쿠버네티스에서 ML 워크로드를 위한 분산 인-메모리 캐싱 방법

[1] 윤동현, [2*]송석일

Distributed In-Memory Caching Method for ML Workload in Kubernetes

[1]Dong-Hyeon Youn, [2*]Seokil Song

요 약

이 논문에서는 기계학습 워크로드의 특징을 분석하고 이를 기반으로 기계학습 워크로드의 성능 향상을 위한 분산 인-메모리 캐싱 기법을 제안한다. 기계학습 워크로드의 핵심은 모델 학습이며 모델 학습은 컴퓨팅 집약적 (Computation Intensive)인 작업이다. 쿠버네티스 기반 클라우드 환경에서 컴퓨팅 프레임워크와 스토리지를 분리한 구조에서 기계학습 워크로드를 수행하는 것은 자원을 효과적으로 할당할 수 있지만, 네트워크 통신을 통해 IO가 수행되야 하므로 지연이 발생할 수 있다. 이 논문에서는 이런 환경에서 수행되는 머신러닝 워크로드의 성능을 향상하기 위한 분산 인-메모리 캐싱 기법을 제안한다. 특히, 제안하는 방법은 쿠버네티스 기반의 머신러닝 파이프라인 관리 도구인 쿠브플로우를 고려하여 머신러닝 워크로드에 필요한 데이터를 분산 인-메모리 캐시에 미리 로드하는 새로운 방법을 제안한다.

Abstract

In this paper, we analyze the characteristics of machine learning workloads and, based on them, propose a distributed in-memory caching technique to improve the performance of machine learning workloads. The core of machine learning workload is model training, and model training is a computationally intensive task. Performing machine learning workloads in a Kubernetes-based cloud environment in which the computing framework and storage are separated can effectively allocate resources, but delays can occur because IO must be performed through network communication. In this paper, we propose a distributed in-memory caching technique to improve the performance of machine learning workloads performed in such an environment. In particular, we propose a new method of pre-caching data required for machine learning workloads into the distributed in-memory cache by considering Kubflow pipelines, a Kubernetes-based machine learning pipeline management tool.

Keywords: *Caching, Kubernetes, Kubeflow, Pipeline, Machine Learning, Deep Learning*

[1] *한국교통대학교 컴퓨터공학과 석사과정(ydh4908@ut.ac.kr)*
[2] *한국교통대학교 컴퓨터공학과 교수, 교신저자(sisong@ut.ac.kr)*

Received: Dec. 17, 2022, Revised: Jan. 29, 2023, Accepted: Aug. 23, 2023

I. 서론

클라우드 기반 ML(Machine Learning) 시스템은 컴퓨팅 자원을 효율적으로 사용하기 위해 쿠버네티스(Kubernetes)[1]와 같은 컨테이너 오케스트레이션(Container Orchestration) 기술을 기반으로 컴퓨팅 인스턴스(Instance)와 스토리지를 분리하는 구조를 이용한다. 이 구조는 확장 가능한 자원을 효과적으로 활용할 수 있는 장점이 있는 반면에 스토리지에서 ML 워크로드에 필요한 데이터를 네트워크 IO 를 해야 하므로 지연이 발생한다. 특히, ML 학습 데이터 크기가 커질수록 GPU 활용률이 낮아지게 되어 ML 워크로드의 처리 시간이 더욱더 지연된다[2].

그림 1 은 전형적인 클라우드 환경에서 데이터 저장소와 컴퓨팅 프레임워크를 분리한 구조의 딥러닝 시스템이다. 그림 에서처럼 데이터 레이크(Data Lake)에 HDFS (Hadoop File System)[3], Lustre 파일 시스템[4], AWS S3[5]와 같은 분산 파일 시스템이 위치하고 다양한 컴퓨팅 프레임워크 (Tensorflow[6]나 Pytorch[7]를 위한 GPU 노드 클러스터, Apache Spark[8] 등)가 네트워크를 통해서 연결되어 있다. 딥러닝 시스템에서 전처리 및 학습을 수행하기 위해서는 컴퓨팅 프레임워크에서 필요한 데이터를 필요시 데이터 레이크에서 가져와서 연산을 수행하게 된다. 이때 네트워크를 통해서 데이터를 가져와야 하기 때문에 지연이 발생하게 되며 이는 GPU 활용률을 낮추는 원인이 된다.

[9]에서는 클라우드 환경에서의 딥러닝 시스템은 데이터 IO 가 전체 딥러닝 소요 시간에 85%를 차지한다고 주장한다. 특히, GPU 의 성능이 증가하면 할 수록 딥러닝 작업에서 데이터 IO 가 차지하는 비중은 더욱 높아질 것이며 이로 인한 GPU 활용률은 더욱 낮아지게 된다. 이 문제를 해결하기 위해서 최근 클라우드 환경에서 ML 워크로드 성능 향상을 위한 캐싱(Caching) 기법이 제안되고 있다. DIESEL[10]은 딥러닝 작업 시 많은 수의 작은 파일들을 처리하는데 소요되는 비용이 매우 큰 것을 지적하며 이를 해결하기 위한 방법들을 제안하고 있다. 이 방법에서는 다수의 작은 파일들을 청크(Chunk) 단위로 그룹화하고 파일의 메타데이터는 각 청크의 헤드(Head)에 분리하여 저장하는 방법을 제안하고 있다. 작은 파일들을 접근하거나 셔플(Shuffle) 연산을 수행할 때 청크 단위로 접근하므로 IO 성능이 증가하는 효과를 얻을 수 있다. 또한, DIESEL 에서는 전역 캐시(Global Cache)에 오류가 발생하게 되면 모든 딥러닝 작업에 영향을 미치게 되어 전체적인 딥러닝 성능이 저하되는 것을 막기 위해 딥러닝 작업 단위의 캐싱을 수행하는 방법을 같이 제안하고 있다.

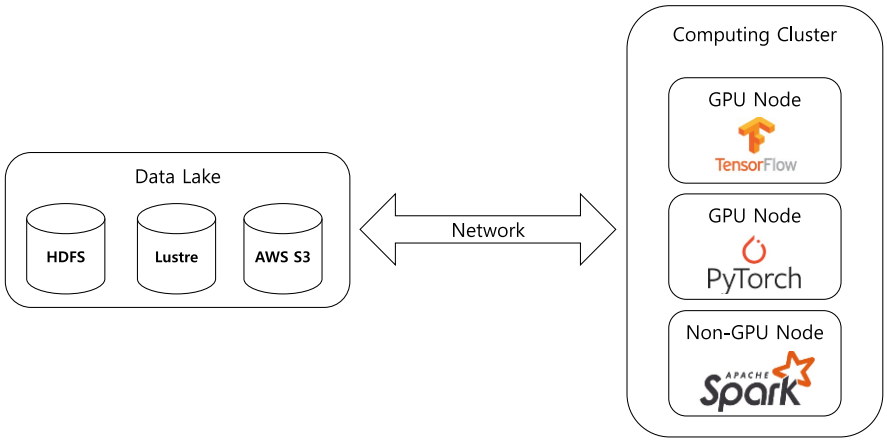


Figure 1. Cloud-based deep learning system with a separation of data storage and computing framework
그림 2. 데이터 저장소와 컴퓨팅 프레임워크가 분리된 Cloud 기반 딥러닝 시스템

커뮤니티 캐시 (Community Cache)[11]는 일반적으로 캐싱을 수행할 때 클러스터 단위로 수행하는데 클러스터 간 데이터 공유 상황에서는 파일 중복 문제와 캐시 적중률(Hit Rate) 저하 문제가 발생할 수 있음을 지적하고 있다. 이를 해결하기 위해서 각 클러스터에 서 수행되는 Apache Spark 및 MapReduce[3]과 같은 컴퓨팅 프레임워크가 DAG(Directed Acyclic Graph)를 생성하는 점을 고려한 새로운 캐싱 기법을 제안하고 있다. 이 방법에서는 각 클러스터의 Apache Spark 및

MapReduce 가 생성하는 DAG 를 분석하여 클러스터간 공유 데이터를 사전에 파악하고 각 데이터가 필요한 시점을 예측하여 캐싱을 수행한다. 또한, 딥러닝 작업에 사용되는 데이터의 크기가 매우 크고 캐싱에 사용되는 주기억장치는 제한되어 있음을 고려하여 데이터의 일부분만 캐싱하는 방법도 같이 제안하고 있다.

Fluid[9]에서는 쿠버네티스 기반의 클라우드 환경에 적합한 캐싱 방법을 제안하고 있다. 특히, Fluid 는 쿠버네티스에서 다양한 종류의 데이터 저장소에 대한 추상화 레이어를 제안하고 있다. 쿠버네티스는 컨테이너 오케스트레이션 도구로 컨테이너 기반의 클라우드 인프라를 구축하고 운영하는데 있어 거의 산업 표준으로 인식되고 있다. 최근 딥러닝 시스템들도 쿠버네티스 기반의 클라우드 환경에서 동작하는 사례가 많이 있다[9]. 쿠브플로우 (Kubeflow)[12]는 쿠버네티스 클라우드 환경에서 딥러닝 작업의 파이프라인을 구축하고 실행 및 관리하는 도구이다. 최근 쿠버네티스와 쿠브플로우를 기반으로 확장성 있는 딥러닝 파이프라인을 구축하고 운영하는 시도가 증가하고 있다.

이 논문에서 조사한 바에 따르면 아직 쿠브플로우의 특징을 고려해서 딥러닝 작업을 위한 캐싱을 수행하는 방법이 제안된 바는 없다. 이 논문에서는 Kubeflow 의 DSL (Domain Specific Language)을 사용하여 정의된 ML 파이프라인을 분석하고 파이프라인에 필요한 데이터를 사전에 분산 인-메모리에 캐싱하는 방법을 제안하고 구현한다. 효과적인 분산 인-메모리 캐싱을 위해서 Apache Ignite[13]을 이용한다. Apache Ignite 는 분산 환경에서 각 노드의 메모리를 이용하여 분산 저장소 및 캐시 기능을 제공한다. 이 논문에서는 Apache Ignite 를 기반으로 ML 파이프라인에서 필요한 데이터를 사전에 캐싱하는 구조를 제안한다. 제안하는 방법은 ML 파이프라인에서 필요한 데이터를 사전에 분산 인-메모리에 캐싱하여 GPU 의 사용률 개선에 기여 할 수 있다.

II. 제안하는 쿠브플로우를 고려한 분산 인-메모리 캐싱 방법

그림 2 는 이 논문에서 제안하는 쿠브플로우의 파이프라인을 고려한 캐싱 시스템의 전체적인 구조를 보여준다. 데이터 저장소는 HDFS, Cassandra, 객체 저장소 등다양한 종류의 저장소를 사용할 수 있다. 쿠버네티스 클러스터에 쿠브플로우가 Tensorflow 와 Pytorch 를 이용해서 딥러닝 작업에 대한 파이프라인을 구축하고 실행하게 된다. 이때 예측 연산자(Prediction Operator) 와 캐싱 연산자(Caching Operator) 는 쿠버네티스의 CRD (Custom Resource Definition)를 이용하여 구현된다. 데이터 저장소의 데이터를 캐싱하기 위해서 Apache Ignite Cache[13]을 이용한다.

예측 연산자는 쿠브플로우 파이프라인 Yaml 파일을 분석하여 캐싱에 필요한 데이터를 수집한다. 데이터를 수집한 후 파이프라인 내의 각 태스크(Task) 실행 순서와 입/출력 데이터 를 예측하여 별도의 DAG 를 생성하여 캐싱 연산자에게 전달한다. 캐싱 연산자는 이를 이용하여 캐싱을 스케줄링 한다.

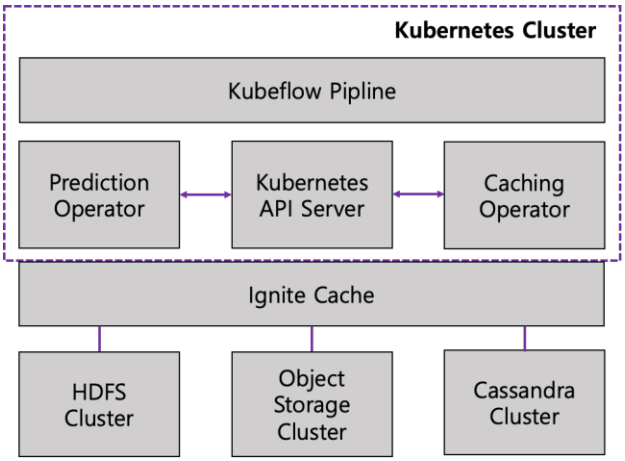
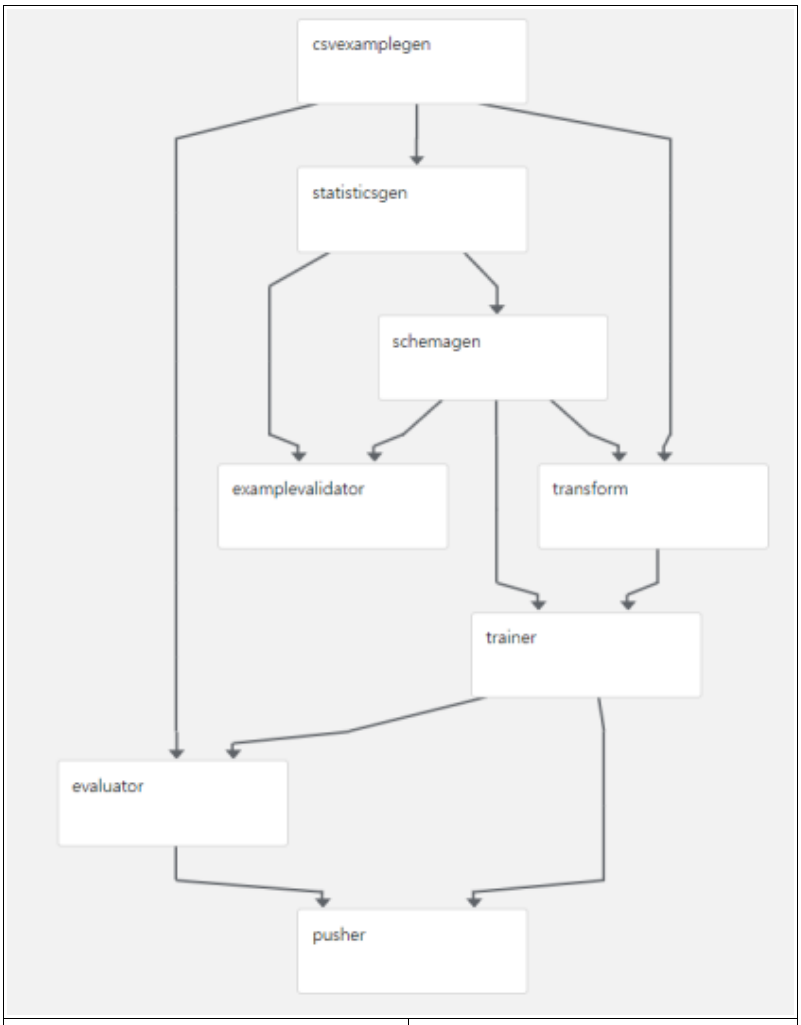


Figure 3. Architecture of the proposed caching system
그림 4. 제안하는 캐싱 시스템 구조




```
metadata:
  generateName: nlp-
spec:
  arguments:
    parameters:
    ...
    - name: raw-text-path
      value: /mnt/text.data
    - name: labels-path
      value: /mnt/labels.data
    - name: clean-text-path
      value: /mnt/clean.data
    ...
    - name: lr-c-param
      value: '0.1'
    - name: tfidf-max-features
      value: '10000'
    ...
    - name: batch-size
      value: '100'
```

Figure 5. Kubeflow pipeline graph and Yaml file example
그림 6. 쿠브플로우 파이프라인 그래프 및 Yaml 파일 예시

그림 3 은 쿠브플로우가 생성한 딥러닝 작업 파이프라인과 파이프라인을 설명하는 Yaml 파일의 일부를 보여준다. 의 예시이다. 이 그림에서 보는 것처럼 쿠브플로우 파이프라인은 딥러닝 작업을 구성하는 각 태스크들의 실행 순서를 나타내고 있다. 또한, Yaml 파일에는 파이프라인을 구성하는 각 태스크의 입력과 출력을 정의한다. 또한, 각 태스크의 학습 파라미터를 보여준다. 즉, 파이프라인 Yaml 파일을 분석하면 파이프라인을 구성하는 각 태스크의 실행순서, 각 태스크의 입출력, 학습 태스크의 경우 학습 파라미터를 추출 할 수 있다. 이를 이용하면 각 태스크의 실행시간 예측이 가능하며 이에 따라 각 태스크에서 필요한 입력을 예측 캐싱을 통해서 제공할 수 있게 된다.

그림 4 는 쿠브플로우 파이프라인 Yaml 파일을 분석한 결과를 저장하기 위해 설계한 데이터베이스 스키마를 보여준다. 주요 내용을 보면 각 파이프라인은 다수의 태스크들로 구성되어 있고 각 태스크에 대한 데이터를 Task 테이블에 저장한다. Task 테이블은 각 태스크의 예측 시작시간 및 예측 종료시간이 핵심 컬럼이다.

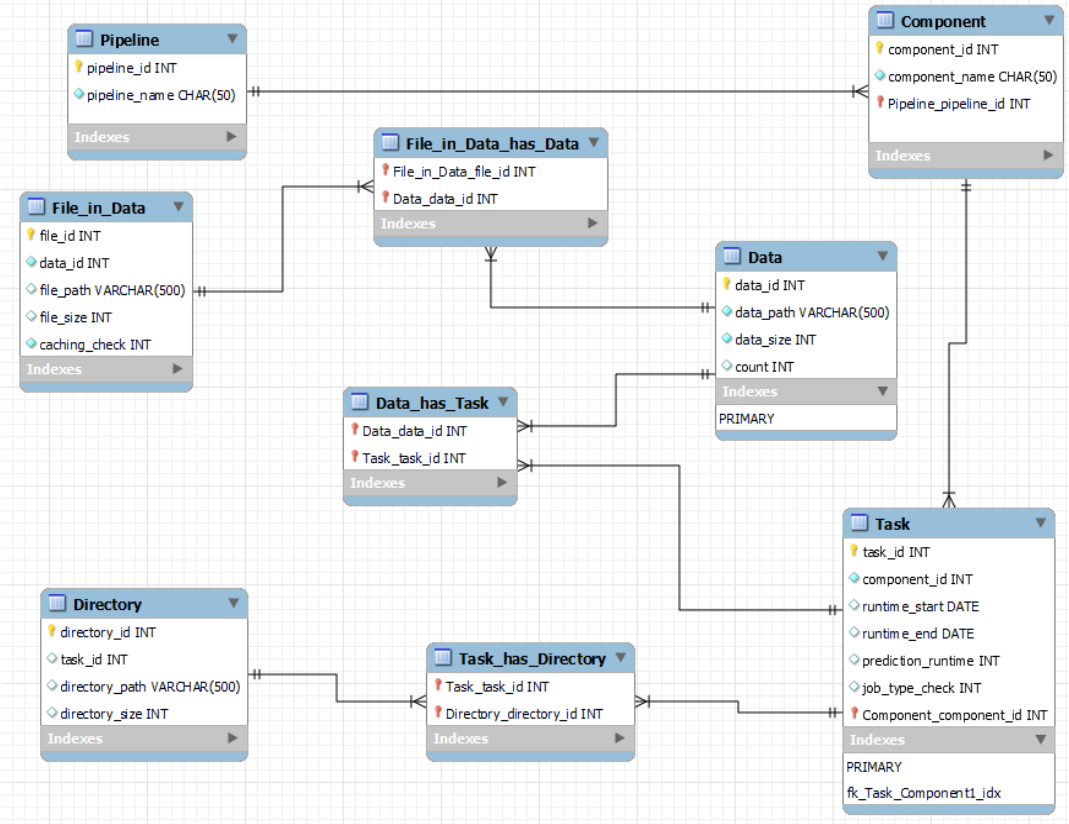


Figure 7. Design of database schema for DAG data
그림 8. DAG 데이터 저장을 위한 데이터베이스 구조 설계

그리고 각 태스크의 입력과 출력 데이터를 저장하기 위해서 Task_has_Director 와 Data_has_task 테이블을 유지한다. Task_has_Directory 는 입력과 출력이 디렉토리 단위인 경우에 이를 표현하기 위한 것이고 Data_has_Task 는 입출력이 파일 또는 다수의 파일인 경우 이를 표현하기 위한 것이다. 또한, 각 입력과 출력은 하나 이상의 태스크에 의해서 공유될 수 있으며 이를 데이터베이스 스키마에 표현하였다.

III. 구현 및 실험

이 논문에서는 제안하는 캐싱 방법을 분산환경에서 구현한다. 표 1 의 구현 환경에서처럼 4 대의 노드로 구성되는 분산 환경을 구축하였다. 1 개의 노드는 쿠버네티스의 마스터(Master) 노드로 하고 나머지 3 개의 노드는 쿠버네티스의 워커(Worker) 노드로 하였다. 분산 인-메모리 캐시를 위해서 Apache Ignite 2.9 를 사용하였으며 파이프라인 구축을 위해서 쿠버네티스는 1.27 버전, 쿠브플로우는 1.5.1 버전을 이용하였다. 또한, 쿠브플로우의 파이프라인을 분석하고 이를 데이터베이스로 구축하는 응용은 Go 언어 3.3 버전을 이용하였다.

Table 1. Implementation environment (HW and SW)
표 2. 구현 환경 (HW 및 SW)

Division		Master (1 Node)	Worker (3 Nodes)
HW	OS	Ubuntu 20.04 LTS	
	SSD	Samsung SSD 980 PRO 2TB	Samsung SSD 970 EVO Plus 1TB
	CPU	Xeon(R) Silver 4214R CPU @ 2.40GHz	Xeon(R) Silver 4112 CPU @ 2.60GHz
	Core	48	60
	Memory	DDR4, 128GB	DDR4, 456GB
SW	Kubernetes	1.27	
	Go Lang	3.3.0	
	Kubeflow	1.5.1	
	Ignite	2.9.0	

이 논문에서는 제안하는 캐싱 방법을 위한 쿠브플로우의 파이프라인 Yaml 파일을 파싱하여 스케줄링에 필요한 데이터를 추출하는 기술을 구현하고 실험한다. 쿠브플로우의 파이프라인을 구현하려면 DSL (Domain Specific Language)[12]로 파이프라인을 기술하고 컴파일해야 한다. 컴파일이 완료되면 파이프라인 Yaml 파일이 생성되고 이 파일은 쿠버네티스의 특정 볼륨에 저장된다. 저장된 쿠브플로우 파이프라인 Yaml 파일을 접근하여 사전에 설계한 데이터베이스 스키마에 맞는 데이터를 추출한다.

```
package main

import (
    "fmt"
    "io/ioutil"

    "gopkg.in/yaml.v2"
)

// 파이프라인 YAML Go Struct로 표현
type Pipelines struct {
    Metadata struct {
        GenerateName string `yaml:"generateName"`
        Annotations  struct {
        } `yaml:"annotations"`
        Labels struct {
            PipelinesKubeflowOrgKfpSdkVersion string `yaml:"pipelines.kubeflow.org/kfp_sdk_version"`
        } `yaml:"labels"`
    } `yaml:"metadata"`
    Spec struct {
        Entrypoint string `yaml:"entrypoint"`
        Templates  []struct {
            Name string `yaml:"name"`
            Dag  struct {
                Tasks []struct {
                    Name         string   `yaml:"name"`
                    Template     string   `yaml:"template"`
                    Dependencies []string `yaml:"dependencies,omitempty"`
                    Arguments    struct {
                        Artifacts []struct {
                            Name string `yaml:"name"`
                            From string `yaml:"from"`
                        } `yaml:"artifacts"`
                    } `yaml:"arguments,omitempty"`
                } `yaml:"tasks"`
            } `yaml:"dag,omitempty"`
            Container struct {
                Args         []interface{} `yaml:"args"`
                Image        string        `yaml:"image"`
                VolumeMounts []struct {
                    MountPath string `yaml:"mountPath"`
                    Name      string `yaml:"name"`
                } `yaml:"volumeMounts"`
            } `yaml:"container,omitempty"`
            Outputs struct {
                Artifacts []struct {
                    Name string `yaml:"name"`
                    Path string `yaml:"path"`
                } `yaml:"artifacts"`
```

Figure 9. Definition of data structure for analyzing pipeline Yaml
그림 10. 파이프라인 Yaml 분석을 위한 자료구조 정의

이미 언급한 것처럼 파이프라인 Yaml 파일 분석 응용 개발은 Go 언어로 하였으며 파이프라인 Yaml 파일의 데이터 추출을 위한 자료구조를 먼저 정의하였다. 그림 5는 이 논문에서 정의한 파이프라인 Yaml 파일 분석을 위한 자료구조를 보여준다. 그림에서처럼 Pipelines 구조체에 Metadata, Spec 구조체가 중첩된 구조이다. 다시 Spec 은 Name, Dag, Container, Outputs 등의 구조체가 중첩된다.

개발한 파이프라인 Yaml 파일 데이터 추출기에 대해 실제 파이프라인 Yaml 파일을 입력으로 하여 정상적으로 캐싱 스케줄링에 필요한 데이터를 추출하는지 확인하였다. 그림 6은 실험에 사용한 파이프라인 Yaml 파일이다. 이 파이프라인은 객체의 연속적인 위치, IoT 센서 데이터 등에 적용할 수 있는 시계열 데이터에 대한 결측치 보정 작업이다.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: jsm-conv-pipeline-
  annotations: {pipelines.kubeflow.org/kfp_sdk_version: 1.8.10, pipelines.kubeflow.org/pipeline_compilation_time:
    pipelines.kubeflow.org/pipeline_spec: '{"description": "Pipeline Test", "name":
      "JSM-Conv-Pipeline"}'}
  labels: {pipelines.kubeflow.org/kfp_sdk_version: 1.8.10}
spec:
  entrypoint: jsm-conv-pipeline
  templates:
  - name: jsm-conv-pipeline
    dag:
      tasks:
      - {name: preprocess, template: preprocess}
      - name: test
        template: test
        dependencies: [preprocess, train]
        arguments:
          artifacts:
          - {name: preprocess-total_x, from: '{{tasks.preprocess.outputs.artifacts.preprocess-total_x}}'}
          - {name: preprocess-total_y, from: '{{tasks.preprocess.outputs.artifacts.preprocess-total_y}}'}
          - {name: train-model, from: '{{tasks.train.outputs.artifacts.train-model}}'}
      - name: train
        template: train
        dependencies: [preprocess]
        arguments:
          artifacts:
          - {name: preprocess-total_x, from: '{{tasks.preprocess.outputs.artifacts.preprocess-total_x}}'}
          - {name: preprocess-total_y, from: '{{tasks.preprocess.outputs.artifacts.preprocess-total_y}}'}
  - name: preprocess
    container:
      args: []
      image: zxc2589/jsm_conv_preprocessing:0.4
      volumeMounts:
      - {mountPath: /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pvc-cf15ffb3-21b7-4978-93b3-5a58cd2a67f3/,
        name: pvc-cf15ffb3-21b7-4978-93b3-5a58cd2a67f3}
    outputs:
      artifacts:
      - {name: preprocess-total_x, path: /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pvc-cf15ffb3-21b7-4978
      - {name: preprocess-total_y, path: /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pvc-cf15ffb3-21b7-4978
    metadata:
      labels:
        pipelines.kubeflow.org/kfp_sdk_version: 1.8.10
        pipelines.kubeflow.org/pipeline-sdk-type: kfp
        pipelines.kubeflow.org/enable_caching: "true"
    volumes:
    - name: pvc-cf15ffb3-21b7-4978-93b3-5a58cd2a67f3
      persistentVolumeClaim: {claimName: jsm-conv}
  - name: test
    container:
      args: [x_train, /tmp/inputs/input-0/data, y_train, /tmp/inputs/input-1/data,
        _model, /tmp/inputs/input-2/data]
      image: zxc2589/jsm_conv_eval:0.4
```

Figure 11. Yaml for time series data imputation pipeline used in the experiment
그림 12. 실험에 사용된 시계열 데이터 결측치 보정 파이프라인 Yaml

그림 7 은 그림 6 의 입력에 대한 분석 결과를 보여준다. 그림에서처럼 사전에 정의한 자료구조에 따라 Yam 파일의 분석결과를 저장하고 있다. 이 논문에서 사용한 결측치 보정 파이프라인에서 전처리, 학습, 테스트 각 태스크에 대한 입력과 출력을 그림 4 의 데이터 베이스 스키마에 따라 정상적으로 추출한 것을 확인 할 수 있다. 저장된 분석결과는 추후 캐싱을 위한 스케줄링 기초 자료로 활용된다. 각 태스크가 어떤 입력을 필요로 하는지 확인이 가능하며, 각 태스크의 입력 데이터 크기, 파라미터, 네트워크 대역폭, 스토리지 대역폭등을 분석하며 각 데이터를 요구하는 시점을 예측하는 것이 가능해진다.

하지만 이 논문의 구현 및 실험에는 한계도 존재한다. 이 논문에서는 결측치 보정이라는 특정 ML 파이프라인에 대한 분석 및 데이터 추출의 정확도를 실험을 통해 확인하였다. 하지만, 실제 더 복잡한 형태의 ML 파이프라인에 대해서는 실험을 통한 동작 확인을 하지 못했다. 향후 이에 대한 추가적인 실험 및 분석 방법의 보완이 필요하다.

```
pipeline_name = jsm-conv-pipeline-


component_name = preprocess
data_path = /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pvc-cf15ffb3-21b7-4978-
data_outps = {preprocess-total_x /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pv

data_outps = {preprocess-total_y /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pv


component_name = test
data_path = /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pvc-cf15ffb3-21b7-4978-
data_inputs = {preprocess-total_x /tmp/inputs/input-0/data}

data_inputs = {preprocess-total_y /tmp/inputs/input-1/data}

data_inputs = {train-model /tmp/inputs/input-2/data}


component_name = train
data_path = /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pvc-cf15ffb3-21b7-4978-
data_inputs = {preprocess-total_x /tmp/inputs/input-0/data}

data_inputs = {preprocess-total_y /tmp/inputs/input-1/data}

data_outps = {train-model /home/dblab/nfs/kubeflow-user-example-com-jsm-conv-pvc-cf15f
```

Figure 13. Output data of the pipeline Yaml parser
그림 14. 파이프라인 Yaml 분석기의 출력 데이터

IV. 결론

이 논문에서는 쿠버네티스 기반의 클라우드 환경에서 쿠브플로우를 이용하여 딥러닝 작업의 파이프라인을 구축하고 운영할때 데이터 IO 를 효과적으로 수행할 수 있는 캐싱 시스템을 제안하고 구현한다. 쿠브플로우가 파이프라인을 운영하기 위해서 생성하는 파이프라인 Yaml 파일을 분석하여 파이프라인의 태스크들 간의 실행 순서, 각 태스크의 입력과 출력 데이터, 학습 태스크의 경우 파라미터를 추출한다. 이를 이용하여 각 태스크가 데이터를 필요로 하는 시점을 예측하고 이를 기반으로 스케줄링을 수행하는 구조를 제안한다. 또한, 파이프라인 Yaml 파일을 분석하여 스케줄링에 필요한 기초데이터를 추출하는 분석기를 실제로 구현하고 정상적으로 데이터를 추출하는지 확인하였다. 향후 연구에서는 이 논문에서 제안하고 구현한 내용을 기반으로 하여 태스크의 데이터 필요시점을 예측하고 캐싱을 수행하는 기술을 개발하고 실험을 수행한다.

V. 감사의 글

This work is supported by the Korea Agency for Infrastructure Technology Advancement (KAIA) grant funded by the Ministry of Land, Infrastructure and Transport (Grant 22AMDP-C161962-02).

VI. 참고문헌

[1] “What is Kubernetes?,” https://kubernetes.io/docs/home/
[2] N. Dryden, R. Böhringer, T. Ben-Nun, and T.R. Hoefler, and T. Bohringer, “Clairvoyant prefetching for distributed machine learning I/O,” in Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC’21), St. Louis, MO, US, 2021, pp. 1–15.
[3] “What is HDFS?,” https://hadoop.apache.org/
[4] “Introducing the Lustre File System,” https://doc.lustre.org/lustre_manual.xhtml
[5] “AWS S3,” https://aws.amazon.com/ko/s3
[6] “Tensorflow Overview,” https://www.tensorflow.org/overview
[7] “Pytorch Documentation,” https://pytorch.org/docs/stable/index.html
[8] “Spark Overview,” https://spark.apache.org/docs/latest/
[9] R. Gu, K. Zhang, Z. Xu, Y. Che, B. Fan, H. Hou, H. Dai, L. Yi, Y. Ding, G. Chen, and Y. Huang, "Fluid: Dataset abstraction and elastic acceleration for cloud-native deep learning training jobs," in Proc. of 2022 IEEE 38th International Conference on Data Engineering (ICDE), Kuala Lumpur, Malaysia, 2022, pp. 2182–2195
[10] L. Wang, S. Ye, B. Yang, Y. Lu, H. Zhang, S. Yan, and Q Luo, "Diesel: A dataset-based distributed storage and caching system for large-scale deep learning training," in Proc. of 49th International Conference on Parallel Processing (ICPP’20), Edmonton, AB, Canada, 2020, pp. 17–20.
[11] M. Abdi, A. Mosayyebzadeh, M.H. Hajkazemi, E.U. Kaynar, A. Turk, L. Rudolph, O. Krieger, and P. Desnoyers, “A community cache with complete information,” in Proc. of 19th USENIX Conference on File and Storage Technologies (FAST’21), 2021, pp. 23–25.
[12] “Kubeflow Introduction,” https://www.kubeflow.org/docs/started/introduction/
[13] “Apache Ignite Documentation Overview,” https://ignite.apache.org/docs/latest/

저자소개

윤동현(*Dong-Hyeon Youn*)

2020 년 2 월 한국교통대학교 컴퓨터공학과 학사
2023 년 2 월 한국교통대학교 대학원 컴퓨터공학과 석사

관심분야 : 클라우드, MLOPS, ML 파이프라인, 쿠버네티스, 빅데이터

송석일(*Seokil Song*)

2000 년 2 월 충북대학교 정보통신공학과(공학석사)
2003 년 2 월 충북대학교 정보통신공학과(공학박사)
2003 년 7 월 ~ 현재 : 한국교통대학교 컴퓨터공학과 교수

관심분야 : 데이터베이스, 스토리지 시스템, 분산 병렬 시스템, 빅데이터, 딥러닝 등