명이며, 여기에는 워크플로우의 모든 컴포넌트 및 각 컴포넌트가 그래프 형태로 서로 어떻게 연관되어 있는지가 포함된다. 또한, 파이프라인 구성에는 파이프라인을 실행하는 데 필요한 입력인 매개변수에 대한 정의와, 각 컴포넌트의 입력 및 출력이 포함된다. 파이프라인 실행 시, 시스템은 파이프라인의 컴포넌트에 해당하는 하나 이상의 Pod를 실행한다. 이때, Pod는 컨테이너를 실행하고, 컨테이너는 프로그램을 시작한다[6].

3. 컨테이너 수에 따른 연산 완료 시간 비교

3.1 시스템 구성도



(그림 2) 시스템 구성도

실험에 사용한 시스템의 구성도는 그림 2와 같다. 온프레미스 클러스터를 플랫폼으로 사용하였으며, 그 위에 Kubernetes와 Kubeflow를 설치하였다. 실험 환경에 대한 자세한 정보는 3.2절에서 설명한다.

3.2 실험 환경

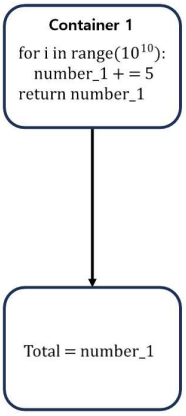
본 연구의 실험은 1개의 마스터 노드와 1개의 워커 노드로 구성된 클러스터에서 진행되었으며, 두 노드의 사양은 동일하다. 하드웨어에 대한 사양은 <표 1>에서 상세히 나타내고 있다.

<표 1> 클러스터 사양

유형	사양
CPU	Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz, 10코어
메모리	DDR4 16GB x 2
디스크	SSD 500GB
OS	Ubuntu 20.04.6

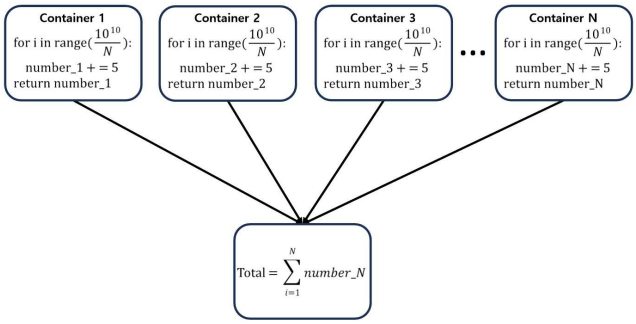
실험에서 사용한 Kubernetes의 버전은 1.24.15이며, Kubeflow의 버전은 1.7이다. 파이프라인 실행을 위한 노트북 서버에는 2개의 CPU 코어와 4GiB의 메모리를 할당하였다. 성능 향상의 한계를 제공하는 주요 원인을 보기 위해서 클러스터 사양에 비해 제한된 환경을 구성하였다.

본 실험은 성능 향상의 한계를 제공하는 원인을 찾는 것이 목표이므로, 워크플로우로 단순한 CPU 집약적인 작업을 사용하였다. CPU 집약적인 작업은 단순 덧셈을 100억번 수행하는 식으로 구현하였다.



(그림 3) 단일 연산 방식

실험을 위해 구현한 단일 연산 방식의 파이프라인은 그림 3과 같으며, 1개의 컨테이너에서 CPU 집약적인 작업인 단순 덧셈을 총 100억번 수행한다. 그다음 과정인 Total=number_1를 통해 단순하게 결과를 반환하며, 이 과정이 가지는 의미는 후술할 분산 연산 방식에서 자세히 다룬다.

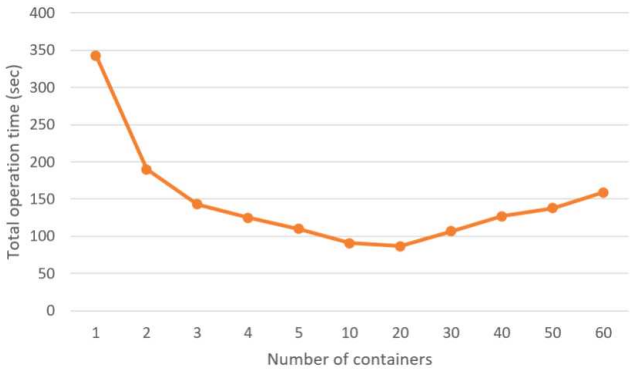


(그림 4) 분산 연산 방식

그림 4에서는 분산 연산 방식을 나타내며, N개의 컨테이너에서 단순 덧셈을 총 100억/N번 수행한다. 각 분산 연산 결과를 병합해야 단일 연산 방식과 연산 결과가 같아지므로, 이를 위해 Total을 구하는 연산을 추가하였다. 해당 연산도 마찬가지로 컨테이너를 통해 수행하므로, 실험 결과에 영향을 미칠 수 있다. 따라서, 실험 결과의 형평성을 위해 해당 과정을 단일 연산에도 추가하였다.

3.3 실험 결과

총 연산 시간은 파이프라인의 모든 실행이 끝난 후, UI를 통해 확인할 수 있는 Duration을 기준으로 측정하였다. 그림 5는 단일 연산 및 분산 연산했을 때 소요된 총 연산 시간을 나타낸다.



(그림 5) 단일 및 분산 연산의 총 연산 시간

컨테이너 수가 1개인 단일 연산 수행 시, 343초가 소요되었다. 컨테이너 2개로 분산해서 수행 시, 단일 대비 1.8배 정도 빨라져서 190초가 소요되었다. 컨테이너 3개로 분산했을 경우에는 단일 대비 2.4배 정도 빨라져서 143초가 걸렸다. 이는 분산 처리로 인해 속도 향상이 나타났다는 것을 나타낸다. 컨테이너 20개까지는 속도가 지속적으로 빨라지긴 하지만, 컨테이너 10개와 20개에서의 빨라진 정도는 각

각 91초인 3.77배와, 87초인 3.94배이다. 즉, 빨라지는 정도가 컨테이너 수가 늘어날수록 줄어들고 있다. 심지어 그 이후로는 빨라지는 정도가 점점 더 줄어든다는 것을 볼 수 있다.

이에 대한 원인 분석은 다음과 같다. 컨테이너를 30개 실행할 때, 가장 먼저 실행된 컨테이너와 가장 늦게 실행된 컨테이너의 시간 간격은 59초이다. 즉, 제한된 리소스로 인해 스케줄러가 동시에 30개의 컨테이너를 모두 실행할 수 없어서 시간 간격이 발생하였다. 시간 간격은 컨테이너 수가 늘어날수록 더 커지게 되고, 결과적으로 속도 향상에 한계가 나타나게 된다.

4. 결론 및 향후 연구

본 연구의 실험은, Kubeflow에서 CPU 집약적인 작업을 여러 컨테이너로 분산하여 연산할 때, 컨테이너 수가 늘어날수록 속도 향상에 한계를 제공하는 원인을 분석하기 위해 진행하였다. 실험 결과, 컨테이너 수가 늘어날수록 리소스의 제한으로 인해 스케줄러가 동시에 모든 컨테이너를 실행하지 못하게 되고, 결과적으로 속도 향상에 있어서 한계가 나타나게 된다. 즉, 속도 향상의 한계가 나타나는 가장 큰 원인은 기존의 스케줄링 정책이다. 따라서, 향후 연구에서는 기존의 스케줄러를 개선하거나 새로운 스케줄링 알고리즘을 제안하여 리소스를 효율적으로 할당하고 시스템의 성능을 향상시키는 방향으로 진행할 것이다. 이를 통해 단순 CPU 집약적인 연산에서 더 나아가서, 머신 러닝 워크플로우의 분산 연산 성능을 최적화하는 것이 목표이다.

사사

이 논문은 2023년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (2022R1I1A1A01063551)

이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2022-0-01198, 융합보안대학원(고려대학교))

참고문헌

[1] Paleyes, Andrei, Raoul-Gabriel Urma, and Neil D. Lawrence. "Challenges in deploying machine learning: a survey of case studies." ACM Computing Surveys 55.6 (2022): 1-29.

[2] Help! My Data Scientists Can't Write (Production) Code 2019, https://insidebigdata.com/2019/08/13/help-my-data-scientists-cant-write-production-code/

[3] Kreuzberger, Dominik, Niklas Kühl, and Sebastian Hirschl. "Machine learning operations (mlops): Overview, definition, and architecture." IEEE Access (2023).

[4] Amaral, Marcelo, et al. "Performance evaluation of microservices architectures using containers." 2015 ieee 14th international symposium on network computing and applications. IEEE, 2015.

[5] Kubeflow, https://www.kubeflow.org

[6] Kubeflow Pipelines, https://www.kubeflow.org/docs/components/pipelines/